Un intergiciel multi-agent pour la composition flexible d'applications en intelligence ambiante

Mathieu Vallee

Ecole Nationale Supérieure des Mines
SAINT-ETIENNE

N° d'ordre : 514 I.

THÈSE

présentée par

Mathieu VALLÉE

pour obtenir le grade de
Docteur de l'École Nationale Supérieure des Mines de Saint-Étienne

Spécialité : Informatique

UN INTERGICIEL MULTI-AGENT POUR LA COMPOSITION FLEXIBLE D'APPLICATIONS EN INTELLIGENCE AMBIANTE

soutenue à Grenoble, le 21 janvier 2009

Membres du jury

Présidente :	
Amal EL FALLAH SEGROUCHNI	Professeur, Université Pierre et Marie Curie, Paris
Rapporteurs :	
Philippe MATHIEU	Professeur, Université Lille 1, Lille
Sylvain GIROUX	Professeur, Université Sherbrooke, Québec
Examinateurs :	
Gaëlle CALVARY	Maître de conférence, Université Joseph Fourier, Grenoble
David MENGA	Ingénieur R&D, EDF R&D
Directeurs de thèse :	
Olivier BOISSIER	Professeur, ENS Mines, St-Etienne
Laurent VERCOUTER	Maître assistant, ENS Mines, St-Etienne
Fano RAMPARANY	Research Scientist, Orange Labs, Meylan

Spécialités doctorales :	**Responsables :**
SCIENCES ET GENIE DES MATERIAUX	J. DRIVER Directeur de recherche – Centre SMS
MECANIQUE ET INGENIERIE	A. VAUTRIN Professeur – Centre SMS
GENIE DES PROCEDES	G. THOMAS Professeur – Centre SPIN
SCIENCES DE LA TERRE	B. GUY Maître de recherche – Centre SPIN
SCIENCES ET GENIE DE L'ENVIRONNEMENT	J. BOURGOIS Professeur – Centre SITE
MATHEMATIQUES APPLIQUEES	E. TOUBOUL Ingénieur – Centre G2I
INFORMATIQUE	O. BOISSIER Professeur – Centre G2I
IMAGE, VISION, SIGNAL	JC. PINOLI Professeur – Centre CIS
GENIE INDUSTRIEL	P. BURLAT Professeur – Centre G2I
MICROELECTRONIQUE	Ph. COLLOT Professeur – Centre CMP

Enseignants-chercheurs et chercheurs autorisés à diriger des thèses de doctorat (titulaires d'un doctorat d'État ou d'une HDR)

AVRIL	Stéphane	MA	Mécanique & Ingénierie	CIS
BATTON-HUBERT	Mireille	MA	Sciences & Génie de l'Environnement	SITE
BENABEN	Patrick	PR 2	Sciences & Génie des Matériaux	CMP
BERNACHE-ASSOLANT	Didier	PR 1	Génie des Procédés	CIS
BIGOT	Jean-Pierre	MR	Génie des Procédés	SPIN
BILAL	Essaïd	DR	Sciences de la Terre	SPIN
BOISSIER	Olivier	PR 2	Informatique	G2I
BOUCHER	Xavier	MA	Génie Industriel	G2I
BOUDAREL	Marie-Reine	MA	Génie Industriel	DF
BOURGOIS	Jacques	PR 1	Sciences & Génie de l'Environnement	SITE
BRODHAG	Christian	MR	Sciences & Génie de l'Environnement	SITE
BURLAT	Patrick	PR 2	Génie industriel	G2I
CARRARO	Laurent	PR 1	Mathématiques Appliquées	G2I
COLLOT	Philippe	PR 1	Microélectronique	CMP
COURNIL	Michel	PR 1	Génie des Procédés	SPIN
DAUZERE-PERES	Stéphane	PR 1	Génie industriel	CMP
DARRIEULAT	Michel	ICM	Sciences & Génie des Matériaux	SMS
DECHOMETS	Roland	PR 1	Sciences & Génie de l'Environnement	SITE
DESRAYAUD	Christophe	MA	Mécanique & Ingénierie	SMS
DELAFOSSE	David	PR 1	Sciences & Génie des Matériaux	SMS
DOLGUI	Alexandre	PR 1	Génie Industriel	G2I
DRAPIER	Sylvain	PR 2	Mécanique & Ingénierie	SMS
DRIVER	Julian	DR	Sciences & Génie des Matériaux	SMS
FOREST	Bernard	PR 1	Sciences & Génie des Matériaux	CIS
FORMISYN	Pascal	PR 1	Sciences & Génie de l'Environnement	SITE
FORTUNIER	Roland	PR 1	Sciences & Génie des Matériaux	CMP
FRACZKIEWICZ	Anna	MR	Sciences & Génie des Matériaux	SMS
GARCIA	Daniel	CR	Génie des Procédés	SPIN
GIRARDOT	Jean-Jacques	MR	Informatique	G2I
GOEURIOT	Dominique	MR	Sciences & Génie des Matériaux	SMS
GOEURIOT	Patrice	MR	Sciences & Génie des Matériaux	SMS
GRAILLOT	Didier	DR	Sciences & Génie de l'Environnement	SITE
GROSSEAU	Philippe	MR	Génie des Procédés	SPIN
GRUY	Frédéric	MR	Génie des Procédés	SPIN
GUILHOT	Bernard	DR	Génie des Procédés	CIS
GUY	Bernard	MR	Sciences de la Terre	SPIN
GUYONNET	René	DR	Génie des Procédés	SPIN
HERRI	Jean-Michel	PR 2	Génie des Procédés	SPIN
KLÖCKER	Helmut	MR	Sciences & Génie des Matériaux	SMS
LAFOREST	Valérie	CR	Sciences & Génie de l'Environnement	SITE
LI	Jean-Michel	EC (CCI MP)	Microélectronique	CMP
LONDICHE	Henry	MR	Sciences & Génie de l'Environnement	SITE
MOLIMARD	Jérôme	MA	Sciences & Génie des Matériaux	SMS
MONTHEILLET	Frank	DR 1 CNRS	Sciences & Génie des Matériaux	SMS
PERIER-CAMBY	Laurent	PR1	Génie des Procédés	SPIN
PIJOLAT	Christophe	PR 1	Génie des Procédés	SPIN
PIJOLAT	Michèle	PR 1	Génie des Procédés	SPIN
PINOLI	Jean-Charles	PR 1	Image, Vision, Signal	CIS
STOLARZ	Jacques	CR	Sciences & Génie des Matériaux	SMS
SZAFNICKI	Konrad	CR	Sciences & Génie de l'Environnement	SITE
THOMAS	Gérard	PR 1	Génie des Procédés	SPIN
VALDIVIESO	François	MA	Sciences & Génie des Matériaux	SMS
VAUTRIN	Alain	PR 1	Mécanique & Ingénierie	SMS
VIRICELLE	Jean-Paul	MR	Génie des procédés	SPIN
WOLSKI	Krzysztof	CR	Sciences & Génie des Matériaux	SMS
XIE	Xiaolan	PR 1	Génie industriel	CIS

Glossaire :

PR 1	Professeur 1ère catégorie
PR 2	Professeur 2ème catégorie
MA(MDC)	Maître assistant
DR (DR1)	Directeur de recherche
Ing.	Ingénieur
MR(DR2)	Maître de recherche
CR	Chargé de recherche
EC	Enseignant-chercheur
ICM	Ingénieur en chef des mines

Centres :

SMS	Sciences des Matériaux et des Structures
SPIN	Sciences des Processus Industriels et Naturels
SITE	Sciences Information et Technologies pour l'Environnement
G2I	Génie Industriel et Informatique
CMP	Centre de Microélectronique de Provence
CIS	Centre Ingénierie et Santé

Remerciements

Je voudrais profiter de cette page pour remercier certaines personnes qui ont contribué à la réalisation du travail décrit ici, directement ou indirectement :

- Merci à mes parents, ainsi qu'à de plusieurs autres membres de ma famille, pour m'avoir donné cette envie de découvrir et de comprendre sans laquelle je n'aurais pu entreprendre ce travail.
- Merci à ceux qui ont suivi ce travail de près. En particulier, merci à Fano pour m'avoir accueilli et fait partager son enthousiasme pour nos recherches. Merci à Laurent pour m'avoir beaucoup soutenu, particulièrement pendant la rédaction du mémoire. Merci à Olivier pour ses conseils toujours particulièrement pertinents.
- Merci aux membres des laboratoires FT R&D/Orange Labs de Meylan, qui m'ont accueilli et offert un environnement particulièrement riche et motivant. Merci en particulier à Gilles, Anne, Thibault, Matthieu, Stan, Émeric, André, Jean-Paul, avec qui j'ai eu vraiment plaisir à collaborer. Merci à Michel, Alexis et Solène pour leur aide sur les démos, et à Vincent Gimeno sa contribution à nos conditions de travail. Un merci tout particulier à Jérôme, avec qui j'ai eu la chance de partager les réussites et les difficultés de ces années de thèse, et à Rémi, sans qui une bonne partie des résultats présentés ici n'auraient pas été possibles.
- Merci à toute l'équipe SMA de St-Étienne, en particulier pour leurs conseils pour la préparation de la soutenance.
- Merci aux participants du projet Amigo, qui ont largement contribué à l'élaboration de ce travail au travers des échanges et des discussions que nous avons eu ensemble.

Résumé

La notion d'intelligence ambiante découle d'une évolution des systèmes informatiques, qui passent progressivement du statut d'outil au statut d'environnement, dans lequel les utilisateurs se trouvent immergés. Les dispositifs informatiques, de plus en plus petits, diversifiés et interconnectés, se fondent en partie dans l'environnement physique et ouvrent de nouvelles possibilités d'interaction et d'expériences. En particulier, l'environnement domestique se transforme en un environnement attentif, capable d'accompagner les activités de la vie quotidienne.

Cependant, la conception d'applications pour un environnement attentif (les applications attentives) soulève de nombreux défis, en particulier liés à l'*hétérogénéité* des objets communicants, à l'*instabilité* de l'environnement et à la *variabilité* des besoins. Afin de faciliter la conception d'applications attentives, les travaux présentés ici s'intéressent à la notion d'*application composite flexible* : des applications qui fonctionnent en continu, en arrière plan de l'attention, et agrègent des fonctionnalités fournies par les objets communicant, dans le but d'accompagner l'interaction des utilisateurs. Un intergiciel pour la composition flexible d'applications, nommé FCAP, est présenté. FCAP fournit un support générique pour intégrer les fonctionnalités, s'adapter à un environnement instable et ajuster l'équilibre entre l'automatisation et le contrôle par les utilisateurs. Cet intergiciel développe des principes issus de l'architecture orientée-service, du Web sémantique et des systèmes multi-agents, dont les caractéristiques sont particulièrement pertinentes pour les environnements considérés.

FCAP a été implémenté et validé par la réalisation de plusieurs applications attentives. Nos expérimentations indiquent ainsi l'intérêt de l'approche de composition flexible pour faciliter la conception d'applications attentives.

Abstract

The concept of ambient intelligence results from a recent evolution of information systems. While information systems and computers were primarily used as tools, they are now transforming into a kind of environment, which literally surrounds humans in their daily lives. As computer devices become smaller, more diverse and more interconnected, they now become an integral part of our physical environment, thus opening new possibilities for interaction and experiences. More specifically, the domestic environment can progressively become a person-aware environment, capable of assisting and enhancing daily life activities.

However, designing applications for such a person-aware environment (called person-aware applications) raises numerous challenges. Especially, devices are *heterogeneous*, environments are unstable and *evolving*, and users' needs are variable and *imprecise*. In order to facilitate the design of person-aware applications, we focus on the notion of *flexible composite applications* (FCAP) : applications working continuously in the background of our attention with the goal of aggregating functionalities in order to support our activities. We thus investigate a middleware for flexible composition of applications (called FCAP). FCAP provides a generic support for integrating heterogeneous functionalities, adapting applications to an unstable environment and adjusting the balance between automation and end-user control. Our approach builds on principles derived from the fields of service-oriented architecture, semantic Web and multi-agent systems, as each of these fields present relevant properties for our environments.

We have implemented and validated FCAP by implementing several person-aware applications and making them evolve. Our experimentations demonstrate that our approach for flexible composition of application is suitable for improving the design of person-aware applications.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Nous introduisons ce mémoire en présentant tout d'abord le contexte dans lequel s'inscrivent nos travaux. Nous exposons ensuite nos objectifs, avant de détailler l'organisation du mémoire.

1.1 Vers les environnements informatiques attentifs

1.1.1 Les évolutions des systèmes informatiques

Depuis la conception des premiers ordinateurs, la portée et les usages des systèmes informatiques se sont considérablement développés. Ils ont aussi beaucoup évolués et trois tendances se dégagent plus particulièrement : la miniaturisation, la multiplication et l'interconnexion.

L'évolution la plus apparente des systèmes informatiques est la *miniaturisation*. Les premières générations d'ordinateurs occupaient toute une pièce. Un ordinateur personnel peut être disposé sur un bureau. Aujourd'hui, les dernières générations de téléphones mobiles ou de lecteurs multimédia sont eux aussi des systèmes informatiques, qu'il est possible de tenir dans une main. La miniaturisation a notamment permis une spécialisation des usages.

La seconde tendance est la *multiplication* des dispositifs informatiques. Cette multiplication change notamment le rapport entre les personnes et les systèmes informatiques. Alors que plusieurs utilisateurs devaient partager les premiers ordinateurs, l'ordinateur personnel associe un utilisateur à un ordinateur. De plus en plus, nous sommes habitués à interagir quotidiennement avec de nombreux systèmes informatiques.

La troisième tendance concerne les *interconnexions* de plus en plus complexes entre les systèmes informatiques. Pendant longtemps, un système informatique a été conçu comme une entité relativement isolée, ne possédant que des interactions limités avec le reste du monde. Avec l'avènement de l'Internet, les relations deviennent plus complexes et chaque machine est potentiellement connectée à de nombreuses autres machines au travers du réseau. La plupart de ces interactions entre machines sont transparentes pour l'utilisateur.

1.1.2 L'ordinateur du 21ème siècle et le cerveau de l'humanité

Malgré ces évolutions, ou peut-être à cause d'elles, on peut constater que l'utilisation des systèmes informatiques reste complexe et requiert souvent une expertise importante. Dans le monde de la recherche scientifique et technique, ce constat a inspiré l'émergence de visions canalisant l'attention de nombreux travaux. Deux visions sont particulièrement importantes : l'informatique diffuse et le Web sémantique.

Le domaine de l'informatique diffuse (« l'ordinateur du 21eme siècle », (Weiser, 1991)) s'intéresse à la possibilité de fondre les ordinateurs dans l'environnement physique, de manière à faire « disparaître » les systèmes informatiques de l'attention des utilisateurs. Dans ce domaine, la miniaturisation facilite évidemment la disparition physique, en intégrant des ordinateurs dans les objets de la vie quotidienne. L'aspect le plus important est cependant l'exploitation de la multiplicité et des interconnexions entre les systèmes informatiques pour leur permettre de fonctionner de manière naturelle et transparente pour les utilisateurs.

Le domaine du Web sémantique (« le cerveau de l'humanité » (Fensel & Musen, 2001)) s'intéresse à la possibilité de faire évoluer l'Internet vers un système global, au sein duquel non seulement les humains peuvent échanger des informations, mais aussi les machines. Grâce à l'utilisation de langages de représentation des connaissances, il devient aussi possible d'exprimer des informations ayant du sens pour les machines, et d'automatiser ainsi une grande partie des interactions complexes qui sont aujourd'hui nécessaires pour utiliser un système informatique.

La combinaison de ces deux visions fait envisager de nouvelles structures d'interaction au sein du système informatique :

- les *interactions entre les humains et les machines* peuvent devenir plus transparentes, au fur et à mesure que les technologies informatiques et l'environnement physique se confondent.
- les *interactions entre les machines* peuvent devenir plus autonomes, au fur et à mesure que les machines possèdent des capacités d'analyse et de raisonnement plus étendues.
- les *interactions entre les humains* peuvent devenir plus fluides, au fur et à mesure que les contraintes d'espace et de temps disparaissent et que certaines tâches annexes sont prises en charge par les machines. En particulier, l'enjeu est non seulement au niveau des interactions globales (interagir avec l'autre bout du monde), mais aussi au niveau des interactions locales : les tâches prises en charge par des machines doivent libérer le temps et l'attention pour privilégier les interactions directes entre les humains.

1.1.3 Vers les environnements attentifs

Les évolutions précédemment soulignées font apparaître une nouvelle dimension. À l'origine, les systèmes informatiques étaient des outils. On les utilisait pour accomplir une tâche bien précise, en particulier pour automatiser des calculs. De plus en plus, le système informatique est un environnement, c'est-à-dire un ensemble d'éléments qui nous entoure et avec lequel nous sommes en interaction permanente.

Cette nouvelle dimension apparaît particulièrement dans le domaine de l'« intelligence ambiante ». L'intelligence ambiante est ainsi un sous ensemble de l'informatique diffuse, qui se focalise sur l'exploitation des interactions avec son environnement devenu à la fois physique et informatique. Confondue avec l'environnement quotidien, l'interface humain-machine peut devenir transparente et intuitive. Relié au système informatique global, cet environnement peut donner accès de manière simple et immédiate à tout type d'information et de service.

Considéré comme un environnement, le système informatique présente de nouvelles opportunités. L'une d'elle est la notion d'environnement attentif : un environnement capable d'accompagner les activités de la vie quotidienne en interagissant en arrière-plan de l'attention.

1.1.4 Les défis des environnements attentifs

En raison de l'intégration dans la vie quotidienne, les environnements attentifs présentent de nombreux défis, non seulement technologiques, mais aussi économiques et sociologiques.

Les défis techniques sont les plus évidents. Un environnement attentif est un système distribué d'entités hétérogènes entre lesquelles se déroulent des interactions complexes. Des méthodes et des outils pour faciliter la conception, la maintenance et la réparation de tels systèmes sont indispensables. Ces méthodes et ces outils doivent notamment supporter l'interopérabilité des différents dispositifs et la robustesse des applications fonctionnant dans ces environnements. De plus, ces méthodes et ces outils doivent tenir compte de l'ouverture et l'évolution du système, qui constituent une différence majeure par rapport au cadre d'un environnement informatique traditionnel.

Les défis économiques sont aussi importants. Le développement d'environnements attentifs présente une opportunité de création de produits innovants utiles pour l'amélioration des conditions de vie, et comportent donc un potentiel de création d'activité économique. Cependant, les environnements attentifs sont caractérisés par la nécessité de coopération entre de nombreux acteurs issus de domaines différents (informatique, électronique, mais aussi design et architecture). D'autre part, il existe de fortes disparités entre le rythme d'évolution des systèmes de haute technologie (quelques années voire quelques mois) et le rythme d'évolution de l'habitat (plusieurs décennies), qui suppose des méthodes de conception radicalement différentes des approches traditionnelles des produits technologiques. En particulier, la solution consistant à vendre un produit intégré (comme une voiture) n'est pas envisageable. À l'inverse, les opportunités dans le cadre des environnements attentifs semblent plutôt se diriger d'une part sur des dispositifs spécialisés capables de s'intégrer dans un environnement existant, et d'autre part sur des services immatériels exploitant un environnement existant (comme des services d'assistance aux personnes).

Enfin, les environnements attentifs présentent des défis sociologiques. En particulier, l'intégration de technologies complexes dans l'environnement quotidien n'est pas neutre et peut présenter des dangers si elle n'est pas encadrée. En raison de leur complexité, il est en effet très difficile pour un utilisateur non expert d'appréhender le fonctionnement de ces environnements, ce qui peut entraîner un fort sentiment de perte de contrôle et un rejet en masse de ces technologies. De plus, le risque d'exploitation incontrôlée et malhonnête de ces technologies est réel et pose de nombreuses questions concernant le respect de la vie privée

et l'utilisation des informations personnelles.

1.2 Objectif : faciliter la conception d'applications dans les environnements attentifs

1.2.1 Le rôle d'une infrastructure générique pour les environnements attentifs

Pour faire face aux défis posés par les environnements attentifs, nous considérons que la définition d'une *infrastructure générique* pour les environnements attentifs est essentielle. On entend ici par infrastructure un ensemble d'éléments cohérents et inter-connectés dont le rôle est de supporter la structure d'un environnement attentif. Une infrastructure pour les environnements attentifs doit de plus être *générique*, c'est-à-dire qu'elle doit pouvoir être utilisée par différents environnements attentifs, indépendamment de leurs caractéristiques propres. Ainsi, tout comme l'infrastructure du réseau électrique fournit un socle stable et bien défini grâce auquel il est possible de faire fonctionner toute sorte d'appareils électriques, l'infrastructure des environnements attentifs doit fournir un socle à partir duquel il sera possible de mettre en œuvre toute sorte de services innovants.

Une infrastructure générique répond aux défis mentionnés précédemment :

- au niveau de défis techniques, une infrastructure générique permet d'élaborer et d'affiner des méthodes et des outils utilisables dans divers environnements, et ainsi de dégager les caractéristiques essentielles de ces environnements.
- au niveau des défis économiques, une infrastructure générique permet l'intégration de dispositifs conçus par différents acteurs ainsi que la création de services immatériels indépendants d'un environnement particulier.
- au niveau sociologique, l'existence d'une infrastructure générique partagée par les différents acteurs facilite la vérification de l'utilisation des systèmes et augmente la confiance des utilisateurs.

1.2.2 Contribution : étude d'un intergiciel pour la composition flexible d'applications

Nos travaux étudient la réalisation d'une infrastructure générique pour les environnements attentifs sous la forme d'un intergiciel pour la composition flexible d'applications. Notre contribution porte principalement sur trois points :

1. une *étude des problématiques de conception d'applications* dans les environnements attentifs. À partir des problèmes identifiés dans les travaux précédents et des applications attentives que nous avons nous-mêmes réalisées, nous déterminons un ensemble de problèmes auxquels les concepteurs d'applications attentives doivent répondre. Nous organisons ces problèmes au sein d'une grille d'analyse, qui définit trois grandes problématiques de conception des applications attentives.
2. la *définition d'un intergiciel pour la composition flexible d'applications*, destiné à faciliter la conception d'applications attentives grâce à une approche de composition flexible d'application. Nous étudions comment une

telle approche peut prendre en charge une partie des problématiques de conception des applications attentives. Deux aspects de cette contribution sont particulièrement importants. D'une part, cet intergiciel définit et organise les principes et les mécanismes de composition flexible d'application dans les environnements attentifs, en intégrant notamment des approches issues de plusieurs domaines de l'informatique au sein d'un système cohérent. D'autre part, cet intergiciel propose de nouveaux mécanismes pour la gestion flexible de composition, destinés à faciliter la prise en charge de l'évolution constante de l'environnement et des besoins des utilisateurs.

3. la *réalisation pratique et l'expérimentation de l'intergiciel* proposé. Cette expérimentation est réalisée par la conception de plusieurs applications attentives basées sur l'intergiciel proposé. En particulier, on montre ainsi comment cet intergiciel facilite la prise en charge des principales problématiques de conception des applications attentives.

1.3 Organisation du mémoire

Le mémoire s'organise en trois parties. Tout d'abord, une étude de l'état de l'art est détaillée dans les chapitres 2, 3 et 4. Ensuite, un intergiciel pour la composition flexible d'application est proposé dans les chapitres 5, 6 et 7. Enfin, les expérimentations menées à l'aide de cet intergiciel sont détaillées dans le chapitre 8.

Partie I : État de l'art

La première partie détaille une étude de l'état de l'art dont l'objectif est double. Tout d'abord, cette étude de l'état de l'art permet de dégager les problèmes de conception des applications attentives qui sont apparus dans les travaux précédents. Ensuite, on s'intéresse aux solutions envisageables pour répondre à ces problèmes.

Chapitre 2 : Conception d'applications attentives Le chapitre 2 présente un état de l'art concernant la *conception d'applications attentives*. Tout d'abord, la notion d'application attentive est définie et illustrée à l'aide d'exemples issus de la littérature. Ensuite, ce chapitre dégage et organise les problèmes de conception d'applications attentives qui sont apparus dans les travaux précédents. Cette étude donne lieu à l'établissement d'une grille d'analyse des problèmes de conception des applications attentives. Les approches existantes sont confrontées à cette grille d'analyse. Cette confrontation permet de dégager le manque de traitement de certains problèmes, en particulier liés à l'hétérogénéité des dispositifs, à l'évolution continuelle de l'environnement et à l'équilibre entre l'automatisation et le contrôle par l'utilisateur. Elle souligne l'intérêt d'une approche de conception basée sur la composition dynamique d'applications.

Chapitre 3 : Composition dynamique et d'adaptation d'applications Le chapitre 3 étudie les approches existantes pour la *composition dynamique et d'adaptation d'applications*. Ces approches permettent la conception d'applications dont les éléments sont organisés à l'exécution, en fonction de la situation. Cette étude permet de montrer l'intérêt de solutions existantes pour traiter

certains problèmes de conception des applications attentives. Cependant, cette étude souligne aussi l'absence d'une solution apportant des solutions satisfaisant à l'ensemble des problèmes de conception des applications attentives.

Chapitre 4 : Synthèse de l'état de l'art Le chapitre 4 effectue une *synthèse des études de l'état de l'art* présentés dans les deux chapitres précédents. Cette synthèse permet d'établir une architecture de référence pour les environnements attentifs. Cette architecture de référence positionne les différents éléments et souligne la nécessité d'une infrastructure pour la gestion de composition d'application. On introduit ainsi la proposition d'une telle infrastructure sous la forme d'un intergiciel pour la composition flexible d'applications.

Partie II : Proposition d'un intergiciel pour la composition flexible d'application

La seconde partie propose un intergiciel pour la composition flexible d'application. Cet intergiciel doit fournir un support pour faciliter la conception d'applications attentives, en prenant en charge certains des problèmes dégagés dans la première partie.

Chapitre 5 : Principes d'un intergiciel pour la composition flexible d'application Le chapitre 5 présente les principes de l'intergiciel pour la composition flexible d'applications que nous proposons, nommé FCAP. Ce chapitre introduit tout d'abord le modèle conceptuel sur lequel se base FCAP. Au centre de ce modèle conceptuel se situe la notion d'application composite flexible, selon laquelle une application est composée dynamiquement par l'assemblage de fonctionnalités faiblement couplées et peut être reconfigurée en fonction de l'évolution des besoins et de l'environnement. Ce chapitre introduit ensuite les principes de fonctionnement de FCAP, basés sur l'utilisation de descriptions sémantiques de fonctionnalités et d'applications. Enfin, l'intégration de FCAP dans l'architecture de référence est précisée.

Chapitre 6 : Mécanismes de gestion de composition flexible d'applications Le chapitre 6 détaille les mécanismes de gestion de composition flexible d'applications. Ces mécanismes reposent sur la manipulation de descriptions sémantiques des fonctionnalités et de l'application. Les descriptions sémantiques définissent de manière formelle des informations sur les fonctionnalités et sur l'application. FCAP repose sur deux types de mécanismes de gestion de composition flexible. D'une part, les mécanismes d'interprétation de description permettent d'extraire des descriptions sémantiques les informations intéressantes pour la gestion de composition. D'autre part, le mécanisme de résolution permet de déterminer un assemblage de fonctionnalités adapté aux besoins de l'utilisateur et à la situation courante de l'environnement. Ce mécanisme est basé sur une modélisation sous la forme d'un problème de satisfaction de contraintes, qui permet d'intégrer les différents aspects intervenant dans la gestion de composition flexible.

Chapitre 7 : architecture multi-agents d'un système de gestion de composition Le chapitre 7 détaille l'architecture générique d'un système de

gestion de composition, qui constitue l'entité principale de FCAP. À chaque application est associé un système de gestion de composition, chargé de déterminer une composition adaptée pour l'application en fonction de la situation. Un système de gestion de composition est formé d'agents logiciels qui mettent en œuvre les mécanismes de gestion de composition dans le cadre d'un système distribué, dynamique et imprévisible. Ce chapitre détaille le fonctionnement des différents agents et du système multi-agents qu'ils constituent.

Partie III : Mise en oeuvre et experimentation

La troisième partie décrit la mise en œuvre de l'intergiciel FCAP dans le cadre de la conception de plusieurs applications attentives. Cette mise en œuvre est destinée à montrer l'intérêt du support fournit par FCAP.

Chapitre 8 : Exemples d'applications réalisées avec FCAP Le chapitre 8 décrit les expérimentations sur la base de l'intergiciel FCAP. Ces expérimentations consistent à mettre en œuvre et à faire évoluer des applications attentives qui utilisent le support fourni par FCAP. On présente ainsi deux applications, en détaillant pour chacune plusieurs variations et évolutions des besoins et de l'environnement. Ces expérimentations permettent de montrer comment l'utilisation de FCAP facilite la conception d'applications attentives, en particulier pour prendre en compte l'évolution.

Conclusion et perspectives

Le chapitre 9 présente les conclusions des travaux présentés et expose les perspectives.

Première partie

État de l'art

Chapitre 2

Conception d'applications attentives

L'objectif de ce chapitre est de décrire la problématique de conception d'applications dans des environnements informatiques particuliers, les environnements de communication ambiante.

Ce chapitre définit tout d'abord la notion fondamentale d'application attentive, et la notion d'environnement attentif qui en découle (section 2.1). Il propose ensuite une étude bibliographique en deux parties. La première partie (section 2.2) met en lumière les besoins particuliers de conception d'applications attentives tels qu'ils sont progressivement apparus au travers des travaux apparentés à l'informatique ubiquitaire. Cette première partie donne lieu à l'élaboration d'une grille d'analyse des problèmes rencontrés lors de la conception d'applications attentives. La seconde partie (section 2.3) s'intéresse aux approches de conception existantes, et les confronte à la grille établie précédemment.

2.1 Environnement attentif et applications attentives

Les environnements et les applications abordés dans ce travail ne sont pas classiques. Il importe donc de préciser dans un premier temps le domaine et les principales notions auxquels s'intéresse cette étude.

Cette section définit tout d'abord les notions d'applications attentives et d'environnements attentifs (2.1.1). Elle propose ensuite un exemple plus détaillé d'application attentive (2.1.2), avant de souligner les défis posés par la conception de telles applications (2.1.3).

2.1.1 Définitions

La première définition concerne le type d'environnement particulier considéré dans ce travail : l'environnement interactif (aussi appelé *smart space*).

Définition 1 (Environnement interactif) *Un environnement interactif est un espace physique dans lequel se trouvent des objets communicants physiques*

(capteurs, actionneurs, terminaux d'interaction), qui forment une interface entre les utilisateurs et un système d'information et de communication.

Cette notion d'environnement interactif s'inscrit dans les domaines plus vastes que sont :

- l'informatique diffuse (*pervasive computing*), qui s'intéresse à la dissémination des dispositifs électroniques et informatiques dans l'environnement physique et à la mise en relation de ces dispositifs par des réseaux de communications variés.
- l'informatique ubiquitaire (*ubiquitous computing*), qui s'intéresse aux applications rendues possibles par cette omniprésence des dispositifs informatiques dans l'environnement physique.
- l'intelligence ambiante (*ambient intelligence*), qui met l'accent sur l'utilisation de ces technologies pour créer des mécanismes d'interactions intuitifs entre les utilisateurs et les systèmes informatiques, en rapprochant ces mécanismes d'interaction des modes d'interactions naturels utilisés par les humains (attention périphérique, affordances des objets, ...).

Les environnements interactifs restreignent la problématique à l'utilisation d'un espace physique équipé de dispositifs comme système unifié d'interaction. En cela, ils se distinguent des problématiques de l'informatique mobile, qui constitue un champ distinct de l'informatique ubiquitaire, dans lequel on s'intéresse à l'interaction avec un terminal mobile, lié à la personne d'un utilisateur, et non à l'environnement physique.

Parmi les nombreux types d'applications explorés par l'informatique ubiquitaire et l'intelligence ambiante, on s'intéressera ici plus particulièrement à un certain type d'applications, les applications attentives :

Définition 2 (Application attentive) *Une application attentive est un système logiciel et matériel qui fournit un support contextualisé à des utilisateurs dans le cadre d'une activité quotidienne au cours de laquelle ils ont à interagir avec un système d'information et de communication.*

Différentes sortes d'activités peuvent être accompagnées par une application attentive. Voici quelques exemples :

Messagerie instantanée : la messagerie instantanée classique est proche d'une application attentive, car son utilisation s'entremêle souvent avec d'autres activités. Dans le cadre d'un environnement interactif, l'expérience d'utilisation de la messagerie instantanée est enrichie par l'utilisation de dispositifs d'interaction différents selon le contexte, ainsi que par la transmission automatique d'information sur le contexte des utilisateurs (Hsieh *et al.*, 2007).

Écoute de musique : l'écoute de musique est une activité qui s'effectue souvent en parallèle d'autres activités. Au sein d'un environnement interactif, une application attentive enrichit cette activité en coordonnant les différents dispositifs et fonctionnalités (lecteurs de musique, haut-parleurs) en fonction du contexte de l'utilisateur. Dans le cadre du projet GAIA, l'application détermine simplement le haut parleur le plus approprié en fonction de la localisation de l'utilisateur (Roman, 2003). D'autres exemples incluent une sélection automatique des morceaux selon l'humeur des utilisateurs (Ranganathan & Campbell, 2003).

Capture automatique d'images et de sons : dans un environnement interactif riche en dispositifs de capture d'image et de son, de nouvelles applications rendent possible de nouvelles formes d'exploitation des appareils numériques de capture d'image et de son, par exemple pour surveiller un enfant, reprendre le cours d'une conversation interrompue (Truong *et al.*, 2004) ou pour transmettre des recettes de cuisine (Terrenghi *et al.*, 2007).

Recherche asynchrone d'information sur le Web : dans le cadre d'un environnement interactif, il est possible de simplifier la recherche d'information grâce à une application attentive. Dans ce cas, le mode d'interaction n'est plus l'émission d'une requête qui donne lieu à une réponse immédiate, mais une réponse différée dans le temps. Différentes modalités d'interaction présentes dans l'environnement sont exploitées en fonction de l'importance de l'information et de la disponibilité du destinataire.

Il faut noter que la notion d'activité est l'élément central de cette définition, et qu'elle se distingue de la notion de tâche. Abowd et al. (Abowd *et al.*, 2002) soulignent quelques particularités des activités de la vie quotidienne et leur impact sur ce qu'ils dénomment *everyday computing* :

- les activités ont rarement un début et une fin bien identifiés.
- une activité peut être interrompue et reprise de nombreuses fois.
- plusieurs activités sont fréquemment effectuées en parallèle.

Comme on le verra dans la suite, ces caractéristiques ont un impact essentiel sur la manière dont les applications attentives doivent être conçues.

Enfin, on peut compléter ces définitions en considérant la notion d'environnement attentif :

Définition 3 (Environnement attentif) *Un environnement attentif est un environnement interactif dans lequel s'exécutent une ou plusieurs applications attentives.*

Cette définition fait apparaître la dimension supplémentaire de l'exploitation d'un environnement commun par plusieurs applications qui, par définition, fonctionnent en permanence et donc simultanément. Cette nouvelle dimension participe de la richesse de tels environnements, et reflète la diversité et la multiplicité des activités effectuées de manière quotidienne par les utilisateurs humains.

2.1.2 Un exemple d'application attentive

L'exemple proposé dans cette section décrit plus précisément une application permettant de rechercher des solutions de co-voiturage dans les environs de Grenoble. Un prototype de cette application a été mis en œuvre dans l'un de nos projets (Dupuis *et al.*, 2007) et permet de donner une illustration plus précise des enjeux.

Scénario Habitant à Grenoble, Thom apprécie les sports d'hiver. Pour se rendre en montagne, il privilégie le co-voiturage et utilise le système iRider : un système pair-à-pair de mise en relation grâce auquel chacun peut proposer et rechercher des solutions de co-voiturage. Ce système est complété par une application attentive qui exploite l'environnement de Thom pour le maintenir

au courant des résultats de sa recherche : pour l'informer des différentes propositions, l'application choisit dynamiquement les modalités d'interface à privilégier. Certaines propositions sont peu intéressantes, car elles ne correspondent pas exactement aux critères définis par Thom. Dans ce cas, le système n'avertit pas directement Thom, mais lui fournit l'information de manière périphérique, par exemple en modulant l'intensité d'éclairage d'une lampe ambiante. En revanche, si une proposition est particulièrement intéressante, une sonnerie est émise et un message apparaît sur un écran proche de Thom pour qu'il puisse en prendre connaissance rapidement.

Cet exemple illustre bien l'intérêt d'exploiter l'environnement et le contexte pour créer un nouveau type d'application. Dans le système de co-voiturage, les informations sont très dynamiques : au cours du temps, de nouvelles propositions apparaissent, puis disparaissent lorsque les places deviennent indisponibles. De plus, les propositions sont extrêmement diverses et potentiellement très nombreuses. En effet, la plupart des propositions ne correspondent que de manière approximative aux critères de Thom, mais peuvent malgré tout l'intéresser s'il n'a pas d'autre choix. Dans un système classique, seules deux alternatives sont possibles : fixer des critères larges et trier manuellement un grand nombre de réponses, ou fixer des critères précis et manquer des opportunités intéressantes. Dans cette application, une nouvelle dimension est rendue possible par la flexibilité de l'interaction avec le système : un grand nombre de réponses est dirigé vers Thom, mais leur pertinence se reflète dans la manière de les présenter et d'attirer son attention.

On peut noter que le scénario évoqué pour cet exemple ne doit pas masquer l'existence de nombreuses possibilités qui peuvent intervenir dans l'utilisation de cette application. En particulier, le choix d'interface d'interaction dépend de nombreux critères, tels que la disponibilité de Thom ou la présence d'autres personnes dans son environnement. Par exemple, un message offrant une proposition de co-voiturage ne doit pas être présenté sur un écran visible par des visiteurs de Thom : s'ils ne sont pas inscrits dans le système iRider, ils ne doivent pas pouvoir accéder aux informations personnelles fournies par l'émetteur de cette proposition. Par ailleurs, dans un environnement réel, les choix de dispositifs dépendent aussi des autres applications déployées dans l'environnement de Thom et de leur utilisation des ressources de cet environnement. Par exemple, l'écran mural proche de Thom peut être utilisé pour l'avertir de l'arrivée d'un invité en affichant son image. Dans ce cas, cet écran ne peut être utilisé par iRider que si la proposition est suffisamment importante et urgente pour être traitée obligatoirement par Thom avant qu'il n'accueille son invité.

2.1.3 Les défis de la conception d'applications attentives

Le concept d'application attentive présenté dans les sections précédentes pose des défis importants sur la conception de ces applications. On peut noter une différence fondamentale entre ces applications attentives et les applications classiques.

De manière générale, le comportement d'une application dépend de trois facteurs :

- le comportement individuel des différentes ressources qu'elle utilise.
- le comportement spécifié lors de la conception de l'application.

– l'environnement d'exécution, c'est-à-dire le contexte dans lequel s'effectue l'interaction.

Dans le cas des applications classiques, le comportement final est entièrement défini par le comportement spécifié à la conception. Le comportement des fonctionnalités élémentaires est bien connu au moment de la conception, et ces fonctionnalités sont choisies de manière à fournir le comportement désiré. L'environnement d'exécution quant à lui fait l'objet d'hypothèses d'utilisation, qui sont prises en compte pour spécifier complètement le contexte dans lequel s'effectue l'interaction.

Dans le cas des applications attentives, la plupart des hypothèses classiques ne sont plus possibles. D'une part, les caractéristiques des différentes ressources ne peuvent être connues parfaitement au moment de la conception. Dans le cadre d'environnements attentifs, intégrés à l'environnement physique, les ressources présentes doivent être laissées au choix des utilisateurs et ne peuvent être imposées *a priori* par les concepteurs. D'autre part, il est essentiel que le comportement de l'application soit adapté au contexte dans lequel les utilisateurs interagissent avec l'application, et ce contexte ne peut être spécifié complètement a priori. Le contexte doit être le facteur le plus influent sur l'application (Coutaz *et al.*, 2005).

2.2 Trois problématiques de conception des applications attentives

Les caractéristiques des applications attentives et l'environnement dans lequel elles évoluent diffèrent largement d'un environnement informatique de bureau classique. Une compréhension plus fine des défis rencontrés pour concevoir des applications attentives passe par un recensement et une catégorisation des contraintes génériques auxquelles est soumise une application de ce type et des propriétés caractéristiques qu'elle doit présenter en conséquence. Cependant, il s'agit ici d'un domaine émergent. Peu d'applications ont été réalisées en pratique, et peu d'études permettent de dégager ces besoins.

Cette section propose une catégorisation des problèmes génériques aux applications attentives en s'appuyant sur des travaux et études réalisés dans le domaine plus vaste de l'informatique ubiquitaire. Les travaux considérés sont ici de deux sortes :

– des comptes rendus d'expérience concernant des applications d'informatique ubiquitaire.
– des études sociologiques menées dans le cadre de l'utilisation de technologies informatiques comme support d'activité.

De ces différents travaux sont extraits des propriétés génériques qu'une application attentive devrait présenter. Cette section est conclue par la proposition d'une grille au travers de laquelle les différentes approches de conception sont analysées dans la section suivante.

2.2.1 Découplage des fonctionnalités

Le découplage de fonctionnalités diverses, hétérogènes et distribuées est une problématique qui se pose à toutes les applications en informatique ubiquitaire. Cette problématique est d'abord fortement liée à une contrainte sur le matériel.

Il n'est pas possible de concevoir de tels environnements complexes de manière intégrée, en maîtrisant tous les aspects de chacun des dispositifs. Des solutions logicielles sont nécessaires pour permettre l'intégration et l'assemblage d'applications à partir de fonctionnalités élémentaires faiblement couplées (Want *et al.*, 2002).

Dans le cas des applications attentives, la problématique de découplage est accentuée par le manque de maîtrise de l'environnement d'exécution lors de la conception. Comme le décrivent Modahl *et al.* (Modahl *et al.*, 2006) sur la base de leur expérience dans le projet AwareHome, le paradigme de déploiement des applications est le suivant :

- l'environnement est constitué de nœuds simples, modulaires qui offrent des capacités simples.
- les applications utilisent ces capacités de manière ad-hoc en fonction de leurs besoins.

Une application attentive doit donc intégrer des fonctionnalités présentes dans un environnement particulier de manière opportuniste, et ces fonctionnalités ne peuvent être modifiées pour les besoins d'une application particulière.

Cette section décompose la problématique de découplage des fonctionnalités en plusieurs problèmes. La figure 2.1 représente graphiquement la décomposition effectuée.

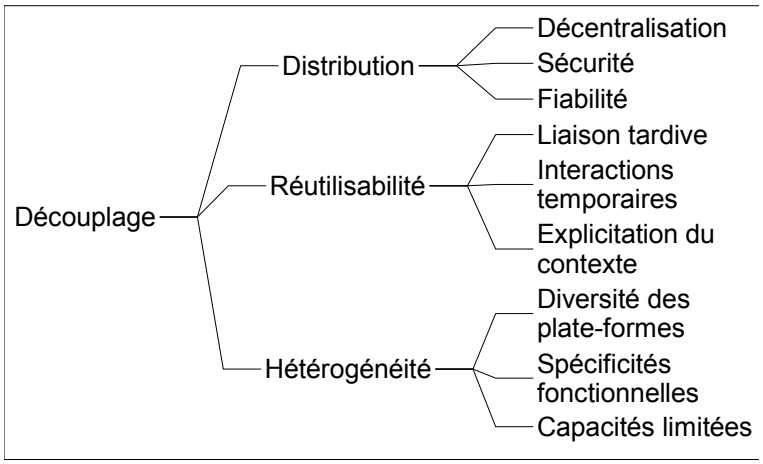

FIGURE 2.1 – Décomposition de la problématique de découplage des fonctionnalités

a) Distribution

Exemple 1 *L'application iRider utilise différentes fonctionnalités distribuées : le système iRider, qui fonctionne sur le Web, l'écran et la lampe qui sont accessibles sur le réseau local de la maison. Cette distribution a un impact sur le fonctionnement de l'application. Lorsqu'une proposition de co-voiturage est envoyée vers l'écran pour informer Thom, il faut s'assurer que l'écran a bien affiché la proposition. Par ailleurs, il faut éviter de bloquer le fonctionnement de iRider en attendant une réponse de l'écran, afin de pouvoir prendre en compte*

de nouvelles propositions.

La distribution des fonctionnalités dans l'environnement physique, et par conséquent dans un réseau, est l'une des caractéristiques essentielles de tous les environnements d'informatique ubiquitaire. Contrairement à certains systèmes d'informatique distribué où l'on cherche à masquer la distribution des fonctionnalités pour simplifier la conception d'applications, les applications d'informatique ubiquitaire doivent tenir compte de cette distribution pour s'adapter aux conditions (Saif & Greaves, 2001).

Décentralisation Dans le cas de systèmes constitués de nombreuses entités, une application doit privilégier des modes d'interaction et de contrôle décentralisés, afin d'améliorer le passage à l'échelle des systèmes et d'éviter l'existence d'éléments critiques ou de goulots d'étranglement. Les différentes entités doivent pouvoir communiquer directement, de manière pair à pair. Pour les environnements d'informatique ubiquitaire, cette problématique est accentuée par l'ancrage dans le monde physique local : deux entités en interaction sont souvent proches physiquement, et devraient pouvoir communiquer directement et localement plutôt que *via* un serveur central.

Sécurité La sécurité est l'une des principales préoccupations des utilisateurs concernant les environnements interactifs (Röcker *et al.*, 2005). Une application doit fonctionner dans un contexte sécurisé, aussi bien au niveau des communications (empêcher l'espionnage des informations, notamment sur les supports sans fil) que de l'accès aux fonctionnalités (empêcher la prise de contrôle de fonctionnalités). Ces contraintes doivent être prises en compte lors de la conception d'applications attentives, qui sont d'autant plus exposées qu'elles fonctionnent sur le long terme. De plus, la gestion de la sécurité doit être simple et la plus transparente possible pour les utilisateurs. Il n'est pas possible de leur demander une saisie de code d'identification pour chaque interaction entre les dispositifs (Bardram, 2005).

Fiabilité des communications Dans un environnement interactif, l'interaction entre l'utilisateur et le système est omniprésente. Dans ces conditions, les applications doivent être réactives aux sollicitations, et parfois assurer une réponse en temps réel. Dans un environnement distribué, cette problématique impose une gestion de la qualité de service du réseau pour garantir la meilleure performance. Cette problématique est de plus accentuée par la présence de grandes quantités de données, en particulier des informations issues de capteurs et des flux multimédia.

b) Réutilisabilité

Exemple 2 *Dans l'environnement de Thom, un écran peut être utilisé aussi bien par iRider pour présenter une proposition de co-voiturage que par le portier électronique pour afficher l'image du visiteur.*

Une application attentive est conçue indépendamment d'un environnement particulier. En effet, dans un environnement interactif, les dispositifs permettant aux utilisateurs d'interagir avec le système sont limités, à la fois par le coût

et par l'espace physique nécessaire pour les accueillir. Dans un contexte où de multiples applications peuplent un environnement, celles-ci doivent réutiliser les fonctionnalités disponibles dans cet environnement pour fonctionner, et non imposer l'ajout d'un ensemble de nouveaux dispositifs ou logiciels à l'environnement. Cette problématique de réutilisabilité impose un couplage faible entre les fonctionnalités utilisées dans une application. La problématique de réutilisabilité est précisée par les paragraphes suivants.

Liaison tardive L'environnement attentif n'étant pas entièrement maîtrisé par les concepteurs d'une application attentive, il n'est pas possible de lier une fonctionnalité fournie par une entité précise à une application, ni d'assurer la disponibilité de cette fonctionnalité. L'application doit obtenir les fonctionnalités nécessaires lors de l'exécution, et se lier aux entités qui les fournissent uniquement lorsqu'elle en a besoin (Modahl *et al.*, 2006).

Interaction temporaires Comme les entités composant une application ne sont pas connues précisément au moment de la conception, elle ne peuvent être mises en relation de manière statique et persistante (Dobson, 2003). Les communications entre les diverses fonctionnalités doivent être établies de manière *ad-hoc* et temporaire, au cours de l'exécution et peuvent être mises à jour au cours du fonctionnement de l'application.

Explicitation du contexte Pour être réutilisables, les différentes fonctionnalités doivent être conçues indépendamment d'un environnement particulier (Yau *et al.*, 2002). Elles ne doivent notamment pas faire d'hypothèse sur la configuration physique d'un environnement, ainsi que sur les objets qui le composent.

c) Hétérogénéité

Exemple 3 *iRider utilise différentes sortes de dispositifs pour tenir Thom informé de l'avancement de sa recherche de co-voiturage, par exemple un écran ou une lampe ambiante. Si tous ces dispositifs permettent de donner une information à Thom, leurs capacités et la manière de communiquer avec eux ne sont pas homogènes ni standardisées.*

L'hétérogénéité des fonctionnalités est l'une des caractéristiques spécifiques aux environnements d'informatique ubiquitaire, particulièrement dans un cadre domestique. Comme le notent Russell et al. (Russell *et al.*, 2005), les environnements d'informatique ubiquitaire sont constitués de multiples dispositifs choisis pour leur efficacité dans une tâche particulière. Cette situation engendre une multiplicité de fonctionnalités particulières, pour lesquelles il n'est pas possible de définir des spécifications homogènes facilitant leur intégration. Les applications doivent donc être conçues pour être capables de prendre en compte cette hétérogénéité des fonctionnalités.

Diversité des plateformes Dans les environnements réels, les concepteurs de dispositifs doivent pouvoir choisir les composants matériels et les plateformes d'exécution appropriées selon la fonction à remplir par le dispositif, et non selon

des besoins d'une application précise. Une application ne peut donc pas imposer l'utilisation de matériel ou de plateforme précise pour pouvoir exploiter les fonctionnalités d'un dispositif.

Spécificités fonctionnelles Contrairement à un ordinateur personnel ou à un terminal mobile, il n'existe pas de configuration standard pour un environnement interactif. Les fonctionnalités présentes dans un environnement, leur nombre et leurs caractéristiques particulières dépendent de la configuration qu'un utilisateur a choisi d'installer dans son environnement. Les différentes fonctionnalités proposées par différents fournisseurs ne sont pas homogènes : la manière de les configurer et de les utiliser dépend des choix effectués par les concepteurs de fonctionnalité, et non des besoins d'une application particulière.

Capacités limitées Les divers dispositifs présents dans l'environnement sont soumis à des contraintes spécifiques, en particulier concernant les ressources en énergie, en mémoire et en capacité de calcul. Pour chaque dispositif, les contraintes à considérer sont différentes et elles font l'objet d'une gestion particulière selon la fonction qu'il remplit (par exemple, un capteur devant fonctionner longtemps impose d'être consulté à une fréquence faible).

2.2.2 Robustesse des applications

Les environnements d'informatique ubiquitaire en général et les environnements interactifs en particulier, ont pour particularité d'être beaucoup plus dynamiques et instables que l'environnement traditionnel de l'ordinateur de bureau (Russell *et al.*, 2005). Ce sont en effet des environnements distribués, constitués de dispositif de capacités diverses, et soumis aux contraintes peu contrôlables du monde physique. Dans ces conditions, les applications attentives, destinées à rester actives sur de très longues périodes, doivent être particulièrement fiables et robustes pour fonctionner de manière satisfaisante dans un environnement dynamique, instable et en évolution.

Cette section décompose la problématique de robustesse des applications en plusieurs problèmes. La figure 2.2 représente graphiquement la décomposition effectuée.

a) Déploiement automatique

Exemple 4 *Lorsqu'il initialise une requête iRider, Thom n'a pas à indiquer les dispositifs qui pourront être utilisés pour le tenir informé (écran et lampe ambiante). Ceux-ci sont découverts et configurés pour être utilisés par iRider de manière automatique.*

Un premier aspect de robustesse d'une application attentive est la capacité à fonctionner dans un environnement particulier, l'environnement d'un utilisateur particulier, dont les caractéristiques complètes ne peuvent pas être connues précisément lors de la conception de l'application. Pour cela, l'application doit pouvoir se configurer plus ou moins automatiquement pour exploiter les ressources disponibles dans l'environnement.

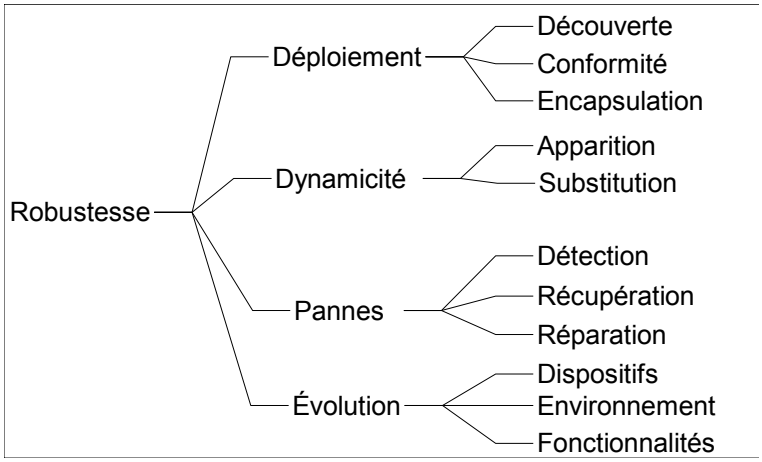

FIGURE 2.2 – Décomposition de la problématique de robustesse des applications

Découverte et sélection L'application doit découvrir et sélectionner dynamiquement les fonctionnalités présentes dans un environnement particulier. Les concepteurs d'applications doivent limiter les hypothèses sur les caractéristiques précises des ressources que l'application utilisera lors de son fonctionnement.

Vérification de conformité L'application doit assurer l'interopérabilité entre les fonctionnalités qu'elle utilise. En effet, l'environnement est un système ouvert (Davies & Gellersen, 2002) dans lequel il n'existe pas de garantie que les différentes fonctionnalités peuvent fonctionner ensemble. Aucune hypothèse ne peut donc être faite *a priori* sur l'interopérabilité des fonctionnalités, et cette interopérabilité doit être vérifiée lors de l'exécution.

Interposition et encapsulation L'application doit assembler des fonctionnalités qui n'ont pas été conçues pour fonctionner ensemble. Dans certains cas, le fonctionnement de l'application peut nécessiter la mise en place de mécanismes d'interposition et d'adaptation qui permettent à des fonctionnalités d'interagir malgré leur manque d'interopérabilité (Grimm, 2004). La capacité de l'application à mettre en place de telles adaptations de manière raisonnablement automatique est primordiale pour assurer un déploiement simple dans un environnement particulier.

b) Dynamicité

Exemple 5 *Au cours de son utilisation, l'application iRider présente différentes configurations, qui dépendent de la situation. En particulier, l'interface utilisée pour informer Thom évolue selon l'importance des informations à transmettre.*

Les applications attentives sont actives sur de longues durées et peuvent rencontrer de nombreuses situations différentes dans lesquelles elles doivent continuer à répondre aux besoins de l'utilisateur. Pour cela, elles doivent être capables de se reconfigurer dynamiquement et d'ajouter/retirer certaines fonctionnalités.

Apparition/disparition de fonctionnalité Dans un environnement interactif, l'ajout ou le retrait de fonctionnalités sont courants, que ce soit en raison de la mobilité des dispositifs ou à cause de dysfonctionnements temporaires. L'application doit prendre en charge la possibilité d'ajouter et de retirer dynamiquement des fonctionnalités au cours de son fonctionnement, sans avoir à redémarrer.

Substitution de fonctionnalités Dans un environnement interactif, il est généralement possible de trouver plusieurs dispositifs fournissant des fonctionnalités équivalentes. Il doit être possible de substituer l'une à l'autre dans l'application attentive sans interrompre son fonctionnement. En particulier, les fonctionnalités d'interface avec l'utilisateur sont souvent modifiées pour s'adapter au contexte d'interaction.

c) Tolérance aux pannes

Exemple 6 *Dans l'application iRider, la lampe ambiante peut cesser de fonctionner si elle ne dispose plus d'énergie. Dans ce cas, l'application peut continuer son fonctionnement en utilisant l'écran.*

Dans les environnements d'informatique ubiquitaire, les dysfonctionnements sont la règle, et non l'exception (Kindberg & Fox, 2002). Dans ces conditions, une application attentive doit être particulièrement tolérante aux pannes pour assurer un fonctionnement sur de longues périodes. Trois problématiques apparaissent dans la tolérance aux pannes.

Détection de panne Lorsqu'une panne survient, l'application doit être capable de détecter et d'isoler sa provenance, de manière à pouvoir corriger son comportement ou avertir l'utilisateur. En particulier, il est important de détecter si une fonctionnalité n'est plus disponible, que ce soit en raison d'une panne interne ou d'une impossibilité de communiquer avec elle. Parfois, il est aussi possible de détecter une panne temporaire.

Dégradation contrôlée En cas de panne, il est souvent possible d'assurer un fonctionnement dégradé ou minimal de l'application, qui continue de répondre en partie au besoin de l'utilisateur, mais sans offrir une qualité parfaite. La dégradation contrôlée peut se traduire par un maintien d'une partie des capacités non impactées par la panne.

Réparation automatique Dans certains cas, l'application peut se réparer automatiquement en cas de panne, notamment en remplaçant des fonctionnalités défaillantes par d'autres fonctionnalités équivalentes. En cas de panne temporaire, l'application peut aussi reprendre automatiquement la configuration qu'elle avait avant la panne.

d) Évolution de l'environnement

Exemple 7 *Dans son environnement, Thom peut acquérir et ajouter de nouveaux dispositifs que l'application iRider utilisera pour le tenir informé des propositions de co-voiturage. Si Thom ajoute un écran dans une pièce qui n'en était*

pas équipée, iRider dispose alors d'une nouvelle possibilité de lui proposer des solutions de co-voiturage dans cette pièce.

Comme le remarquent (Kindberg & Fox, 2002), les systèmes d'informatique ubiquitaire se développent par morceaux, au fur et à mesure que les utilisateurs ajoutent de nouvelles fonctionnalités. Il n'est cependant pas possible de « redémarrer le monde » pour ajouter une fonctionnalité. L'environnement doit donc offrir des possibilités d'évolution.

Ajout et remplacement de dispositifs Comme les objets physiques qui équipent un environnement domestique, des fonctionnalités peuvent être ajoutées à un environnement interactif au cours du temps. Certains fonctionnalités peuvent aussi être remplacées par d'autres plus récentes ou plus performantes. L'application attentive doit chercher à utiliser les fonctionnalités présentes et adapter son fonctionnement à ce type de modification durable.

Reconfiguration physique de l'environnement L'environnement physique est rarement figé et les éléments physiques qui le composent doivent pouvoir être déplacés et réorganisés sans interrompre le fonctionnement de l'application.

Évolution interne des fonctionnalités Comme tout système informatique, les fonctionnalités d'un environnement attentif peuvent bénéficier de mises à jour logicielles destinées à corriger des erreurs ou à apporter de nouvelles possibilités. Dans ce cas, l'application attentive doit pouvoir prendre en compte l'évolution individuelle des fonctionnalités pour adapter leur utilisation. Elle peut aussi chercher à bénéficier des améliorations qu'elles fournissent, même si elles n'étaient pas prévues au moment de sa conception.

2.2.3 Adaptabilité à des besoins non prévisibles

Les applications attentives se distinguent par la nécessité de supporter des activités mal définies et changeantes. Cette problématique s'explique par le cadre très différent par rapport aux environnements de travail (Howard *et al.*, 2007). Dans ces conditions, le comportement d'une application ne peut être défini de manière générique par des concepteurs ignorants l'environnement particulier dans lequel elle fonctionnera. Une application attentive doit pouvoir être adaptée aux besoins particuliers d'un utilisateur dans un environnement spécifique.

Cette section décompose la problématique d'adaptabilité à des besoins non prévisibles. La figure 2.3 représente graphiquement la décomposition effectuée.

a) Personnalisation

Exemple 8 *Lorsqu'il utilise iRider, Thom souhaite être averti des propositions moins intéressantes à l'aide d'un dispositif d'interaction peu intrusif. D'autres utilisateurs pourraient souhaiter que toutes les propositions soient présentées sur l'écran. L'application attentive de notification des propositions doit donc être personnalisable pour chacun des utilisateurs.*

La possibilité de personnaliser le comportement d'une application est essentiel pour garantir une adaptation aux besoins de chaque utilisateur. Dans des environnements physiques partagés, où les dispositifs et les applications peuvent être

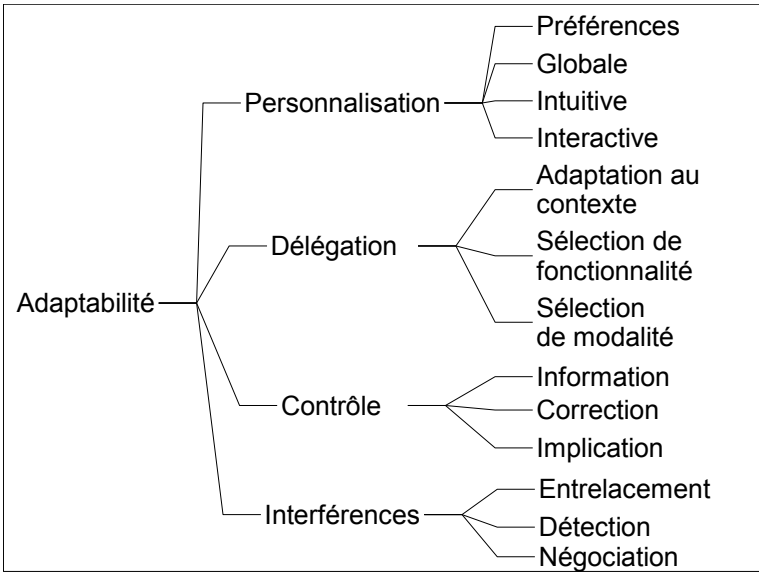

FIGURE 2.3 – Décomposition de la problématique d'adaptation à des besoins non prévisibles

utilisés par différentes personnes, les applications doivent prévoir des moyens de personnalisation appropriés. Plusieurs aspects interviennent dans la personnalisation :

Préférences spécifiques Chaque application doit fournir des moyens de personnaliser son comportement de manière spécifique à chaque utilisateur. Par exemple, (Brush & Inkpen, 2007) montrent l'importance de l'utilisation de profils personnalisés dans le cas de dispositifs partagés par plusieurs membres d'une famille. Plus généralement, dans le cadre d'une étude sur l'utilisation de la technologie dans les environnement domestiques, Howard *et al.* (Howard *et al.*, 2007) insistent ainsi sur l'existence de nombreux profils d'utilisateurs différents, dont les objectifs, la formation et les capacités sont différents et difficilement prévisibles. Cette caractéristique constitue une distinction fondamentale avec les environnements de travail, dans lesquels ces profils sont connus lors de la conception.

Personnalisation globale Dans le cadre d'un environnement attentif englobant plusieurs dispositifs et applications, les préférences des utilisateurs ne sont pas nécessairement liées au comportement d'une application particulière, mais peuvent s'appliquer globalement à plusieurs fonctionnalités et à plusieurs applications. Il faut éviter d'isoler ces préférences dans chaque application et permettre aux applications de partager ces préférences.

Personnalisation intuitive Dans le cas d'applications attentives réelles, il peut exister un grand nombre de préférences devant être définies pour chacun des

utilisateurs. La personnalisation doit donc être intuitive et ne doit pas nécessiter une tâche fastidieuse préalable à l'utilisation.

Interactivité La personnalisation doit être possible tout au long du fonctionnement de l'application, sans interrompre ce fonctionnement. En effet, les préférences des utilisateurs apparaissent souvent dans le cadre d'un contexte d'utilisation particulier : il n'est alors pas possible d'effectuer une personnalisation complète *a priori*, en tentant de prévoir tous les cas qui peuvent apparaître. L'utilisateur doit ainsi disposer de mécanismes intéractifs qui lui permettent de modifier ces préférences sans suspendre le fonctionnement de l'application.

b) Délégation

Exemple 9 *L'application iRider agit de manière pro-active pour informer Thom des propositions qui peuvent l'intéresser. L'application choisit les dispositifs les plus appropriés en fonction de la situation, sans que Thom n'ait besoin de sélectionner explicitement l'interface qu'il souhaite utiliser.*

Dans un environnement interactif, l'attention des personnes est la ressource la plus rare et la plus précieuse (Garlan *et al.*, 2002). Alors que les dispositifs et les moyens de communication entourent les utilisateurs, ils doivent éviter de les distraire et de solliciter leur attention. En particulier, certaines tâches d'administration et de contrôle peuvent être prises en charge par le système sans nécessiter l'intervention de l'utilisateur. Plusieurs types de mécanismes de délégation doivent être considérés :

Adaptation au contexte L'utilisation d'informations sur le contexte de l'interaction pour adapter automatiquement le comportement d'une application est l'une des caractéristiques essentielles de l'informatique diffuse. Barkhuus et al. (Barkhuus & Dey, 2003) montrent par exemple l'intérêt de mécanismes d'adaptation contextuels par une étude comparative entre plusieurs moyens d'adaptation.

Sélection de fonctionnalité Une manière d'adapter le fonctionnement du système est la sélection pro-active des interfaces d'interaction entre l'utilisateur et le système, notamment en fonction de leur localisation.

Sélection de modalité d'interaction En fonction de l'attention disponible, une application peut déterminer quelles sont les modalités d'interaction pour communiquer avec l'utilisateur. En particulier, les modalités d'interaction périphériques (Matthews *et al.*, 2004) permettent d'économiser l'attention de l'utilisateur.

c) Équilibre pro-activité/contrôle

Exemple 10 *Si un étranger visite Thom, les informations iRider sont automatiquement masquées pour préserver des informations sur les propositions de co-voiturage. Cependant, Thom peut décider de montrer ces informations à son visiteur : il reprend alors le contrôle sur le mécanisme automatique.*

Dans les environnements d'informatique ubiquitaire, l'utilisateur est immergé dans un environnement qui présente des caractéristiques de pro-activité permettant de soulager son attention. Cependant, les décisions d'un système automatisé sont rarement parfaites, en particulier dans un environnement complexe : un système automatisé qui prend de mauvaises décisions est alors plus complexe qu'un système manuel (Derrett, 2006). Pour cette raison, l'utilisateur doit disposer du contrôle de l'environnement, tout en laissant le système prendre en charge les tâches répétitives : la frontière qui sépare ce qui doit être fait par l'utilisateur et ce qui doit être fait par le système (le *semantic Rubicon* (Kindberg & Fox, 2002)), doit être flexible et simple à définir.

Information Afin de garantir un contrôle minimal par l'utilisateur, une application doit être capable de fournir une information simple et accessible sur les décisions prises de manière automatique.

Correction du comportement Les choix de l'utilisateur doivent être prioritaires sur les décisions effectuées par le système. Si ces dernières sont erronées, il doit être possible de les corriger et de les empêcher simplement, sans pour autant désactiver tout le système.

Implication Dans certaines situations, il est intéressant d'impliquer l'utilisateur dans le mécanisme automatique, même si celui-ci est initié par le système et non par l'utilisateur. Cette implication peut notamment passer par des actions de contrôle ponctuelles au cours du fonctionnement de l'application. Il est aussi possible de proposer des actions à réaliser par les utilisateurs, en les signalant dans l'environnement. (Intille, 2002) donne l'exemple d'un mécanisme de climatisation qui indique à l'utilisateur d'ouvrir une fenêtre en allumant un voyant lumineux sur celle-ci, plutôt que d'automatiser l'ouverture de la fenêtre.

d) Gestion des interférences

Exemple 11 *iRider utilise l'écran pour présenter des propositions de co-voiturage à Thom. Lorsque quelqu'un se présente à la porte de Thom, le portier automatique pourrait utiliser le même écran pour présenter son image.*

Comme le soulignent Davies *et al.* (Davies & Gellersen, 2002), les environnements d'informatique ubiquitaire sont le siège de nombreuses interférences et conflits entre les différentes applications et dispositifs qui s'y trouvent. En raison de la multiplicité des fonctionnalités présentes et des activités menées en parallèle, les applications doivent gérer ces interférences pour garantir un fonctionnement répondant aux différents besoins de l'utilisateur.

Entrelacement des activités Dans un environnement attentif, l'attention que l'utilisateur consacre aux différentes applications varie au cours du temps. Les activités de la vie quotidienne font souvent l'objet d'arrêts et de reprises intempestifs (Abowd *et al.*, 2002) : chaque application doit gérer l'intermittence des interactions, et ne pas monopoliser des ressources lorsqu'elle est suspendue.

Détection des conflits Lorsqu'un conflit existe entre différentes applications conçues indépendamment, des mécanismes génériques sont nécessaire pour détecter ces conflits. En particulier, il doit être possible de détecter l'impossibilité d'utiliser une ressource si elle est déjà impliquée dans une autre application.

Négociation Dans certains cas, il doit être possible de résoudre automatiquement les conflits qui apparaissent. Cela nécessite de disposer de mécanismes de négociation appropriés entre les applications concurrentes.

2.2.4 Grille d'analyse pour la conception d'applications attentives

Cette section a permis d'identifier et de catégoriser les différentes problématiques auxquelles les concepteurs d'une application attentive doivent faire face. Il faut cependant noter que la complexité majeure des applications attentives tient à la *conjonction* de tous ces aspects. En effet, pour ces environnements complexes, il n'est pas possible de séparer nettement les problèmes : chaque dimension influe sur les autres et rend plus complexe leur résolution. Pour étudier une solution potentielle à l'un des problèmes, il convient donc de bien dégager les hypothèses sous-jacentes et de vérifier qu'elles sont compatibles avec les autres problèmes à traiter par ailleurs.

L'ensemble des problématiques soulevées dans cette étude permet d'établir la grille d'analyse suivante. La table 2.2 de la section 2.3 utilise cette grille.

	Découplage			Robustesse				Adaptabilité			
Approche	Dst	Reu	Htr	Dpl	Dyn	Pan	Evo	Per	Del	Ctl	Itf

TABLE 2.1 – Grille d'analyse des problématiques de conception des applications attentives. Les abréviations utilisées pour désigner les différents problèmes sont :

Découplage des fonctionnalités (section 2.2.1)
- Dst Distribution
- Reu Réutilisabilité
- Htr Hétérogénéité

Robustesse des applications (section 2.2.2)
- Dpl Déploiement automatique
- Dyn Dynamicité
- Pan Tolérance aux pannes
- Evo Évolution de l'environnement

Adaptabilité à des besoins non prévisibles (section 2.2.3)
- Per Personnalisation
- Del Délégation
- Ctl Équilibre pro-activité/contrôle
- Itf Gestion des interférences

Cette grille permet d'évaluer les approches potentielles de conception d'applications attentives. Chaque case contient l'un des symboles suivants :

✔ : cette approche permet de traiter ce problème de manière générique.

≈ : cette approche permet de traiter certains aspects de ce problème de manière générique.

! : cette approche laisse chaque application traiter ce problème de manière spécifique.

✗ : cette approche empêche le traitement adéquat de ce problème.

On peut noter que certains problèmes ne peuvent être traités que de manière générique. C'est en particulier la cas des problèmes liés à l'hétérogénéité des fonctionnalités, à l'évolution de l'environnement et aux interférences entre applications.

2.3 Approches de conception d'applications en informatique ubiquitaire

Comme on l'a vu dans la section précédente, une application attentive doit être conçue pour répondre à un grand nombre de problématiques. Afin de simplifier la conception de nouvelles applications, il est utile d'étudier des approches de conception génériques pour traiter ces problématiques.

Cette section s'intéresse aux différentes approches de conception d'applications dans les environnements d'informatique ubiquitaire, afin de déterminer dans quelle mesure elles permettent de satisfaire les besoins des applications attentives. Ces différentes approches sont regroupées selon trois grandes catégories : les approches de prototypage d'application (section 2.3.1), les environnements interactifs programmables 2.3.2 et les environnements intelligents et adaptatifs 2.3.3. La table 2.2 détaille les résultats de cette analyse pour les différents travaux étudiés.

2.3.1 Prototypage d'applications

Dans le domaine de l'informatique ubiquitaire, de nombreux travaux étudient les applications novatrices qui peuvent être construites sur la base des technologies qui autorisent un accès pervasif aux systèmes d'information et de communication. Pour étudier des applications, l'approche privilégiée est souvent le prototypage, c'est-à-dire la construction d'un système plus ou moins fonctionnel reproduisant les caractéristiques de l'application envisagée, que l'on va pouvoir ensuite tester face à diverses conditions de fonctionnement, et surtout face à divers utilisateurs.

Cette section présente quelques approches employées pour faciliter le prototypage d'applications et étudie comment ces approches peuvent s'appliquer au cas des applications attentives.

a) Prototypage d'applications isolées pour une conception centrée-utilisateur

La conception centrée-utilisateur est l'approche privilégiée pour la conception d'applications en informatique ubiquitaire. Cette approche est tout particulièrement défendue par Norman (Norman, 1998). Celui-ci explique la complexité croissante des systèmes informatiques et leur absence d'intuitivité par la prédominance d'une orientation technologique dans l'évolution des systèmes informatiques. À l'opposé de l'ordinateur personnel intégré, il préconise des produits focalisés sur une tâche précise. Pour remplir leur mission, ces produits doivent être conçus en impliquant des utilisateurs dès les premiers stades de conception.

Approche	Découplage			Robustesse				Adaptabilité			
	Dst	Reu	Htr	Dpl	Dyn	Pan	Evo	Per	Del	Ctl	Itf
Prototypage	!	✗	✗	!	!	!	✗	!	!	!	✗
Environnement	✔	✔	✗	!	!	!	✗	!	!	!	✗
Support log	≈	≈	≈	✔	✔	✗	✗	!	!	!	✗
Compo Manuelle	✔	✔	✗	✗	✗	✗	✗	✔	✗	✔	✗
Concept Intuit	✔	✔	✗	✗	!	≈	≈	✔	✔	!	✗
Programmation	≈	✔	≈	✔	✔	✔	≈	≈	✔	!	✗
Assistant	✔	✔	✗	✔	✔	≈	✗	✗	✔	≈	✗
Buts	✔	✔	≈	✔	≈	✗	≈	✗	✔	≈	≈
Auto-org	✔	✔	≈	✔	✔	≈	✔	✗	✔	✗	✔

TABLE 2.2 – Présentation de l'analyse des approches potentielles pour la conception d'applications attentives. Les approches considérées sont les suivantes :

Prototypage	Prototypage d'applications isolées (section 2.3.1, a)
Environnement	Environnements intégrés (section 2.3.1, b)
Support Log.	Support logiciel pour le prototypage (section 2.3.1, c)
Compo Manuelle	Composition manuelle de fonctionnalités (section 2.3.2, a)
Concept. Intuit.	Conception intuitive d'applications (section 2.3.2, b)
Programmation	Langages de programmation de l'environnement (section 2.3.2, c)
Assistant	Assistant intelligent (section 2.3.3, a)
Buts	Applications dirigées par des buts (section 2.3.3, b)
Auto-org	Auto-organisation (section 2.3.3, c)

Dans le cadre de l'informatique ubiquitaire, la démarche de conception centrée-utilisateur est particulièrement nécessaire. L'informatique ubiquitaire repose sur une dissémination des systèmes informatiques dans le monde physique. La plupart des travaux se concentre sur des applications isolées dont la conception est soigneusement étudiée pour répondre aux besoins des utilisateurs. Taylor et al. (Taylor *et al.*, 2007) décrivent par exemple plusieurs applications domestiques développées en utilisant ce type d'approche.

Dans ce cadre, de nombreuses méthodologies de prototypage rapide ont été développées, afin de faciliter l'évaluation d'application en l'absence de prototypes complètement fonctionnels, coûteux et longs à mettre en place. Ces méthodologies exploitent par exemple la technique de magicien d'Oz, où le comportement des applications est simulé par des humains (Dow *et al.*, 2005) ou sur des prototypes partiellement fonctionnels (Abowd *et al.*, 2005).

D'autres travaux insistent sur la nécessité du prototypage en contexte (Reilly *et al.*, 2005), car un prototypage rapide en laboratoire ne révèle souvent pas les problèmes complexes posés par le déploiement dans le monde réel. (Rogers *et al.*, 2007) détaillent ainsi un exemple d'application et soulignent ainsi la difficulté à anticiper les contraintes du monde réel. Il faut noter que les différentes méthodes de prototypage rapide et de prototypage en contexte peuvent être combinées pour mener à un développement en phase, au cours duquel une application est évaluée incrémentalement (Fitton *et al.*, 2005).

Analyse Les caractéristiques de cette approche sont les suivantes (table 2.2)
Avantages
- Établissement d'une bonne connaissance du besoins par les concepteurs.
- Exploitation de modalités d'interaction spécifiques à la tâche (affordances, attention périphérique ..).

Limitations
- Peu de mécanismes de personnalisation des besoins, qui sont censés être explicités lors de la phase de conception.
- Pas d'adaptation à des situations imprévues, notamment en présence d'autres applications dans l'environnement.
- Pas de perspective d'évolution de l'environnement, car l'application est liée à un ensemble de fonctionnalités fixes.
- Manque de réutilisabilité des fonctionnalités développées, généralement conçues en fonction des besoins de l'application étudiée.

Dans le cas des applications attentives, les méthodes utilisées pour la conception isolée d'applications ne sont pas suffisantes. De nombreuses problématiques essentielles ne peuvent en effet pas être traitées de manière spécifique par chaque application. Elles doivent être prises en compte de manière générique au niveau de l'infrastructure même d'un environnement attentif.

b) Environnements intégrés pour le prototypage d'applications

Lorsque l'on s'intéresse à des applications reposant sur de multiples dispositifs et destinées à être déployées dans des environnements complexes, il n'est pas suffisant de concevoir un prototype simple de l'application. En effet, il est souvent difficile pour les utilisateurs potentiels de se rendre compte des conditions réelles d'interaction avec une application de manière prolongée. Il est donc néces-

saire de réaliser ces prototypes dans le cadre d'un environnement plus complet, qui permet d'immerger les utilisateurs dans l'environnement.

Le prototypage d'un environnement complet est complexe et coûteux. Pour faciliter cette démarche, il importe de favoriser la réutilisabilité des dispositifs matériels et logiciels. Plusieurs projets ont ainsi construit des environnements destinés à être utilisés pour le prototypage d'applications diverses. On peut relever plusieurs catégories d'environnements :

Les environnements de travail reproduisent les conditions d'un espace de travail équipé de dispositifs d'interaction spécifiques (écran muraux). Ils permettent en particulier d'évaluer des applications destinées au travail collaboratif (Russell *et al.*, 2005).

Les environnements domestiques reproduisent les conditions pouvant exister dans une maison interactive, c'est-à-dire équipée de nombreux objets électroniques et d'un réseau domestique qui les relie (AwareHome (Kidd *et al.*, 1999), Philips HomeLab, InHaus). Ils permettent d'évaluer des applications tournées vers les loisirs et la communication entre personnes.

Les environnements d'assistance aux personnes reproduisent les conditions particulières d'un habitat spécialement équipé dans le but de fournir une assistance directe aux personnes âgées et/ou dépendantes (Pigot *et al.*, 2003). Ils permettent d'évaluer des applications favorisant la maintenance à domicile de ces personnes, en les aidant à accomplir leurs tâches quotidiennes et en assurant un contact avec un réseau d'aidants externes.

Parmi ces différentes catégories d'environnements, les problématiques des applications attentives concernent principalement les *environnements domestiques*. Dans les environnements de travail, les applications envisagées sont orientées vers la réalisation de tâches dirigées vers un objectif bien précis, à l'inverse des applications attentives qui supportent des activités diffuses (Howard *et al.*, 2007).

Les environnements d'assistance aux personnes constituent un cas particulier d'environnements domestiques, mais présentent deux différences fondamentales :

- ils sont peu soumis aux contraintes d'hétérogénéité des dispositifs, car l'instrumentation de l'environnement peut être réalisée de manière intégrée et évolue peu ;
- les applications envisagées permettent une modélisation précise des besoins, établie en particulier dans le cadre de recherches médicales. Ainsi, il est possible de déterminer de manière relativement fiable les besoins des utilisateurs, qui concerne principalement des activités routinières pour lesquelles des méthodes de reconnaissance de plan peuvent être utilisées (Bouchard *et al.*, 2006).

Analyse Les caractéristiques de cette approche sont les suivantes (table 2.2)

Avantages

- Une réutilisabilité plus grande des fonctionnalités de l'environnement, qui peuvent être partagées entre plusieurs applications.

Limitations

- Un manque de traitement des problématiques d'hétérogénéité des fonctionnalités et d'évolution de l'environnement. Les environnements de prototypage sont conçus de manière intégrée et statique, et ne prévoient pas

de support spécifique pour intégrer de nouvelles fonctionnalités et faire évoluer l'environnement.

c) Support logiciel pour le prototypage

Face à la complexité et au coût de mise en œuvre des environnements complets évoqués dans la section précédente, certains travaux se sont intéressés à des outils plus génériques pour faciliter le prototypage d'applications. Cette approche repose notamment sur 2 aspects :

- un modèle conceptuel de constituants élémentaires génériques pour les applications ;
- des outils facilitant la construction d'applications à partir de ces constituants.

ContextToolkit (Dey *et al.*, 2001) est une solution pour le prototypage rapide d'applications sensibles au contexte. En étudiant les applications sensibles au contexte existantes, Dey dégage les problèmes communs de toutes ces applications et décrit plusieurs classes d'éléments à distinguer pour garantir une meilleure réutilisabilité. Ces classes d'éléments sont implémentées dans le ContextToolkit, une « boîte à outils » (*toolkit*) pour les applications sensibles au contexte (qui s'inspire du domaine des interfaces graphiques). Les concepteurs d'applications peuvent construire des applications en combinant les éléments (*widget*) disponibles dans le ContextToolkit, qui fournissent de manière standardisée un accès aux ressources de l'environnement (capteurs, aggrégateurs ...).

GAIA OS (Roman & Campbell, 2003) est une infrastructure destinée à faciliter la conception d'applications dans un environnement actif (*Active space*). Dans un environnement actif constitué de nombreux appareils (écran, haut-parleur) et capteurs, GAIA OS fournit un support permettant de développer des applications exploitant les fonctionnalités de plusieurs dispositifs (Roman *et al.*, 2003). Cette infrastructure repose sur la structuration des applications en plusieurs composants qui jouent des rôles spécifiques. En particulier, le coeur de l'application est clairement séparé des interfaces utilisateurs. Ainsi, une application peut utiliser différentes interfaces, et les interfaces peuvent être utilisées par différentes applications. Plusieurs applications réparties sont ainsi décrites dans (Roman, 2003), par exemple un lecteur de musique dont le comportement s'adapte au contexte en utilisant par exemple des hauts parleurs adaptés à la position de l'utilisateur.

WCOMP (Cheung-Foo-Wo *et al.*, 2006) est un framework à base de composant pour le prototypage rapide d'applications multi-dispositifs, en particulier pour l'informatique mobile et ubiquitaire. Dans cette approche, tous les constituants de l'environnement sont des composants logiciels, qui peuvent être assemblés en créant des connexions entre eux. Les outils fournit par WCOMP permettent de simplifier la définition d'applications par assemblage des composants dans le cadre de systèmes hétérogènes et dynamiques, dont le contexte varie au cours de l'utilisation. La solution proposée repose sur des mécanismes de tissage d'aspects (Tigli *et al.*, 2006) qui permettent aux concepteurs de définir différents aspects des applications de manière séparée. Un langage de spécification des interactions (nommé

ISL4WCOMP) est utilisé pour définir les interactions particulières entre certains dispositifs ainsi que les conditions contextuelles dans lesquelles ses interactions sont valides. On peut par exemple définir comment un joystick contrôle une télévision lorsqu'il est face à l'écran et comment un joystick contrôle d'une chaîne hi-fi lorsqu'il est dans la même pièce que les hauts parleurs. Ces différents aspects sont ensuite combinés à la volée lorsque plusieurs opérations s'appliquent aux conditions courantes. Cette approche facilite ainsi le prototypage en autorisant une meilleure réutilisabilité de comportements élémentaires.

Analyse Les caractéristiques de cette approche sont les suivantes (table 2.2)
Avantages
- Un traitement générique de certaines caractéristiques de distribution et de la robustesse de fonctionnement.

Limitations
- Un traitement restreint de l'hétérogénéité des fonctionnalités, qui doivent être intégrées selon le modèle prédéfini par l'approche considérée.
- Une absence de traitement générique de la problématique d'adaptation aux besoins, qui est traitée uniquement lors du prototypage d'une application particulière.

d) Synthèse

L'ensemble des approches fondées sur le prototypage partagent les caractéristiques suivantes :
- Une bonne connaissance de la tâche utilisateur ;
- La conception d'interfaces spécifiques dédiées à la tâche (exploitant en particulier les affordances, l'attention périphérique, un contexte spécifique) ;
- Un fort couplage entre fonctionnalités et la connaissance *a priori* du contexte
- Une complexité de déploiement, très peu de possibilités d'évolution ;

Ces approches offrent la meilleure expérience utilisateur pour une tâche donnée, mais ne permettent pas de s'adapter à des environnements et des besoins non prévus.

Dans le cas des applications attentives, destinées à être déployées dans des environnements très divers et à fonctionner sur de longues périodes, l'utilisation des méthodes de conception centrée-utilisateur est confrontée à des limitations intrinsèques. Pour ces applications, la définition précise des besoins est très complexe, et doit être continuellement revue lors de l'utilisation.

La conception centrée-utilisateur est bien adaptée pour des dispositifs et des fonctionnalités isolés, mais n'est pas applicables à des environnements entiers. Pour cela, il est nécessaire de définir des infrastructures facilitant l'intégration des fonctionnalités conçues indépendamment (Norman, 1998).

2.3.2 Environnements interactifs programmables

Une autre approche de conception d'application d'informatique ubiquitaire met l'accent sur la simplification de la programmation de l'environnement dans son ensemble. Dans cette approche, les applications ne sont plus conçues *a priori*

par des concepteurs d'applications, mais sont définies de manière personnelle pour chaque utilisateur.

Les environnements interactifs programmables visent ainsi à mettre à disposition les fonctionnalités distribuées et variées. Ces fonctionnalités sont destinées à être assemblées par les utilisateurs pour former des applications qui correspondent à leurs besoins. Cette section distingue trois catégories qui se rattache à la notion de programmation de l'environnement : la composition manuelle de fonctionnalités, la configuration intuitive d'applications et les langages de programmation de l'environnement.

a) Composition manuelle de fonctionnalités

Une première possibilité pour favoriser la conception directe d'applications par les utilisateurs consiste à mettre à leur disposition des outils qui permettent de découvrir et de composer les fonctionnalités présentes en fonction des besoins immédiats.

Cooltown (Kindberg *et al.*, 2000) Afin de rendre les différentes fonctionnalités d'un environnement accessible de matière plus intuitive, le projet Cooltown s'intéresse à représenter les entités du monde réel par des URLs et des pages HTML accessibles sur le Web. Cette "présence sur le Web" (*Web presence*) offerte aux personnes, aux lieux et aux choses permet d'effectuer des combinaisons ad-hoc entre ces entités, grâce au mécanisme *eSquirting*, qui s'apparente à un mécanisme de glisser-déposer des URLs des entités. Par exemple, en déposant l'URL d'une présentation sur la page Web proposée par un projecteur, celui-ci affiche la présentation. La particularité principale de ce système est l'implication de l'utilisateur dans la découverte des fonctionnalités de l'environnement, au travers des pages Web proposées par les lieux (le *PlaceManager*) qui agrègent les liens vers les pages des dispositifs de l'environnement.

ICrafter (Ponnekanti *et al.*, 2001) propose une solution pour la génération d'interface graphiques permettant de contrôler les dispositifs d'un environnement, ainsi que leur composition. Les dispositifs décrivent de manière abstraite les fonctionnalités qu'ils proposent. Ces descriptions sont utilisées par un gestionnaire d'interface (*Interface Manager*) pour construire une interface graphique adaptée permettant de contrôler le dispositif. Lorsque plusieurs dispositifs entrent en jeu, une interface composée est générée de manière à faciliter le contrôle des interactions entre les fonctionnalités. L'interface d'un service composé n'est donc pas la juxtaposition des interfaces individuelles, mais une construction adaptée qui prend en compte les interactions possibles.

Analyse Les caractéristiques de cette approche sont les suivantes (table 2.2)

Avantages

- Ces approches présentent d'abord un intérêt pour la réutilisabilité des fonctionnalités et la personnalisation des applications.

Limitations

- Il n'existe pas de mécanismes assurant la robustesse des applications. En pratique, l'imprévu est géré par l'utilisateur qui doit modifier l'application en fonction de la situation.

– Il n'existe pas de mécanisme proactif prenant en charge certains aspects de l'adaptation et libérant l'attention des utilisateurs.

Si ces approches conviennent bien à certains types d'applications d'informatique ubiquitaire (en particulier dans des environnements de travail), leur utilisation pour les applications attentives est limitée par l'obligation pour l'utilisateur d'effectuer une configuration constante de l'environnement au cours de son utilisation.

b) Conception intuitive d'applications

Afin de limiter l'implication de l'utilisateur dans le contrôle des applications au cours de leur exécution, certains travaux s'intéressent à lui permettre d'exprimer ses besoins à l'aide de méthodes de conception intuitive d'application.

iCAP (Dey *et al.*, 2006) propose un mécanisme de conception d'applications sensibles au contexte à base de règles. Dans ce système, une interface graphique permet à un utilisateur d'exprimer des règles simples à l'aide de pictogrammes qui représentent les différents concepts de l'environnement (lieux, personnes, dispositifs, évènements). Par exemple, ... L'intérêt d'une telle approche est démontré par une étude au cours de laquelle des exemples d'applications souhaitées par des utilisateurs ont été recueillies. Ces applications sont simplement réalisées

CAMP (Truong *et al.*, 2004) propose une technique de conception d'applications de capture et de restitution d'information audio-vidéo dans un environnement domestique. L'approche proposée repose sur une métaphore de « poésie magnétique », dans laquelle des mots sont assemblés pour former des phrases qui décrivent le comportement de l'application. Par exemple, il est possible d'exprimer une application pour « prendre une photo du bébé toutes les minutes au cours de la nuit ». Ces phrases sont construites par l'utilisateur à partir des concepts proposés par le système. Elles sont ensuite interprétées et traduites en une description d'application pour une infrastructure nommée INCA. Cette infrastructure se charge de contrôler les différents dispositifs de capteur (caméra, micro) et de restitution (écran, haut-parleur) qui équipent l'environnement domestique. Cette étude dégage l'intérêt d'exprimer le comportement de l'application en terme de but ou de tâche abstraite, et non en terme d'interaction entre les dispositifs de l'environnement.

extrovert-Gadget (Kameas *et al.*, 2003) propose une solution pour la conception d'applications d'informatiques ubiquitaire dans un environnement quotidien. Cette solution repose sur un style architectural (GAS : *gadgetware architectural style*) : l'environnement quotidien est considéré comme une collection de eGadgets, c'est-à-dire d'objets capables de percevoir leur environnement et de communiquer, et les utilisateurs peuvent assembler ces eGadgets pour former des applications. Pour cela, un eGadget fournit des prises (*plugs*) grâce auxquelles ils peuvent interagir avec d'autres eGadgets. Des synapses peuvent relier les prises de plusieurs eGadgets et les mettre en interaction. En créant des synapses, l'utilisateur peut créer des applications (appelées eGadgetWorld) dans lesquelles plusieurs eGadgets interagissent et communiquent pour remplir une fonction. Pour créer les applications, l'utilisateur peut utiliser une interface graphique ou la ma-

nipulation des objets pour créer des associations. Chaque eGadget peut aussi être associé à un agent intelligent, qui enrichie les applications de capacités de raisonnement. Un agent peut apprendre des synapses pertinentes pour l'utilisateur ou réparer les synapses endommagées par la perte d'un eGadget en trouvant un eGadget équivalent.

Analyse Les caractéristiques de cette approche sont les suivantes (table 2.2)
Avantages
- Ces approches sont particulièrement intéressantes pour permettre aux utilisateurs de définir leurs besoins de manière intuitive. Cette caractéristique va dans le sens d'une meilleure adaptation à des besoins changeants.
- Une certaine robustesse peut être assurée au niveau de l'infrastructure de l'environnement.

Limitations
- Les infrastructures proposées sont relativement rigides et ne prévoient pas l'évolution simple de l'environnement.

Ces techniques de programmation intuitive d'un environnement apportent des solutions particulièrement adaptées pour la personnalisation des applications aux besoins réels des utilisateurs.

c) Langages de programmation de l'environnement

Les méthodes de conception intuitive d'application fournissent une bonne adaptabilité aux besoins des utilisateurs, mais ignorent généralement les problèmes liés à la dynamicité et l'évolution de l'environnement. Ces problèmes supposent en effet des mécanismes d'adaptation plus complexes, qui vont à l'encontre de l'objectif de simplicité de programmation de l'environnement. Certains travaux s'intéressent à définir des solutions pour programmer un environnement en définissant de manière explicite les mécanismes d'adaptation.

Olympus (Ranganathan *et al.*, 2005) propose un modèle de programmation haut niveau pour les environnements interactifs. Ce framework repose sur l'intergiciel GAIA, qui permet d'accéder aux différentes fonctionnalités d'un environnement de manière unifiée. Les différents éléments qui peuvent intervenir dans une application sont définis à l'aide d'entités virtuelles : ces entités sont utilisées pour programmer l'environnement de manière abstraite, sans faire référence à des fonctionnalités précises. Au moment de l'exécution, Olympus se charge d'instancier les entités virtuelles avec des fonctionnalités existant dans l'environnement. Ce choix est basé sur une fonction d'utilité, qui prend en compte les contraintes spécifiées dans le programme ainsi que des politiques plus générales de l'environnement. Cette approche permet d'abstraire la programmation des contraintes précises de l'environnement dans lequel est exécuté le programme.

Plan B (Ballesteros *et al.*, 2006) cherche à rendre la gestion et la programmation d'un environnement interactif plus abordable en proposant une approche dans laquelle les différentes ressources de l'environnement sont représentées par des fichiers. Il est ainsi possible d'obtenir des informations en lisant certains fichiers et de contrôler des dispositifs en écrivant d'autres fichiers. L'environnement interactif repose alors sur un système

de fichier distribué (nommé Plan B), grâce auquel les différentes fonctionnalités peuvent être « montées » et accédées. Plan B propose de plus une solution pour sélectionner des fonctionnalités à l'aide de contraintes, par exemple leur localisation, sans avoir à connaître préalablement leur nom. La programmation d'une application consiste d'une part à « monter » les différentes ressources nécessaires et à programmer des scripts pour gérer les lectures et écritures dans les fichiers appropriés. Par exemple, il est possible de construire un script pour démarrer la lecture d'un fichier MP3 dans la pièce où se trouve un utilisateur. Pour cela, il faut lire le fichier indiquant la position de l'utilisateur, monter un haut-parleur avec cette contrainte de localisation, et écrire le fichier à lire sur le système du haut-parleur.

Analyse Les caractéristiques de cette approche sont les suivantes (table 2.2)
Avantages
- Ces approches permettent d'intégrer les fonctionnalités de manière satisfaisante, et prennent partiellement en compte l'hétérogénéité de ces fonctionnalités.
- Les infrastructures employées fournissent une certaine robustesse de manière générique.

Limitations
- Seuls des utilisateurs expérimentés peuvent avoir l'expertise requise pour concevoir des applications adaptées à leur besoins.
- Les questions complexes de gestion des interférences et d'équilibre autonomie/contrôle ne sont pas traitées de manière générique.

d) Synthèse

L'approche des environnements interactifs programmables fournit des solutions appropriées pour concevoir des applications adaptées aux besoins des utilisateurs, car ils définissent eux-mêmes le fonctionnement de l'application. Elles sont aussi intéressantes concernant l'intégration flexible de fonctionnalités, car celles-ci peuvent être intégrées à l'environnement de manière relativement indépendante des applications. En revanche, les fonctionnalités considérées sont souvent limitées à certaines classes bien précises, et l'hétérogénéité n'est pas suffisamment prise en compte.

2.3.3 Environnements intelligents et adaptatifs

La section précédente a présenté quelques techniques destinées à laisser un utilisateur personnaliser et programmer des applications d'informatique ubiquitaire. Cependant, dans le cas des applications attentives soumises à des environnements variés et actives sur le long terme, cette personnalisation complète pose le problème de l'attention qu'il est possible de consacrer à la gestion de l'environnement, au détriment de son utilisation.

Afin de permettre une interaction plus fluide dans un environnement interactif, certains travaux se sont intéressés à des approches plus pro-actives, destinées à assister les utilisateurs dans leur interaction. Dans ce cas, l'environnement devient intelligent et prend certaines décisions à la place de l'utilisateur. Cette

section présente ces approches en trois catégories : les assistants intelligents, les applications dirigées par des buts et les systèmes auto-organisés.

a) Assistant intelligent

Une première approche qui vise à assister l'utilisateur dans son interaction fait intervenir un gestionnaire de tâche ou un assistant intelligent, dont le rôle est de se substituer à l'utilisateur pour réaliser certaines tâches d'adaptation. Cet assistant est généralement programmé par les concepteurs de l'application afin de prendre en compte les besoins d'adaptation qui peuvent intervenir au cours du fonctionnement de l'application.

Le projet Aura (Garlan *et al.*, 2002) propose une infrastructure fonctionnant sur ce principe. Il s'agit d'un système dans lequel un utilisateur peut effectuer des tâches sans interruption tout en se déplaçant entre différents environnements. Un scénario d'application illustre la rédaction d'un document à la maison et au bureau, le système prenant automatiquement en charge le choix et le lancement des applications dans l'environnement dans lequel se trouve l'utilisateur. Dans ce cas, il est possible d'utiliser indistinctement des applications différentes (par exemple des traitements de texte tels que MS Word ou Emacs) pour offrir une fonctionnalité de traitement de texte à l'utilisateur. Ces différentes applications sont encapsulées et présentées à l'infrastructure sous la forme de fournisseurs (*supplier*) d'un même service de traitement de texte, et proposent la même API (Sousa & Garlan, 2002). De cette manière, un gestionnaire de tâches peut activer de manière transparente la fonctionnalité de traitement de texte fournie par l'une des applications. Dans le cadre de systèmes dont le fonctionnement est centré sur l'activité de l'utilisateur, il est possible de représenter une tâche de l'utilisateur sous la forme d'une coalition des différentes fonctionnalités nécessaires pour cette tâche (Garlan *et al.*, 2002). Dans le cadre du projet Aura, une telle représentation permet à un gestionnaire de tâche, nommé *Prism*, de réaliser de manière autonome une adaptation du système selon la situation de l'utilisateur. En particulier, *Prism* utilise des informations sur le contexte de l'utilisateur et sur les dispositifs présents pour choisir un dispositif sur lequel déployer une application utile pour la tâche courante, par exemple un traitement de texte. Ce choix peut par exemple être effectué en fonction de la localisation de l'utilisateur, au bureau ou chez lui, ainsi que du système d'exploitation du dispositif (Sousa & Garlan, 2002).

Analyse Les caractéristiques de cette approche sont les suivantes (table 2.2)

Avantages

- L'utilisation d'un assistant assure une bonne robustesse dans un environnement dynamique et incertain, mais des possibilités réduites d'évolution. En effet, le gestionnaire de tâche est conçu *a priori* et ne peut pas prendre en compte des situations imprévues au moment de sa conception.

Limitations

- Une intégration complexe de fonctionnalités, qui doivent être conçues de manière à pouvoir être utilisées de manière uniforme par le gestionnaire de tâche.

– Un manque d'adaptation personnalisé des applications. Le comportement est complètement défini par le gestionnaire d'application, qui est censé réaliser des adaptations appropriées sans consulter l'utilisateur. Le contrôle et la personnalisation des tels gestionnaires sont complexes.

Cette analyse révèle que l'approche proposée par les gestionnaires de tâche se heurte au caractère imprévisible de l'environnement et des besoins précis des utilisateurs. Un gestionnaire de tâche repose en effet sur un modèle d'adaptation complexe grâce auquel il peut prendre des décisions. De tels modèles peuvent être définis dans le cadre d'environnements de travail, ou encore d'assistance aux personnes, mais ne sont pas viables dans le cas des applications attentives.

b) Applications dirigées par les buts

Afin de permettre une définition plus personnalisée du comportement des applications, certains travaux se prononcent en faveur d'une approche où les applications sont dirigées par des buts de haut niveau, que le système se charge de satisfaire de la meilleure manière possible. Dans ce cas, les objectifs à remplir sont exprimés au cours du fonctionnement, et non lors de la conception de l'application.

EMBASSI (Heider & Kirste, 2002) propose un système de présentation multimodal dans le fonctionnement est explicitement dirigée par les buts de l'utilisateur. Ces buts sont tout d'abord déterminés en analysant des commandes et des interactions multi-modales (reconnaissances de paroles, de gestes). Un but est ensuite décomposé de manière hiérarchique pour déterminer un plan qui permet d'y répondre. Un plan composé de primitives disponibles dans le système. Les primitives correspondent à diverses méthodes de présentation multi-modales (synthèse vocale, images, personnage animé ...), que le plan organise de manière à répondre au but de l'utilisateur et à prendre en compte ses préférences générales ainsi que le contexte de l'environnement. Ce système est fondé sur une architecture en couche, dont la structure est dictée par le processus de traitement et de rendu des informations multi-modales (obtention des entrées, analyse, définition des buts, planification, ordonnancement des primitives, exécution). Dans chaque couche, il est possible d'ajouter de nouveaux composants de manière à augmenter les possibilités offertes par le système.

O2S (Saif *et al.*, 2003) s'intéresse aux environnements intelligents et propose une séparation en deux couches, qui distinguent les mécanismes d'adaptation et les fonctionnalités élémentaires. Cette approche permet de dissocier les besoins des utilisateurs des fonctionnalités qui seront réunies pour les satisfaire, afin de mieux adapter les applications à des environnements particuliers non prévisibles. Les mécanismes d'adaptation exploitent un système de planification : un besoin de l'utilisateur est exprimé formellement sous forme de but, décomposé dynamiquement en un ensemble de « Techniques » qui détaillent comment le réaliser (Paluska *et al.*, 2006). La solution décrite par ces techniques est construite en inter-connectant des fonctionnalités élémentaires fournies par des composants génériques, les *pebbles*. Ces composants s'échangent des flux d'informations et peuvent être inter-connectés à la demande au cours du fonctionnement du système pour former une application. Par exemple, (Paluska *et al.*, 2006) présente un lecteur de musique grâce auquel le but `PlayMusic(type=jazz)` est satisfait en connectant un composant de lecture MP3, un composant de diffusion de

musique sur haut-parleur et un composant d'interface de contrôle à distance. Dans cette architecture, les caractéristiques d'un environnement particulier sont prises en compte par les différentes Techniques assemblées pour former une solution. Ces Techniques sont de plusieurs natures et dépendent de l'environnement considéré. Certaines Techniques sont directement liées aux fonctionnalités présentes et sont une sorte de pilotes génériques pour ces fonctionnalités. D'autres Techniques sont liées à un utilisateur et définissent ses préférences pour réaliser un but.

Analyse Les caractéristiques de cette approche sont les suivantes (table 2.2)
Avantages
- Un découplage satisfaisant des fonctionnalités. Les fonctionnalités individuelles sont découplées des applications et leurs caractéristiques propres peuvent être prises en compte par le système lorsqu'il cherche à les utiliser
- Une adaptation partielle aux besoins des utilisateurs. Les besoins peuvent être définis et pris en charge de manière dynamique au cours du fonctionnement.

Limitations
- En revanche, l'utilisateur ne conserve que peu de contrôle sur la manière dont le système répond aux besoins.
- Des difficultés à fournir une robustesse aux pannes et à la dynamicité de l'environnement. Ces approches sont destinées à élaborer une réponse ponctuelle à un besoin, mais ne prévoient pas un fonctionnement sur de longues durées dans un environnement dynamique.

c) Systèmes auto-organisés

Dans les deux approches précédemment citées, les mécanismes d'adaptation sont essentiellement centralisées dans une entité chargée de gérer le fonctionnement d'une application. Cependant, afin de faire face à la dynamicité et au caractère imprévisible des environnements, d'autres travaux se sont intéressés à des approches plus décentralisées, dans lesquelles les applications sont formées par une auto-organisation des fonctionnalités.

PCOM (Becker *et al.*, 2004) est un système à base de composants permettant de réaliser des applications composées à partir de fonctionnalités offertes par divers dispositifs. Ce système ne repose pas sur une infrastructure préalablement installée dans l'environnement, mais privilégie un fonctionnement totalement décentralisé. Chaque composant définit explicitement les fonctionnalités dont dépend son fonctionnement. Lors de son activation, le composant recherche dans son environnement d'autres composants qui fournissent les fonctionnalités dont il a besoin. Une application se construit ainsi de manière spontanée, en fonction des composants présents dans le système. L'utilisation d'un système à composant permet plus de flexibilité et la reconfiguration dynamique des applications.

Soda-Pop (Hellenschmidt, 2005) est une infrastructure fondée sur l'auto-organisation de dispositifs, en particulier de dispositifs de présentation multi-modales. Soda-Pop est fondée sur une architecture dans laquelle les différents composants interagissent en envoyant des tâches à effectuer par un autre composant au travers de canaux (*channels*). Les canaux redirigent dynamiquement les tâches vers les composants les plus appropriés, en utilisant

une fonction d'utilité qui évalue la capacité d'un composant à effectuer la tâche. Ces mécanismes de redirection dynamique des tâches créent une organisation spontanée des dispositifs, qui sont amenés à interagir en fonction de la situation.

Cette approche a en particulier l'intérêt d'éviter de définir un comportement global permettant de s'adapter à toutes les situations pouvant se présenter, au profit d'un système qui s'adapte automatiquement à la situation courante. Cette infrastructure est notamment utilisée dans un système dédié aux interactions multimodales utilisant divers modalités d'interfaces disponibles dans l'environnement (Elting & Hellenschmidt, 2004).

PEACH (Busetta *et al.*, 2003) s'intéresse à un environnement d'intelligence ambiante dans un musée qui offre aux visiteurs des informations dépendantes de leur contexte. Cet environnement multi-dispositifs et multi-utilisateurs est basé sur un système multi-agents, dans lequel les différentes fonctionnalités sont fournies par des agents logiciels capables d'agir de manière autonome et de communiquer en utilisant des langages de haut niveau. Les agents peuvent non seulement communiquer directement en s'adressant des messages, mais aussi en utilisant une technique de communication particulière dans laquelle des messages sont envoyés à des groupes d'agents qui écoutent un certain canal (*channeled multicast*). Des mécanismes d'« écoute indiscrète » (*overhearing*), par lesquels des agents écoutent des messages destinés à d'autres, permettent la mise en place d'interactions complexes qui n'impliquent pas seulement un demandeur et un fournisseur de fonctionnalités, mais tout un groupe d'agents susceptibles de prendre part à l'interaction.

Analyse Les caractéristiques de cette approche sont les suivantes (table 2.2)

Avantages

- Une bonne robustesse même dans un environnement imprévisible, car peu d'hypothèses sont faites sur les fonctionnalités présentes. Cette robustesse est due à l'absence de définition complète du comportement de l'application lors de la conception : le comportement dépend des conditions rencontrées à l'exécution et des fonctionnalités qu'il est possible d'exploiter dans l'environnement.

Limitations

- Ces approches supposent souvent que chacune des fonctionnalités dispose de capacités qui lui permettent de prendre des décisions locales. Cette hypothèse s'oppose à la volonté de focaliser chaque fonctionnalité sur une tâche particulière, indépendant du fonctionnement global du système.
- Un manque de contrôle du fonctionnement du système. En effet, le fonctionnement de tels systèmes est difficile à appréhender pour un utilisateur, et il est complexe de lui fournir des moyens de contrôle sur le comportement des applications.

d) Synthèse

Les approches basées sur des environnements intelligents et adaptatifs sont destinées à prendre en charge de manière automatique la plupart des aspects du fonctionnement des applications. Cela permet de soulager les utilisateurs des

tâches de configuration et de contrôle qui perturbent l'interaction principale avec le système. Ces approches fournissent quelques solutions pour l'intégration des fonctionnalités et la robustesse de fonctionnement.

En revanche, ces approches ne traitent pas correctement la problématique de l'adaptabilité aux besoins des utilisateurs. Dans le cas des applications attentives, il n'est pas possible de définir complètement les besoins en dehors d'un contexte d'interaction précis : les utilisateurs doivent disposer de moyens de contrôle pour préciser leurs besoins au cours du temps. Cet aspect est ignoré par ces approches qui supposent qu'il est possible de définir les besoins de manière abstraite pour permettre au système d'y répondre de manière automatique.

2.4 Synthèse

Ce chapitre a pour objectif de présenter la problématique de conception d'un type particulier d'applications : les applications attentives. Tout d'abord, une définition de la notion d'application attentive est proposée et illustrée à l'aide de plusieurs exemples. Cette définition fait particulièrement ressortir l'importance de la notion d'activité quotidienne, interruptive et sans objectif quantifiable, comme cadre d'utilisation des applications attentives.

Les problématiques qui apparaissent dans la conception d'applications attentives sont ensuite identifiées, à l'aide d'exemples et de travaux précédents. Il apparaît alors que les concepteurs d'applications attentives doivent faire face à de nombreuses questions, dont certaines ne peuvent être abordées qu'au niveau de l'infrastructure globale de l'environnement attentif. Il apparaît aussi que ces problématiques sont souvent interdépendantes et ne peuvent être adressées de manière isolée. Cette étude permet finalement d'établir une grille d'analyse destinée à évaluer la pertinence d'approches de conception potentielles.

La grille d'analyse ainsi définie est confrontée à diverses approches de conceptions d'applications en informatique ubiquitaire, domaine dans lequel s'inscrivent les applications attentives. Cette analyse est structurée selon trois grandes familles d'approches : elle révèle que chacune de ces familles apporte des réponses partielles aux problématiques des applications attentives, mais échoue à fournir un cadre satisfaisant pour couvrir l'ensemble des problématiques. L'étude des approches couramment utilisées pour concevoir des applications dans des environnements interactifs a révélé qu'elles ne sont pas complètement adaptées aux applications attentives. Dans chacune des trois problématiques fondamentales, plusieurs aspects essentiels sont insuffisamment traités :

- Concernant le découplage des fonctionnalités, la problématique d'*hétérogénéité des dispositifs* n'est pas résolue, en particulier en raison de la diversité des fonctionnalités existant dans ces environnements. De nombreux travaux se basent sur l'existence d'une infrastructure unifiée masquant les disparités et permettant une interconnexion simple des dispositifs. Cependant, cette position ne semble pas réaliste au vu du nombre d'acteurs impliqués et des particularités des différentes fonctionnalités.
- Concernant la robustesse de fonctionnement, les solutions envisagées ne prennent pas en compte la nécessité de permettre l'*évolution continuelle de l'environnement*, et notamment l'apparition de situations non prévues à la conception. Dans un environnement attentif, il doit être possible d'ajouter des fonctionnalités et de modifier l'environnement au cours du fonction-

nement.

– Concernant l'adaptabilité aux besoins, deux approches contradictoires sont prépondérantes : l'utilisateur adapte manuellement les applications à ses besoins ou bien le système s'adapte automatiquement selon des mécanismes prédéfinis. La problématique de l'*équilibre autonomie/contrôle* reste ainsi ouverte même si elle est abordée dans le cadre de certaines applications particulières. Par ailleurs, les problèmes liés à la *gestion des interférences* sont souvent ignorés, car les applications sont étudiées indépendamment, dans un environnement simplifié.

Chapitre 3

Composition et adaptation d'applications

Le chapitre 2 a introduit la notion d'application attentive et a dégagé les problématiques liées à la conception de ce type d'applications. Les différentes problématiques soulevées par la conception d'applications attentives ont été abordées dans d'autres domaines de l'informatique, qui peuvent donc fournir des approches intéressantes. Ce chapitre s'intéresse plus particulièrement aux approches qui visent à *composer dynamiquement* des applications. Dans ces différentes approches, on considère une application non comme une entité monolithique, mais comme un ensemble organisé de modules logiciels : on parle de la *composition* de l'application. Les travaux considérés proposent ainsi des mécanismes qui permettent d'adapter cette composition à un besoin et à un contexte particulier : on parle de *mécanisme de composition*, dont le résultat est l'obtention d'une composition adaptée pour une application. Une *application composée* est alors une application construite suivant ces principes. Selon la situation, une application composée peut ainsi prendre la forme de compositions diverses.

Ce chapitre s'intéresse aux trois grandes problématiques dégagées dans le chapitre 2 et détermine les caractéristiques des approches de composition dynamique d'applications pour chacune de ces problématiques :

- Pour répondre à la problématique de découplage de fonctionnalités hétérogènes, la section 3.1 présente des solutions pour intégrer dynamiquement des fonctionnalités hétérogènes.
- Pour répondre à la problématique de robustesse de fonctionnement dans un environnement évolutif, la section 3.2 présente des solutions pour adapter dynamiquement la composition d'une application.
- Pour répondre à la problématique d'adaptabilité à des besoins individuels et imprécis, la section 3.3 présente des solutions pour impliquer des utilisateurs dans l'adaptation dynamique de la composition d'une application.

Pour chaque problématique, l'apport d'une approche est synthétisé de manière à mettre en lumière ses avantages et ses limitations.

3.1 Intégration dynamique de fonctionnalités hétérogènes

Comme décrit dans la section 2.2.1, une application attentive mobilise des ressources distribuées et hétérogènes. Elle doit de plus exploiter de manière opportuniste les fonctionnalités présentes dans un environnement particulier. Lors de la conception d'une application attentive, il est nécessaire de prendre en compte ces aspects et de garantir un couplage faible entre les fonctionnalités. Cependant, l'analyse de la section 2.3 montre que la plupart des approches de conception d'application en informatique ubiquitaire reposent sur un environnement dont les fonctionnalités sont bien connues et relativement homogènes.

Les approches de composition dynamique d'application favorisent le découplage des fonctionnalités individuelles. L'intégration des fonctionnalités au sein d'une application est en effet reportée au moment de l'exécution, et non à la conception. Cependant, dans le cas où les fonctionnalités sont hétérogènes et n'ont pas été prévues pour fonctionner ensemble, l'intégration dynamique soulève la question de l'*interopérabilité* des fonctionnalités. Plus généralement, cette question rejoint la problématique des *systèmes ouverts*, dans lesquels de nouvelles fonctionnalités non prévues peuvent apparaître au cours du fonctionnement.

Cette section présente des solutions permettant d'intégrer des fonctionnalités hétérogènes au sein d'une application. La section 3.1.1 présente les principes de l'architecture orientée-service, qui sont destinés à faciliter l'intégration de fonctionnalités hétérogènes, en favorisant en particulier l'indépendance du fonctionnement de chaque fonctionnalité. La section 3.1.2 présente les principes des services Web sémantiques, qui fournissent une approche permettant d'automatiser l'intégration de fonctionnalités en exploitant des solutions de représentation des connaissances.

3.1.1 Architecture orientée-service

L'intégration de fonctionnalités hétérogènes est une problématique particulièrement présente dans le domaine de l'intégration des applications d'entreprises (EAI). L'approche orientée-service s'est ainsi développée dans ce domaine, afin de simplifier l'intégration entre des systèmes propriétaires complexes, fondés sur des architectures et des langages de programmation variés. Les architectures orientée-service permettent de répondre aux problèmes d'hétérogénéité en encourageant un couplage faible entre les fonctionnalités.

Cette section présente tout d'abord les principes généraux d'une architecture orientée-service. Elle détaille ensuite l'architecture des services Web, qui constitue la réalisation la plus avancée d'architecture orientée-service. Enfin, elle aborde d'autres solutions qui adoptent les principes de l'architecture orientée-service dans des environnements d'informatique diffuse.

a) Principes d'une architecture orientée-service

Une architecture orientée-service repose sur la notion de service : toute fonctionnalité est considérée comme un service indépendant, qui possède une interface bien définie grâce à laquelle elle peut-être invoquée. Une architecture orientée-service fait intervenir deux types d'acteurs. Le *fournisseur de service*

est une entité qui propose un service pour qu'il soit utilisé par d'autres. Le *client de service* est une entité qui utilise un service proposé.

Les caractéristiques d'un service sont les suivantes (McGovern *et al.*, 2003; Fensel *et al.*, 2007) :

Modularité et composabilité Chaque service est un module indépendant qui fournit une fonction spécifique. La modularité vise à décomposer des applications en modules réutilisables (*top-down*) et à composer des applications à partir de modules existants (*bottom-up*). Dans l'approche orientée-service, les modules ont une granularité modérée (*coarse-grained*) : chaque service fournit une fonction de relativement haut niveau, et non une fonction élémentaire. Une fonction fournie par un service est généralement réalisée par un ou plusieurs programmes potentiellement complexes. Les services sont ainsi être composés pour construire des applications plus larges. Ils peuvent participer à plusieurs applications.

Distribution et accessibilité réseau Dans une architecture orientée service, les fonctionnalités sont distribuées. L'interface d'un service est accessible sur le réseau. Un client utilise cette interface pour invoquer le service à distance. Par ailleurs, la localisation précise d'un service sur une machine particulière n'est pas pertinente. Lorsqu'il interagit avec un service, le client prend seulement en compte la fonctionnalité qu'il fournit, ainsi que ses propriétés fondamentales (latence, coût ...).

Couplage faible Il y a peu de dépendance entre un service et les clients qui l'utilisent. Les clients peuvent accéder au service uniquement à travers une interface bien définie qui cache son implémentation (le service est une boîte noire). Un client peut utiliser différents services fournissant la même fonction de manière transparente. Un service est utilisable par n'importe quel client indépendamment de sa plate-forme d'exécution et de son langage de programmation. Pour cela, l'interface d'un service doit être compréhensible et utilisable par des clients variés et inconnus.

Découverte et liaison dynamique Les clients découvrent des services existants, les sélectionnent selon leurs critères et les utilisent grâce à la description de leur interface. Ils ne possèdent pas de connaissance préalable sur le service. L'utilisation d'un service se déroule donc en 3 temps :

1. *description* des interfaces qui permettent l'accès au service sur le réseau.
2. *découverte* des descriptions de services et *sélection* des services répondant aux besoins d'un client,
3. *invocation du service*, par l'envoi de messages.

À ces caractéristiques fondamentales, McGovern et al. (McGovern *et al.*, 2003) ajoutent qu'une application conçue dans une architecture orientée-service peut se réparer en cas d'erreur au niveau d'un service en utilisant dynamiquement un autre service qui fournit la même fonction. Fensel et al. (Fensel *et al.*, 2007) insistent quand à eux sur le fait que les interactions avec un service sont fondamentalement point-à-point et généralement synchrones et limitées dans le temps.

Preist (Preist, 2004) précise plusieurs concepts importants sur la notion de service dans une architecture orientée-service. En particulier, il importe de distinguer trois notions de services :

Un service abstrait est le service proposé et décrit par un fournisseur de service. Cette description couvre généralement toute une gamme de services. Certains paramètres doivent être précisés pour fournir effectivement un service (par exemple, la date de départ pour un service de réservation de vol).

Un service négocié (*agreed service*) est le service qu'un client souhaite recevoir de la part d'un fournisseur de service. La définition de ce service négocié est généralement construite à partir de la description du service abstrait proposé par le fournisseur.

Un service concret est une réalisation effective d'un service, caractérisée par des échanges de messages entre le fournisseur et le client.

À ces trois notions vient s'ajouter la notion de *service attendu*, qui est le service initialement recherché par un client. Le service négocié est le résultat de la mise en correspondance du service attendu et du service abstrait fourni (car ces deux services ne sont pas nécessairement exactement les mêmes).

b) L'architecture des services Web

Les services Web constituent la réalisation la plus avancée d'architecture orientée-service. L'architecture des services Web permet la mise à disposition de services accessibles sur le Web d'une manière bien définie et standardisée. Les services Web permettent en particulier à des entreprises de proposer des services à d'autres entreprises ou à des clients sans dévoiler le fonctionnement interne des services.

Le W3C définit ainsi la notion de service Web : « Les services Web fournissent des moyens standards d'interoperabilité entre différents logiciels qui s'exécutent sur différents types de plate-formes » (W3C, 2004c). L'intérêt des services est qu'ils peuvent être combinés pour effectuer des opérations complexes. Des programmes qui fournissent des services simples peuvent interagir avec d'autres programmes pour fournir des services plus complexes et plus intéressants.

Les services Web bénéficient d'une architecture bien définie, en grande partie standardisée par le W3C. Cette architecture comprend notamment les éléments suivants :

Description d'un service : WSDL L'interface d'un service Web est spécifiée à l'aide d'un document WSDL (*Web Service Definition Langage* (Christensen, E. et al., 2001)). Un document WSDL définit l'ensemble des messages supportés par le service et leur contenu. Une caractéristique importante de WSDL est l'indépendance par rapport à une plate-forme ou à un langage de programmation particulier. L'implémentation du service et celle du client sont laissés au libre choix de leur concepteur.

Découverte de services : UDDI L'architecture des services Web propose un mécanisme de découverte basé sur un UDDI (*Universal Description Discovery and Integration* (Clement *et al.*, 2004)). Dans ce mécanisme, un fournisseur de service enregistre un service fourni auprès d'un serveur UDDI en fournissant la description de l'interface WSDL du service, ainsi que des informations sur les caractéristiques du service. Pour utiliser un service, un utilisateur de service doit effectuer une recherche auprès du serveur

UDDI pour déterminer le service dont il a besoin, obtenir sa description et l'utiliser pour concevoir un client capable d'invoquer le service.

Invocation d'un service : SOAP Les Web services sont accessibles sur le réseau à une adresse bien définie. Un programme peut ainsi utiliser un service Web en lui envoyant des messages à cette adresse. Les messages sont des documents formatés sous forme XML et plus particulièrement à l'aide du standard SOAP. Ce standard définit un format XML particulier pour l'invocation de méthodes. Les messages peuvent être transportés par différents protocoles de communications, tels que HTTP ou SMTP.

Les services Web sont conçus pour être faiblement couplés. Un client peut uniquement interagir avec un service en lui envoyant des requêtes et en recevant des réponses. Une caractéristique fondamentale est l'absence d'état implicitement partagé entre le service et le client. Cette caractéristique favorise le découplage et l'indépendance des services. Cependant, elle limite l'utilisation de services à des actions discrètes et finies, telles que l'obtention d'une information à un moment donné ou le traitement d'un document.

c) Architectures orientées-service pour l'informatique diffuse

La notion d'architecture orientée-service a aussi été employée dans le domaine de l'informatique diffuse, là aussi dans le but de gérer l'hétérogénéité des différentes fonctionnalités logicielles et matérielles utilisées.

UPnP (Universal Plug and Play) définit une architecture pour l'interconnexion en réseau pair-à-pair de divers types d'objets communicants, de dispositifs sans fils et d'ordinateurs personnels (UPnP Forum, 2006). L'architecture UPnP est fondée sur trois types d'entités :

un dispositif (*Device*) représente un dispositif physique réel capable d'utiliser l'architecture UPnP : il peut être découvert et contrôlé à travers le réseau. Une webcam possédant une interface réseau peut par exemple être découverte comme dispositif UPnP, et donner ainsi accès aux images qu'elle perçoit.

un service (*Service*) représente une fonctionnalité élémentaire d'un dispositif, et offre des moyens de contrôle accessibles sur le réseau. Un dispositif peut ainsi fournir plusieurs services. Une webcam peut par exemple proposer des services pour contrôler la prise d'images numériques et pour contrôler l'envoi d'un flux vidéo vers un dispositif de rendu. UPnP définit de manière standardisée certains services qui peuvent être offerts par des dispositifs présents dans un environnement domestique. Par exemple, UPnP standardise les services `MediaServer` et `MediaRenderer`, qui permettent respectivement de fournir du contenu multimédia et de rendre ces contenus à un utilisateur. Le service `MediaServer` peut par exemple être implémenté par un lecteur DVD, tandis que le service `MediaRenderer` peut être fourni par un poste de télévision

un point de contrôle (*Control Point*) est le client d'un service UPnP. Il s'agit généralement d'un dispositif spécifique destiné à contrôler d'autres dispositifs UPnP. En particulier, un point de contrôle fournit généralement une interface grâce à laquelle un utilisateur peut contrôler les dispositifs UPnP présents.

L'architecture UPnP s'articule autour des étapes de fonctionnement suivantes :

Obtention d'une adresse réseau : Il s'agit d'une étape préliminaire au fonctionnement d'un dispositif UPnP. Afin de prendre en charge de manière spontanée l'arrivée et le départ de nouveaux dispositifs, UPnP définit un mécanisme permettant à un dispositif de joindre un réseau en obtenant dynamiquement une adresse grâce à laquelle il peut communiquer. On peut noter que cet aspect n'est pas présent dans le cas des services Web et constitue une caractéristique importante d'une architecture orienté-services pour les environnements physiques.

Description d'un service : Un dispositif UPnP doit fournir une description de ses services. Pour chaque service, la description inclut une liste des commandes (ou actions) auxquelles le service répond ainsi qu'une liste des paramètres de chaque action (UPnP Forum, 2006). Il n'existe pas de langage de description d'interface, mais UPnP définit un format XML permettant de décrire les services les plus courants. Des points d'extension sont prévus afin de permettre l'ajout de fonctionnalités non prévues par le standard. Dans ce cas, le service ne pourra être utilisé que par des clients spécifiquement conçus pour prendre en charge cette extension.

Découverte de services : Lorsqu'un point de contrôle est actif, il découvre tous les dispositifs UPnP présents sur le réseau. Cette étape repose sur l'échange de messages qui permettent au point de contrôle d'obtenir les informations essentielles sur chaque dispositif, et en particulier l'adresse réseau qui lui permet de communiquer.

Invocation d'un service : Un client peut invoquer une action d'un service en envoyant un message XML à son adresse.

On peut distinguer deux aspects dans l'invocation d'un service UPnP :

- *contrôle* : les services UPnP sont destinés au contrôle d'un dispositif. La description d'un service définit les actions par lesquelles un point de contrôle peut contrôler un dispositif.
- *évènements* : un service UPnP peut définir des variables d'états qui caractérisent son état interne, et un mécanisme d'évènement permet à un point de contrôle d'être informé lors du changement de valeur d'une variable d'état. Les variables d'état peuvent notamment être affectées par l'invocation d'actions, et le mécanisme d'évènement permet une surveillance asynchrone par le point de contrôle. Là encore, il s'agit d'une différence importante avec l'architecture des services Web, dans laquelle l'interaction entre un service et son client est discrète et synchrone.

On peut noter que les services présents dans l'architecture UPnP sont principalement destinés au contrôle des dispositifs, et ne fournissent généralement pas de fonctionnalité intéressante par eux-mêmes. Ainsi, ce sont généralement les effets produits par l'invocation d'un service qui doivent être considérés. Cet aspect est particulièrement présent pour des applications dans lesquelles des flux audio et vidéo sont échangés entre dispositifs (UPnP Forum, 2002). Dans ce cas, le point de contrôle interagit avec des services de type MediaServer et MediaRenderer de manière à les configurer pour qu'ils échangent un flux audio-vidéo. La transmission de ce flux a ensuite lieu sans aucune intervention du point de contrôle, et sans autre invocation de services.

DPWS (*Device Profile for Web Services*) définit une architecture permettant à des dispositifs contraints de proposer des services accessibles sur le réseau.

L'objectif est de faciliter l'interopérabilité en se basant sur les standards existants pour les services Web et en définissant de nouveaux standards couvrant les aspects manquant (comme la découverte dynamique). Les éléments essentiels sont les suivants :

Description d'un service : WSDL Comme les services Web, un service DPWS est décrit à l'aide d'un document WSDL.

Découverte de services : WS-Discovery La découverte de services s'effectue à l'aide du protocole WS-Discovery (Beatty, J. et al., 2005). Il s'agit d'un protocole de découverte asynchrone. À son arrivée sur le réseau, un client diffuse une requête de découverte, en indiquant éventuellement une restriction à un contexte (`Scope`) et à un type de service (`Type`). Les services présents répondent à cette requête de manière asynchrone, en envoyant au client un message indiquant leur présence sur le réseau. Le client doit ainsi attendre un certain temps pour obtenir des réponses.

Invocation de service : SOAP + WS-Eventing Comme un service Web, un client invoque un service DPWS en envoyant des messages SOAP à son adresse et reçoit des messages en réponse. Pour faire face aux caractéristiques des environnements dynamiques, DPWS utilise de plus le mécanisme d'évènement WS-Eventing (L. Cabrera, et al., 2004). Grâce à ce mécanisme, un client peut s'inscrire auprès d'un service de manière à recevoir des évènements dynamiquement.

On peut noter que l'architecture DPWS permet de reproduire des mécanismes similaires à ceux proposés par UPnP, mais s'appuie de plus sur les standards issus des services Web. Par ailleurs, la sécurité d'invocation des services est une des préoccupations majeures de DPWS, et des mécanismes de communication sécurisés basés sur des standards existants sont prévus.

d) Synthèse

Intérêts L'approche orientée-service fournit des solutions pour faciliter la réutilisation et l'intégration de fonctionnalité hétérogènes. En particulier, elle permet de masquer l'hétérogénéité des plateformes et des langages de programmation sur lesquels reposent les différentes fonctionnalités. Un des atouts de l'architecture orientée-service est l'existence de standards bien définis et acceptés.

Inconvénients La limitation majeure des réalisations existantes d'architecture orientée-service est la nécessité d'intégrer manuellement les applications à partir de services. Dans le cadre des environnements attentifs, cette approche est insuffisante, en raison de la dynamicité et de l'évolution des environnements.

Par ailleurs, les réalisations existantes, en particulier les services Web, se focalisent sur des modes d'interaction simples de type requête-réponse, qui sont peu adaptés aux applications attentives. Pour les applications attentives, les fonctionnalités interagissent sur de longues périodes, et échangent parfois des flux de données. Ces aspects sont mieux traités dans le cas de UPnP, en se limitant à certains types d'interactions bien particuliers.

3.1.2 Services Web sémantiques

Les principes de l'architecture orientée-service apportent une réponse à l'hétérogénéité des plate-formes et des langages de programmation des fonctionna-

lités. Cependant, leur utilisation repose sur une intégration manuelle par des développeurs, qui choisissent les services appropriés et les intègrent aux applications. Cette démarche ne convient pas complètement aux cas d'environnements ouverts, dans lesquels il n'est pas garanti que les fonctionnalités intégrées lors de la conception soient disponibles et conservent leurs caractéristiques.

Le Web sémantique s'intéresse à prendre en charge cette hétérogénéité à l'aide de techniques de représentation et de gestion des connaissances. Grâce à ces techniques, des outils automatisés peuvent faciliter la conception d'application à partir de services hétérogènes. L'approche des services Web sémantiques exploite les possibilités du Web sémantique dans le cadre des services Web.

Cette section détaille les principes du Web sémantique et des services Web sémantiques qui permettent une intégration dynamique de services Web hétérogènes.

a) Le Web sémantique

Le Web sémantique propose une vision du Web dans laquelle les informations peuvent être interprétées par des machines, et non plus seulement par des humains (Berners-Lee *et al.*, 2001). Cette vision pose le problème de gestion de l'hétérogénéité des informations représentées sur le Web. Pour y répondre, le Web sémantique propose la représentation et l'échange de l'information de manière à faciliter son traitement automatique.

La vision du Web sémantique repose sur un certains nombres de technologies organisées en plusieurs couches. Les principaux éléments de l'architecture du Web sémantique sont les suivants :

URI : identification de ressources Les URI (*Uniform Resource Identifiers*) fournissent une solution de nommage global des ressources. En particulier, les URI sont utilisées pour définir des adresses accessibles sur le Web (aussi nommées URL, *Uniform Resource Locator*). Un URI suit la syntaxe suivante :

scheme:[//authority][/path][#fragment][?query]

Dans le cas d'une ressource effectivement accessible sur le Web, scheme désigne généralement un protocole qui permet d'accéder à la ressource (par exemple http), authority désigne un nom de domaine ou de serveur, path désigne le chemin auquel est accessible le document, fragment désigne un emplacement dans le document et query désigne un ensemble de paires paramètre-valeur. Cependant, un URI ne correspond pas nécessairement à une ressource accessible et il est possible d'utiliser un URI pour identifier n'importe quel type de ressource.

RDF : description de ressources RDF (*Resource Description Framework*) est un langage qui permet d'ajouter des méta-données aux ressources du Web. Chaque ressource est identifiée par un URI. RDF permet d'exprimer des informations sur les ressources sous la forme de triplets `(Subject Property Object)` (parfois noté P(S,O)). Dans un triplet, `Subject` et `Property` sont des URI (identifiant des ressources) et `Object` est soit un URI, soit un littéral. Un ensemble de triplets décrit des connaissances sur les ressources référencées par ces triplets, en établissant des relations entre elles. Un tel ensemble de triplet est un jeu de données RDF (*RDF dataset*). Un jeu de donnée RDF peut s'exprimer sous la forme d'un graphe orienté étiqueté, dans lequel les nœuds sont les sujets ou les objets et les propriétés sont les étiquettes.

OWL : expression d'ontologies OWL (*Web Ontology Language*) permet d'expliciter le sens des termes utilisés dans un document, de manière à autoriser une interprétation automatisée par des machines. Dans le cadre du Web sémantique, les connaissances sont définies formellement sous la forme d'ontologies, c'est-à-dire d'une « spécification formelle explicite d'une conceptualisation partagée » (Gruber, 1993). L'intérêt d'une ontologie est de définir les différents concepts utilisés au sein d'un domaine et leurs relations de manière explicite et formelle. De cette manière, une machine peut effectuer des raisonnements sur ces connaissances et fournir une interprétation similaire à une interprétation humaine. Le partage de cette formalisation est une caractéristique essentielle des ontologies, en particulier dans le cadre du Web sémantique. Le langage OWL est le standard qui permet de définir des ontologies destinées à être publiées et partagées sur le Web.

Langages de règles Les ontologies permettent d'exprimer des connaissances sur les classes, les propriétés, leurs hiérarchies et certaines contraintes. Cependant, il est parfois nécessaire d'exprimer des relations plus complexes entre les entités, telles que des règles de la forme « si A, alors B ». Bien qu'il n'existe pas encore de standard pour l'expression de règles dans le Web sémantique, plusieurs propositions existent (SWRL, WSML).

b) Langages de description sémantique de services

Les technologies du Web sémantique permettent de représenter, de manipuler et de partager des connaissances hétérogènes. De nombreux travaux se sont ainsi attachés à définir des langages de description sémantique pour les services Web, qui permettent d'exprimer de manière précise les fonctionnalités offertes. Ces langages permettent ensuite des traitements automatiques pour la sélection, la composition et l'invocation de services Web.

SAWSDL *Semantic Annotation for WSDL* (SAWSDL) (W3C, 2004a) est destiné à faciliter le traitement automatique de descriptions WSDL en précisant ces descriptions à l'aide de concepts définis dans des ontologies. Ce langage privilégie l'extension du langage de description d'interface existant (WSDL) pour faciliter son adoption. En particulier, SAWSDL permet d'associer des concepts décrits formellement dans des ontologies de domaine aux différents éléments d'une description WSDL (par exemple, les paramètres des opérations du service).

OWL-S *OWL for Services* (OWL-S) (Martin *et al.*, 2007) est une ontologie qui permet de décrire des services. L'objectif est d'exploiter la formalisation apportée par OWL pour permettre la découverte, la composition et l'invocation de services (Ankolekar *et al.*, 2002). OWL-S organise la description d'un service en trois parties : le *profile* (profil), le *process model* (modèle) et le *grounding* (ancrage). Ces trois parties correspondent à trois aspects du service.

- le *profile* décrit ce que fait le service. Il a notamment pour but de permettre la découverte et la sélection automatique d'un service par un agent logiciel. Il comprend une description textuelle sur le service (pour un humain), une description fonctionnelle, en termes d'entrées, de sorties, de pré-conditions et d'effets (IOPE : *input*, *output*, *precondition*, *effect*) et d'autres attributs.
- le *process model* décrit comment fonctionne le service. Il définit les entrées, sorties, pré-conditions et effets. Il permet de spécifier un processus atomique (*atomic process*) ou composé de plusieurs autres processus (*composite process*) à l'aide de structures de contrôle (telles que Sequence, If-then-else).
- le *grounding* décrit comment invoquer le service. Cet aspect de la description exprime comment former les messages qui permettent d'invoquer les opérations

décrites. En particulier, il peut être exprimé en lien avec la spécification WSDL d'un Web service. Le *grounding* permet une certaine indépendance par rapport au type de services considéré. Martin et al. (Martin *et al.*, 2007) rapportent ainsi plusieurs expériences d'utilisation de OWL-S en dehors du cadre des services Web, notamment dans des systèmes pair-à-pair ou en utilisant l'infrastructure UPnP (UPnP Forum, 2006).

OWL-S se caractérise aussi par la volonté d'exprimer les descriptions de services en utilisant OWL. Cependant, il apparaît que de nombreuses constructions ne peuvent être exprimées simplement en utilisant OWL. En particulier, les pré-conditions et les effets nécessitent l'expression de formules logiques, pour lesquelles un langage annexe doit être utilisé.

WSMO *Web Service Modeling Ontology* (WSMO) (Fensel *et al.*, 2007) fait partie d'un ensemble de solutions destinées à permettre la réalisation de services Web sémantique (WSMO framework). WSMO définit une ontologie destinée à décrire tous les aspects d'un système fondé sur les services Web sémantiques. WSMO définit ainsi quatre concepts fondamentaux :

- les *ontologies*, qui définissent de manière formelle les concepts employés pour les descriptions.
- les *services Web*, qui offrent des fonctionnalités accessibles par des interfaces bien définies.
- les *buts*, qui décrivent les besoins des utilisateurs de services (humain, applications ou autre services)
- les *médiateurs*, qui assurent l'interopérabilité entre les différentes entités dans le cadre d'un environnement ouvert et hétérogène.

Pour chacun de ces types d'éléments, WSMO définit un modèle de description qui permet de concevoir des systèmes fortement découplés. Chaque élément peut-être défini indépendamment des usages possibles par d'autres entités du système, dans la mesure où il est accompagné d'une description exprimée à l'aide de WSMO. Lors de l'utilisation, l'intégration des ressources est réalisée grâce aux descriptions et permet de fournir un service intégré qui satisfait un but particulier.

Dans un tel système ouvert, le rôle des médiateurs pour assurer l'interopérabilité est particulièrement important. WSMO définit plusieurs types de médiateurs opérant entre les différents types d'éléments précédemment cités. Les médiateurs ont à résoudre plusieurs types d'hétérogénéité. Ainsi, l'hétérogénéité au niveau des données est traitée par un alignement des termes utilisés pour définir des entités similaires. L'hétérogénéité au niveau des processus d'interaction est traitée par à des mécanismes de manipulation des messages échangés entre services (fusion, ordonnancement ...). L'hétérogénéité au niveau des fonctionnalités est traitée par des mécanismes de comparaison et de redéfinition des fonctionnalités fournies et attendues.

Contrairement à OWL-S, WSMO repose sur un langage de représentation des connaissances spécifiquement développé pour répondre aux besoins de description de service : WSML. WSML intègre ainsi plusieurs types de construction logique pour atteindre l'expressivité requise.

c) Éléments de description non-fonctionnelle des services

Les langages de description des services présentés dans la section précédente s'intéressent en priorité à la description fonctionnelle des services. Ils permettent d'enrichir les langages de descriptions d'interface avec des concepts bien définis. Cependant, dans de nombreux cas, le choix de services est conditionné par des informations qui ne concernent pas directement la fonctionnalité fournie, mais plutôt les caractéristiques non-fonctionnelles.

Description non-fonctionnelle de services Web Dans le cas des services Web, certains travaux se sont intéressés à décrire des caractéristiques non-fonctionnelles concernant en particulier la qualité de services (QoS). La prise en compte de ces aspects permet de discriminer des services fonctionnellement similaires. Par exemple, Claro *et al.* (Claro *et al.*, 2006) considèrent un modèle dans lequel les aspects de coût, de temps de réponse, de disponibilité et de réputation sont pris en compte. Ben Mokhtar *et al.* (Ben Mokhtar *et al.*, 2005) proposent de considérer d'autres critères, en différenciant les aspects quantitatifs et qualitatifs.

Description non fonctionnelle d'autres types de services Dans le cas d'autres architectures orientée-service, d'autres aspects peuvent apparaître. Ainsi, certaines architectures sont destinées à la gestion de services multimédia. Dans ces architectures, il existe de nombreuses contraintes non fonctionnelles afin notamment d'assurer une qualité de service suffisante pour les utilisateurs. Behre (Berhe Hagos, 2006) propose par exemple une description des caractéristiques de contenu multimédia.

Description du contexte physique Dans le domaine de l'informatique diffuse, l'utilisation de description sémantique de services est aussi envisagée (Chakraborty *et al.*, 2006) . Dans ce cas, la description du contexte physique associé à un service est essentielle (Maamar *et al.*, 2005).

d) Synthèse

Intérêts Grâce à l'introduction de technique de représentation des connaissances, l'approche des services Web sémantiques comble certaines lacunes de l'approche orientée-service. En particulier, les descriptions sémantiques permettent d'automatiser l'intégration dynamique de fonctionnalités qui n'ont pas été conçues pour fonctionner ensemble. De plus, les descriptions sémantiques offrent un cadre plus flexible, dans lequel il est possible d'introduire des modèles permettant d'expliciter des aspects non-fonctionnels et contextuels des fonctionnalités.

Inconvénients Dans le cadre des applications attentives, l'utilisation de services sémantiques semble intéressante. Cependant, il apparaît que les langages de description existants (principalement destinés aux services Web) sont insuffisants pour les besoins des applications attentives. En particulier, les descriptions d'interactions prolongées et d'informations non-fonctionnelles et contextuelles sont assez peu développées. Ces aspects sont particulièrement importants dans le cas des applications attentives.

3.2 Adaptation dynamique d'une application composée

Les applications attentives soulèvent des défis importants en terme de robustesse face à un environnement dynamique et évolutif. L'analyse des travaux existants (section 2.3) révèle que les problèmes liés à la dynamicité sont parfois abordés, mais que les problèmes liés à l'évolution de l'environnement sont généralement ignorés : les applications s'adaptent à des situations prévues lors de la conception, mais ne peuvent pas prendre en compte les nouvelles situations créées par l'évolution de l'environnement.

Les approches de composition dynamique d'application peuvent fournir une solution générique à certains des aspects de la problématique de robustesse. Pour faire face à l'évolution de l'environnement, les applications ne sont pas conçues par un assemblage statique de fonctionnalités, mais par une composition dépendant du contexte et

des fonctionnalités disponibles dans une situation donnée. Pour faire face à la dynamicité de l'environnement, cette composition doit pouvoir être adaptée dynamiquement aux changements qui peuvent survenir au cours du fonctionnement de l'application.

Cette section présente trois familles de propositions pour adapter dynamiquement la composition d'une application en fonction de la situation. La section 3.2.1 s'intéresse aux mécanismes de composition automatisée de services Web, grâce auxquels il est possible de construire dynamiquement des applications regroupant les services Web appropriés. La section 3.2.2 introduit les mécanismes de reconfiguration d'applications à base de composants, dans lesquels une application peut être adaptée à de nouvelles conditions par une modification de l'organisation des éléments qui la composent. Enfin, la section 3.2.3 présente les applications adaptatives à base d'agents, dans lesquelles une application est réalisée par un système d'agents autonomes capables de modifier leur comportement et leur organisation pour faire face à de nouvelles conditions.

3.2.1 Mécanismes de composition automatisée de services Web

Les services Web constituent une solution permettant la construction d'applications complexes à partir de fonctionnalités hétérogènes. L'un des intérêts principaux des services Web est aussi la possibilité de composer des services existants pour former de nouveaux services, adaptés à un besoin précis. Cependant, il s'agit de systèmes destinés à fonctionner dans le cadre dynamique et ouvert du Web : il n'existe en particulier aucune garantie pour un utilisateur que les services utilisés seront toujours disponibles et n'évolueront pas. Pour répondre à ces problématiques, de nombreux travaux se sont intéressés à des approches de composition automatisée de services Web, qui permettent de répondre à des besoins utilisateur à la demande, et de manière ad-hoc. Une application correspondant à un besoin d'utilisateur est ainsi construite dynamiquement, en assemblant des fonctionnalités disponibles sous forme de services Web.

Cette section présente tout d'abord les principes de composition des services Web, qui repose sur l'orchestration de *workflow* de services Web. Plusieurs approches envisagées pour automatiser la création d'une composition de services répondant à un besoin sont ensuite présentées. Enfin, quelques solutions pour gérer l'adaptation dynamique d'applications construites à base de services Web sont décrites.

a) Principes de la composition de services Web

L'architecture des services Web décrite dans la section 3.1.1 porte principalement sur la fourniture, la découverte et l'utilisation individuelle des services Web. Des services Web élémentaires peuvent par ailleurs être composés pour former des services plus larges, répondant de manière plus précise aux besoins d'un utilisateur. Par exemple, un service permettant l'organisation complète d'un voyage peut être créé à partir de services permettant de réserver des billets d'avion et des services permettant de réserver des chambres d'hôtel.

Une composition de services Web est généralement définie sous la forme d'une orchestration, qui définit un ensemble d'étapes et de transitions. Chaque étape correspond à l'invocation d'un service et chaque transition correspond à la transmission d'une sortie d'un service Web vers un autre service Web. Des structures de branchement et de conditions permettent de définir des applications complexes. Dans une orchestration de services Web, les fournisseurs de services ne communiquent pas directement entre eux. L'orchestration est exécutée par une entité centrale, qui invoque les services Web l'un après l'autre, reçoit leurs réponses, les analyse puis les transmet aux services suivants.

Afin de simplifier la conception d'applications sous la forme d'orchestration services Web, des langages d'orchestration ont été définis. À l'aide d'un tel langage, il est

possible de décrire le *workflow* à exécuter. Cette description est interprétée par un moteur d'exécution de composition, qui se charge d'invoquer les services Web et de fournir le résultat final de l'orchestration. BPEL4WS (*Business Process Execution Language for Web Service*) est la solution la plus utilisée dans le cadre des services Web, mais d'autres travaux s'appuient sur les diagrammes d'état UML ou sur des formalismes proches des réseaux de pétri.

b) Composition automatisée de services Web

Les langages décrits dans la section précédente permettent uniquement une composition statique des services : l'orchestration est conçue par un développeur à l'aide de services disponibles au moment de la conception de l'application. Afin de simplifier la composition de services, certains travaux s'intéressent à la composition automatisée de services Web à partir de la description d'un besoin.

La plupart des travaux sur la composition automatique de services adoptent une démarche similaire :

1. Adoption d'une technique connue de résolution de problème, possédant des propriétés intéressantes pour la composition de services,
2. Modélisation du problème de composition dans le cadre du formalisme correspondant à la technique choisie,
3. Description des services, généralement en faisant ressortir les notions d'entrée/sortie et de préconditions/effets (IOPE),
4. Résolution du problème à l'aide de la technique choisie. Cette résolution nécessite parfois l'ajout de nouvelles constructions ou de contraintes particulières par rapport à la technique originale,
5. Expression du résultat sous forme d'une orchestration de services Web.

De nombreuses approches ont ainsi été proposées en utilisant des techniques de résolution de problème existantes. Henocque et Kleiner (Henocque & Kleiner, 2007) distinguent par exemple 17 approches envisagées pour la composition automatisée de services Web, dont voici quelques exemples :

Système à base de règles SWORD (Ponnekanti & Fox, 2002) est un outil destiné à assister des développeurs pour effectuer une composition de services Web. Dans cette approche, un service est décrit de manière relativement simple sous la forme d'une règle associant les sorties produites aux entrées fournies. Pour déterminer une composition, un développeur doit décrire un ensemble de sorties souhaitées et un ensemble d'entrées disponibles. SWORD utilise un système à base de règles pour déterminer un enchaînement de règles qui permet d'obtenir les sorties à partir des entrées. Si un tel enchaînement est trouvé, la composition correspondante est reconstituée en analysant les règles exploitées. Une des particularités de SWORD est la prise en compte de l'existence de plusieurs possibilités de composition, en particulier lorsqu'un service peut produire plusieurs sorties pour une même entrée. SWORD prévoit ainsi un mécanisme de filtres, que le développeur peut intercaler dans la composition afin de traiter les informations produites par un service. Deux catégories de filtres sont proposées : les uns sont des filtres automatiques, qui permettent par exemple de trier les réponses fournies par le service ; les autres sont basés sur la génération d'une interface utilisateur, grâce à laquelle les ambiguïtés peuvent être levées manuellement.

Logique des situations Certains mécanismes de composition automatique s'appuient sur des systèmes de planification existants. McIlraith et al. (McIlraith & Son, 2002) s'appuie sur le langage Golog. Golog est un langage de programmation logique spécialisé pour les domaines dynamiques, qui est ainsi adapté aux services Web. Dans l'approche proposée, les services sont décrits comme des actions, représentables dans le langage Golog. La composition de services est alors

conçue comme une planification d'actions menant d'une situation de départ à un but. Dans le cas des services Web, des « actions » sur les connaissances et les perceptions sont ajoutés. Une des applications proposées est la composition de services à partir de procédures génériques de composition et de la spécification des contraintes de l'utilisateur.

Planification hiérarchique Une autre approche proposée est l'utilisation de mécanismes de planification hiérarchique (Hierarchical Task Network, ou HTN). Pour effectuer une tâche, un HTN la décompose en sous-tâches jusqu'à obtenir des tâches atomiques. La décomposition doit tenir compte des contraintes existant entre les différentes tâches. Wu et al. (Wu *et al.*, 2003) proposent ainsi l'utilisation du planificateur SHOP2 pour effectuer la composition de services exprimée dans OWL-S. Une composition de haut niveau (exprimé par un processus OWL-S) est l'entrée du système et la sortie est un chemin d'exécution formé de processus atomiques, qui peuvent être exécutés. Cette approche est justifiée par le fait que concept de décomposition des tâches des HTN est très similaire à la décomposition de processus composites dans OWL-S (Wu *et al.*, 2003).

c) Adaptation dynamique de composition de services Web

Les mécanismes précédents fonctionnent essentiellement hors-ligne : la composition est déterminée à partir de connaissances collectées *a priori* et censées ne pas évoluer. Ces mécanismes ne sont adaptés que lorsque les services à utiliser sont bien connus au départ et que leur fonctionnement est garanti. Cependant, le Web est un environnement ouvert et il n'est pas garanti que les services proposés par un fournisseur soient toujours disponibles et conservent des caractéristiques constantes.

La problématique de *l'ouverture* du domaine des services Web se pose de manière cruciale. En effet, les techniques de résolution de problème sont souvent conçues pour fonctionner dans un monde fermé, dans lequel il est non seulement possible de spécifier précisément le problème à résoudre, mais aussi de décrire *a priori* toutes les connaissances et actions disponibles. Dans le cas des services Web, il importe d'intégrer les mécanismes de planification avec les mécanismes de découverte et de sélection dynamique de service (voir section 3.1.2). Sirin (Sirin, 2006) propose en particulier des solutions pour l'intégration des mécanismes de planification HTN avec les formalismes de description sémantique de services.

Une deuxième problématique importante est la *robustesse d'exécution* de la composition. Dans le cas où une erreur survient lors de l'invocation d'un service, les techniques de composition hors-ligne nécessitent de calculer de nouveau une composition appropriée. À l'inverse, Maamar et al. (Maamar *et al.*, 2003; Maamar *et al.*, 2004) présentent une approche dans laquelle la planification et l'exécution sont entrelacées. Dans cette approche, la composition est réalisée par un système d'agents logiciels, qui se répartissent les tâches de composition et d'exécution.

Une troisième problématique essentielle est la *prise en compte dynamique d'informations*. Dans le cadre du projet LanguageGrid, Ben Hassine et al. (Ben Hassine *et al.*, 2006) décrivent une solution pour réaliser dynamiquement une composition de services en intégrant les informations manquantes au cours du processus de composition. Ces informations peuvent notamment provenir de l'utilisation ou d'autres services Web. Dans cette approche, le problème de composition est modélisé par un problème de satisfaction de contraintes.

d) Synthèse

Intérêts L'intérêt principal de la composition automatisée de services Web est la grande flexibilité de déploiement qui résulte du découplage entre la fonctionnalité glo-

bale de l'application et les fonctionnalités élémentaires qui la réalisent. Une application peut ainsi être définie indépendamment de fonctionnalités particulières et instanciée dans un environnement particulier en fonction des fonctionnalités présentes.

Dans le cas où la composition automatique est couplée avec l'utilisation de descriptions sémantiques, cette approche permet une ouverture plus grande de l'environnement et une évolution indépendante des fonctionnalités. La description explicite des fonctionnalités permet de résoudre dynamiquement les incompatibilités qui peuvent survenir.

Inconvénients Dans cette approche, la dynamicité de l'environnement est difficilement prise en compte, car ces approches s'intéressent à des applications dont la durée de vie est courte et où les fonctionnalités (services Web) sont donc généralement stables. Ainsi, les mécanismes de composition ont un fonctionnement linéaire qui débouche sur une description fixe de l'application. Cette description prend la forme d'une orchestration destinée à être exécutée pas à pas, plutôt que d'un assemblage durable de fonctionnalités.

3.2.2 Mécanismes de reconfiguration d'applications à base de composants

La programmation à base de composants fournit un cadre général pour la conception d'applications modulaires, dans lesquelles les différentes fonctionnalités sont bien séparées au sein de modules, appelés composants. Dans le cadre de cette approche, il est possible de concevoir des applications capables de s'adapter aux situations rencontrées grâce à des modifications des éléments qui la composent.

Cette section présente les mécanismes de reconfiguration d'applications à base de composants. Elle présente tout d'abord les principes généraux de la programmation à base de composants. Elle introduit ensuite les mécanismes de reconfiguration dans les applications à base de composants. Enfin elle détaille quelques travaux portant sur l'adaptation d'applications par reconfiguration en prenant en compte le contexte.

a) Principes de la programmation à base de composants

Dans un cadre où les systèmes informatiques deviennent de plus en plus complexes, la réutilisabilité est un objectif essentiel : il est préférable de réutiliser des fonctionnalités déjà implémentées et testées plutôt que de développer entièrement les nouvelles applications. L'utilisation de composants vise à encourager cette réutilisabilité : un composant peut ainsi être défini comme un module indépendant de logiciel réutilisable. La programmation à base de composant consiste à construire des applications en assemblant et en faisant interagir des composants.

Pour être facilement réutilisable, un composant doit présenter un certain nombre de propriétés, dont les principales sont (Szyperski, 1998) :

- l'encapsulation : l'utilisation de composants permet notamment de masquer les détails de fonctionnement d'une fonctionnalité. Le composant ne laisse apparaître qu'une interface extérieure grâce à laquelle on peut l'intégrer dans une application. Cela permet de manipuler les fonctionnalités à un niveau d'abstraction assez élevé, ce qui simplifie la construction de systèmes complexes.
- l'indépendance à un contexte d'utilisation.
- la composabilité avec d'autres composants.

Modèles de composants La conception d'application à base de composants suppose que les différents composants employés soient effectivement capables d'interagir et d'être intégrés dans une application. Pour cela, un composant se fonde sur un modèle

de composant, qui définit les propriétés qu'il doit posséder. Divers modèles de composants existent (par exemple, CCM (OMG, 2006), Fractal (Bruneton *et al.*, 2002)). Les modèles de composants se distinguent notamment par les modes d'interaction et de contrôle que doivent fournir les composants. Ainsi, un modèle de composant est souvent lié à un domaine d'application et à un environnement de déploiement particulier.

Architecture d'une application à base de composants L'étude des architectures logicielles (Shaw & Garlan, 1996) permet de considérer une application à un niveau d'abstraction élevé, où seule la structure de l'application, ses fonctionnalités et les interconnexions apparaissent. Les détails des algorithmes et des formats de données sont masqués. Il existe plusieurs définitions de la notion d'architecture logicielle, mais la plupart font intervenir la notion de composants et de connecteurs (Kaisler, 2005) :

Les composants sont les unités de traitement élémentaires, qui réalisent les différentes fonctionnalités d'une application (aussi bien des calculs qu'une interface utilisateur ou un stockage de données).

Les interfaces permettent à un composant de communiquer avec l'extérieur, et notamment avec d'autres composants.

Les connecteurs définissent les interactions entre des composants.

Une configuration est un ensemble de composants et de connecteurs qui les relient pour former la structure d'une application.

Un langage de description d'architecture (ADL, *Architectural Description Language*) permet ainsi de spécifier une application à un haut niveau d'abstraction en décrivant son architecture : les composants dont elle est formée et les connecteurs qui relient entre ces composants. Il existe plusieurs types de langage de définition d'architecture, depuis les langages graphiques à base de boîtes et de traits jusqu'aux langages formels qui permettent une spécification stricte de l'architecture (Allen, 1997).

b) Reconfiguration des applications à base de composants

Une des possibilités offertes par la programmation à base de composants est la reconfiguration de l'application au cours de l'exécution. En effet, la séparation des préoccupations et le faible couplage entre les composants facilite le remplacement d'un composant en limitant l'impact sur les autres. Il devient ainsi possible de faire évoluer l'architecture d'une application au cours de son exécution.

Oreizy et al. (Oreizy *et al.*, 1999) distinguent deux approches concernant la reconfiguration d'application à base de composants. Les deux approches possèdent des caractéristiques communes qui facilitent la reconfiguration dynamique : la distinction entre composants et connecteurs, l'utilisation d'interactions asynchrones et l'absence d'hypothèse sur la complexité des composants. La première approche privilégie la reconfiguration des connecteurs. Les composants reçoivent des objets en entrée et produisent des objets en sortie, mais obéissent au principe de communication aveugle (*blind communication*) : ils sont ignorants de la provenance et de la destination de ces objets, ils ne connaissent pas les caractéristiques des connecteurs qui les acheminent et ils ne sont pas informés des pertes de communication. Ainsi, il est possible de modifier les connecteurs de manière transparente. La seconde approche privilégie l'ajout et la suppression de composants. Les composants sont organisés hiérarchiquement et connectés par des connecteurs qui re-dirigent les messages provenant des composants supérieurs vers les composants inférieurs. Un composant connaît les composants supérieurs dont il reçoit des messages, mais ne connaît pas les composants inférieurs.

Bruneton et al. (Bruneton *et al.*, 2002) détaillent les mécanismes de reconfiguration offerts par le framework Fractal. Ces travaux proposent une approche à base de composants hiérarchiques : de nouveaux composants sont créés à partir des composants

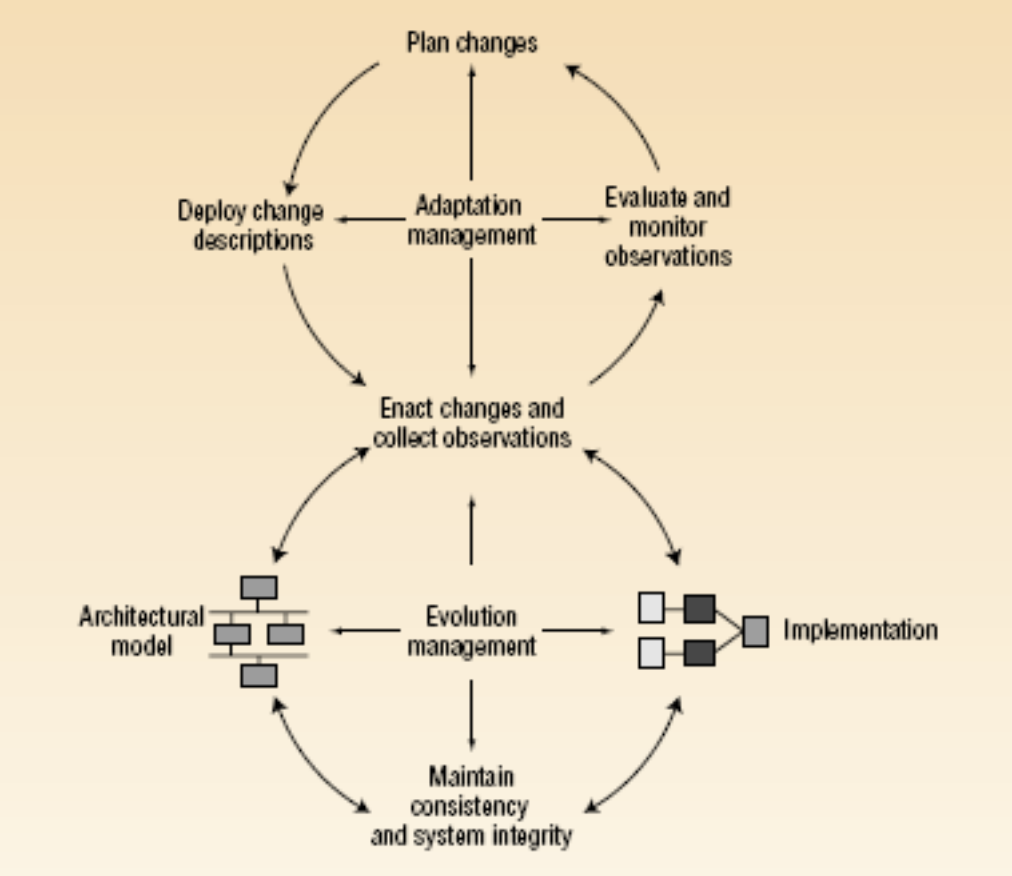

FIGURE 3.1 – Activités de haut niveau d'une approche générique pour les systèmes logiciels auto-adaptatifs

existants. Desnos et al. (Desnos *et al.*, 2007) considèrent la reconfiguration par substitution de composants. Ils proposent un modèle permettant de remplacer un composant par un ensemble de composants.

c) Auto-adaptation des applications à base de composants

Dans la majorité des cas, les applications à base de composants sont conçues de manière statique. Les développeurs choisissent des composants et les assemblent de manière à produire l'application désirée. Cependant, les environnements dynamiques nécessitent une adaptation de la configuration d'une application à l'exécution. Plusieurs travaux ont ainsi étudié des solutions pour l'adaptation dynamique d'applications à base de composants.

Oreizi et al. (Oreizy *et al.*, 1999) présentent une architecture générique pour les applications auto-adaptatives. Le fonctionnement de cette architecture est illustré par la figure 3.1. Cette architecture est constituée de deux niveaux. Le niveau inférieur est celui de la gestion de l'évolution : il est concerné par la possibilité de permettre la reconfiguration d'une application au cours de son exécution. Il doit ainsi prendre en charge deux missions :

– la *mise en place des changement*s consiste à agir sur l'architecture de l'applica-

tion pour réaliser les changements nécessaires. Cette tâche repose sur la possibilité de reconfigurer dynamiquement une architecture, comme indiqué dans la section précédente.
- le *maintien de la cohérence* consiste à vérifier la validité des changements proposés par rapport à diverses propriétés telles que la sécurité ou la disponibilité. Cet aspect est essentiel pour assurer le maintien en fonctionnement continu de l'application.

Le niveau supérieur est celui de la gestion de l'adaptation : il est concerné par la prise de décision quant aux adaptations à réaliser. Il prend en charge trois missions :
- la *collecte d'observations* sur le fonctionnement de l'application. C'est à partir de ces informations que peut être détectée la nécessité d'adapter l'application.
- l'*évaluation d'observation*s pour déterminer si une adaptation est nécessaire.
- la planification des changements à effectuer pour effectuer l'adaptation, c'est-à-dire le choix des actions à effectuer pour rendre le fonctionnement plus adaptée à la situation.
- le *déploiement des changements*, qui s'appuie sur le niveau inférieur pour mettre en place une nouvelle configuration.

L'utilisation d'une approche à base de composants permet ainsi de concevoir les applications auto-adaptatives en considérant l'adaptation comme une préoccupation séparée. Comme l'indiquent David *et al.* citeDavid2003dais, l'adaptation dépend essentiellement du contexte d'exécution de l'application, qui n'est pas nécessairement connu des concepteurs de l'application. Leur approche se fonde ainsi sur deux étapes dans la conception d'une application à base de composants :

1. Les développeurs créent le cœur de l'application, c'est-à-dire les fonctionnalités essentielles, sous la forme de composants .
2. L'utilisateur (administrateur ou installateur) définit les politiques d'adaptation au contexte dans lequel l'application est exécutée. Ces politiques prennent la forme de règles Évènement-Condition-Action (ECA). Ces règles sont attachées individuellement aux composants et exploitent le framework Fractal pour insérer, remplacer ou paramétrer dynamiquement des composants de l'application.

Afin de gérer la reconfiguration dynamique d'application en fonction du contexte, MaDcAr (Grondin *et al.*, 2006) propose une solution pour l'assemblage et le re-assemblage d'applications à base de composants. Dans cette approche, un moteur d'assemblage (*assembling engine*) utilise une description d'application dans laquelle les propriétés abstraites des composants (appelées rôles) sont exprimées. Le moteur détermine un assemblage correct de composants correspondant à cette description en prenant en compte différents éléments du contexte. Il utilise pour cela un mécanisme de résolution de contrainte, qui permet de déterminer une configuration de composants appropriée, c'est-à-dire qui respecte les contraintes imposées par la spécification de l'application et le contexte.

Une approche de reconfiguration d'une application à base de composant peut aussi être utilisée dans le cadre des interfaces humain-machine. On parle en particulier d'interfaces plastiques (Calvary *et al.*, 2001), qui sont capables de s'adapter à des conditions changeantes tout en maintenant continuellement l'interface utilisable. Pour réaliser des interfaces plastiques, Calvary et al. proposent un type particulier de composant appelé « Comet » (Calvary *et al.*, 2005). Pour faire face aux diverses demandes d'adaptation, les « Comets » supportent non seulement l'adaptation par un tiers mais possèdent aussi des capacités d'auto-adaptation internes.

d) Synthèse

Intérêts Par rapport à la composition de services Web, le mode de composition est plus approprié aux applications attentives. Les langages de description d'architecture permettent en effet de décrire une application sous la forme d'un assemblage

de composants qui interagissent durablement par l'intermédiaire de connecteurs. Ces modèles autorisent de plus la reconfiguration dynamique, en particulier permise par la modularité des composants. Il est ainsi envisageable de modifier le comportement de l'application en ajoutant, en supprimant ou en remplaçant certains de ses composants.

La modularité et la séparation des préoccupations au sein de composants bien définis augmentent aussi la tolérance aux pannes. En particulier, un problème isolé dans un composant peut être détecté, puis réparé par la ré-initialisation du composant ou par l'utilisation d'un composant fournissant une fonctionnalité équivalente.

Inconvénients L'inconvénient principal des approches de composants est la nécessité d'employer un modèle de composant homogène pour toutes les fonctionnalités. Chaque modèle de composant impose des contraintes particulières, qui permettent de garantir les propriétés des applications à base de composants. Cependant, il n'existe pas de modèle de composants susceptible d'être adopté par toutes les fonctionnalités d'un environnement attentif.

Une conséquence de l'emploi de modèle de composant homogène et d'interfaces fixes est l'évolution limitée des fonctionnalités de l'environnement. En particulier, l'introduction d'une nouvelle fonctionnalité n'est possible que si les interfaces qui permettent de l'utiliser sont déjà connues par d'autres composants du système.

On peut noter que les principes des architectures orientées-service peuvent être appliqués aux systèmes à base de composants pour faciliter le couplage faible et l'évolution indépendante des composants du système. Escoffier et al. (Escoffier & Hall, 2007) proposent ainsi un modèle de composant orienté-service, qui intègre les principes de l'architecture orientée-services dans un système à base de composants.

3.2.3 Mécanismes d'organisation flexible de systèmes multi-agents

Dans le cadre de l'approche agent, une application peut être conçue sous la forme d'un système multi-agents (SMA). Chaque agent fournit alors des capacités particulières et les agents se coordonnent pour pour réaliser une tâche et donner un comportement globalement cohérent à l'application. En raison du faible couplage entre les agents et de la décentralisation du contrôle, cette approche de conception favorise la flexibilité et la robustesse des applications. Les systèmes multi-agents sont ainsi un paradigme de programmation pertinent pour la conception de systèmes complexes (Luck *et al.*, 2005).

Cette section s'intéresse aux mécanismes qui permettent à un système multi-agents de s'adapter dynamiquement à des conditions changeantes. Elle introduit ainsi tout d'abord les principes de fonctionnement d'un système multi-agents dans lequel les agents coopèrent pour réaliser une tâche, puis mentionne les capacités d'adaptation locales d'un agent dans un tel système. Ensuite, elle détaille des mécanismes pour l'organisation flexible d'agents, qui permettent l'adaptation dynamique d'application. Enfin

a) Principes des systèmes multi-agents

Un système multi-agents peut être défini comme un système formé d'entités logicielles autonomes (appelées agents) qui travaillent ensemble pour accomplir une tâche qu'ils ne pourraient pas réaliser seuls (Jennings *et al.*, 1998). Un tel système est caractérisé par :

– l'absence d'informations ou des capacités permettant à un unique agent d'accomplir la tâche seul. Chaque agent a un point de vue limité sur la tâche et sur le fonctionnement du système.

– l'absence de contrôle global du système. Chaque agent agit de manière autonome selon ses connaissances individuelles.
– la décentralisation de l'information. Chaque agent possède des informations sur sa tâche, mais il n'existe aucun endroit où toutes les informations du système sont centralisées.
– les traitements sont asynchrones. Il n'existe pas de procédures d'exécution globale synchrone qui ordonne le fonctionnement des agents. L'exécution de traitements par chaque agent est déclenchée de manière indépendante par ses perceptions de l'environnement, ses interactions avec d'autres agents et ses objectifs propres.

Concevoir une application sous la forme d'un système multi-agents coopérant présente plusieurs avantages (Jennings *et al.*, 1998) :

- la *modularité* des différentes fonctionnalités qui sont réparties au sein des différents agents, possédant chacun des capacités et des mécanismes d'adaptation spécifiques. Cette modularité permet notamment de traiter des problèmes dans lesquels les informations ou le contrôle sont nécessairement distribués.

- *l'optimisation* des ressources et des capacités des différents agents, qui peuvent s'organiser à l'exécution pour réaliser la tâche globale de la manière la plus efficace possible. La séparation des différentes fonctionnalités permet à chaque agent de se concentrer sur une tâche spécifique et de prendre les décisions locales adaptées à l'accomplissement de cette tâche.

- la *robustesse* du fonctionnement est permise par la réduction des points critiques et la possibilité de remplacer un agent en cas de panne ou de répartir sa contribution sur les autres agents du système.

b) Autonomie et flexibilité locale des agents

En tant qu'entité logicielle, un agent se distingue d'un objet ou d'un composant par plusieurs caractéristiques :

Rationalité Un agent cherche généralement à effectuer des actions qui lui permettent de maximiser son utilité. L'utilité est une mesure de l'efficacité du comportement de l'agent. Il faut noter que dans certains cas, l'utilité d'un agent peut dépendre de l'utilité globale du système, ce qui le pousse à coopérer pour améliorer son efficacité. La notion de rationalité permet notamment de découpler les buts de l'agent des moyens qu'il peut mettre en œuvre pour les atteindre.

Autonomie Une des caractéristiques principale des agents est leur autonomie : la capacité individuelle de chaque agent à déterminer ses actions en fonction de ses propres objectifs. L'autonomie individuelle des agents est une des clés de la robustesse des systèmes multi-agents. L'autonomie permet une réelle décentralisation d'un système distribué et évite l'existence d'éléments critiques : il n'est pas nécessaire qu'un mécanisme de coordination centralisé commande les différentes entités du système. De plus, l'autonomie individuelle d'un agent lui permet de continuer à fonctionner même en cas de disparition ou de panne d'autres agents dans le système : il reste capable de prendre ses décisions, et parfois de contourner la défaillance d'un autre agent.

Sociabilité Dans un système multi-agents, un agent nécessite souvent la participation d'autres agents pour atteindre ses propres buts.

c) Organisation flexible d'un système multi-agents

Dans un SMA coopératif, il faut considérer deux aspects : la tâche globale que le système doit accomplir et les capacités locales de chacun des membres du système. Ces deux aspects sont articulés par l'organisation du SMA, qui définit comment les différents agents mettent leurs capacités à contribution pour réaliser la tâche globale.

L'organisation d'un SMA peut être définie de manière fixe, à la conception du système. Cependant, dans le cas d'environnements dynamiques et incertains, il est essentiel de disposer de mécanismes d'organisation flexible, c'est-à-dire de permettre la modification dynamique des relations entre les agents (Mathieu *et al.*, 2002). Grâce à de tels mécanismes, le système peut avoir un comportement plus robuste et faire face aux aléas rencontrés lors de son fonctionnement.

Équipes flexibles Les travaux sur les équipes flexibles (Tambe, 1997) (Hübner *et al.*, 2002) s'intéressent en particulier à la possibilité de réorganiser une équipe en fonction des événements qui peuvent survenir. Ces travaux se fondent sur un modèle de description de l'organisation de l'équipe, sur lequel les agents peuvent raisonner, et sur des mécanismes permettant de déterminer la réorganisation de l'équipe. Cette approche est particulièrement appropriée lorsqu'un système doit fonctionner de manière optimale dans tout un ensemble de situations.

Planification distribuée Dans le cas d'un environnement dans lequel les interactions entre les tâches à effectuer ne sont pas bien connues, il est nécessaire de fournir des mécanismes de coordination plus flexibles. GPGP (Lesser *et al.*, 2002) est un framework générique pour la coordination d'agents qui effectuent des activités ayant des interdépendances complexes. Dans GPGP, les agents échangent des descriptions de leurs activités. Ces descriptions sont exprimées dans le formalisme TÆMS, qui permet aux agents d'effectuer des raisonnements complexes sur les interactions positives ou négatives entre leurs activités. Les agents sont ainsi capables d'organiser leurs activités de manière à optimiser le fonctionnement global du système. De plus, les interactions entre les activités peuvent être découvertes dynamiquement, au cours de leur exécution, ce qui entraîne une réorganisation du système pour atteindre de nouveau une organisation cohérente.

Coordination par problème de satisfaction de contraintes Dans certains problèmes, il n'est pas possible aux agents d'échanger des informations précises sur leurs activités. C'est en particulier le cas lorsque ces informations sont privées ou lorsque les agents sont hétérogènes et ne possèdent pas de représentation unifiée de leurs activités. Dans ce cas, l'utilisation de mécanismes de coordination sous la forme d'un problème distribué de satisfaction de contraintes à été proposé (Yokoo & Hirayama, 2000). Dans cette approche, on modélise le choix d'une action par un agent sous la forme d'une affectation d'une valeur à une variable et les interactions entre les choix des agents sous la forme de contraintes entre les variables. Les agents échangent des informations sur leurs choix de manière à vérifier qu'ils sont cohérents avec les choix des autres agents. Dans cette approche, une modification dynamique des tâches des agents donne lieu à une nouvelle résolution du problème de satisfaction de contrainte distribué, qui permet la réorganisation du système. Yokoo et Hirayama (Yokoo & Hirayama, 2000) détaillent plusieurs algorithmes qui permettent d'obtenir des choix globalement cohérents au sein du système.

d) Ouverture d'un système multi-agents

La problématique du découplage entre les différentes fonctionnalités d'un système est aussi au cœur de la problématique des systèmes multi-agents. Ces systèmes sont naturellement faiblement couplés : le fonctionnement d'un agent n'est pas complètement dépendant d'autres entités du système. Dans ce cadre, des solutions ont été étudiées pour faciliter le découplage et autoriser la plus grande autonomie possible pour les agents.

Cette section décrit tout d'abord l'utilisation de langages de communication de haut-niveau, grâce auxquels des agents peuvent interagir indépendamment d'une ap-

plication précise. Elle s'intéresse ensuite aux architectures de systèmes multi-agents conçues pour favoriser l'ouverture du système. Enfin, elle aborde la problématique de découverte de capacités des agents dans le cas de systèmes ouverts où de nouveaux agents inconnus peuvent apparaître.

Langage standardisé pour de communication de haut-niveau L'utilisation de langage de communication de haut-niveau standardisé constitue un premier pas vers le découplage et l'ouverture d'un système multi-agents. Grâce à un tel langage, il devient en effet possible de concevoir des mécanismes de communication indépendamment d'une application particulière et des agents impliqués dans la communication.

FIPA-ACL (*FIPA-Agent Communication Language*, (FIPA, 2002a)) est le langage de communication standardisé proposé par la FIPA (*Foundation for Intelligent Physical Agents*). Ce langage se fonde sur la théorie des actes de langages, et vise ainsi à reproduire les types de messages échangés entre des agents humains. FIPA-ACL repose en particulier sur l'utilisation de performatifs standardisés. Un performatif indique la teneur du message et explicite l'effet attendu par l'émetteur du message. Ainsi, le performatif REQUEST indique que l'émetteur souhaite que son interlocuteur accomplisse une action. Le performatif INFORM indique que l'émetteur souhaite informer son interlocuteur d'une croyance particulière.

Divers langages de contenus peuvent être utilisés dans un message FIPA-ACL, mais le langage de contenu utilisé dépend du modèle d'agent employé et non de l'application : l'agent n'est pas spécifiquement programmé pour répondre aux messages d'une application, mais possède un mécanisme d'interprétation du sens du langage de contenu utilisé. Plusieurs langages de contenu sont basés sur des formalismes logiques : un agent, équipé des mécanismes de raisonnement appropriés, peut interpréter le contenu du message sans avoir explicitement été prévu pour recevoir ce message. En particulier, le langage FIPA-SL (FIPA, 2002c) s'appuie sur des constructions basées sur une logique modale qui exprime les croyances et les intentions d'un agent.

Architectures pour les systèmes d'agents ouverts L'architecture du système multi-agents intervient fortement dans sa capacité à accueillir de nouveaux agents. Un agent qui joint le système doit pouvoir proposer ses capacités à d'autres agents du système.

Découverte dynamique Une solution pour résoudre le problème de connexion dans les systèmes multi-agents est d'utiliser une architecture à base de facilitateur : le facilitateur est un agent particulier qui joue le rôle d'intermédiaire entre les agents. L'architecture standard FIPA définit un type simple de facilitateur : le *directory facilitator* (DF). Cette agent joue le rôle de pages jaunes : un agent s'adresse à lui pour obtenir le nom (et l'adresse) d'un agent répondant à certaines caractéristiques.

Courtage Open agent Architecture (Martin *et al.*, 1999) est un exemple de système reposant sur un mécanisme de facilitateur afin de permettre l'inter-opération et la coordination entre des agents indépendants. Des agents peuvent entrer dans le système au cours de son fonctionnement et renseigner le facilitateur sur leurs capacités et le vocabulaire qu'ils comprennent. Lorsqu'une tâche est demandée par un utilisateur, le facilitateur se charge de déterminer les agents qui peuvent participer à cette tâche, et organise leurs interactions pour mener à bien cette tâche.

Architecture décentralisée Les architectures à base de facilitateur présentent l'inconvénient de créer un élément critique dont une panne empêche le système de fonctionner. Afin d'éviter ce problème, des architectures décentralisées ont été proposées : dans ces architectures, la capacité à incorporer un agent au système

est distribuée au sein des différents agents. Vercouter (Vercouter, 2000) propose un tel système, fondé sur la notion d'agent accueillant.

Communication indirecte Un dernier type d'architecture pour faciliter l'ouverture des systèmes multi-agents est l'utilisation de mécanismes de communication indirecte. Dans ces mécanismes, les agents ne s'adressent pas directement des messages mais communiquent par l'intermédiaire d'un canal de communication partagé (Busetta *et al.*, 2002). Cette approche permet à un agent d'intégrer un système en écoutant le canal de communication et en proposant ses services lorsqu'il pense qu'un autre agent en a besoin.

Découverte et représentation de capacités d'agents inconnus Afin de permettre une réelle ouverture d'un système multi-agents, il importe que les agents se représentent les capacités d'autres agents qu'ils ne connaissent pas *a priori*, afin de déterminer comment interagir avec eux.

Une solution repose sur l'explicitation des capacités à l'aide d'un langage de description de capacité. Un tel langage permet à chaque agent de décrire ses capacités, et aux autres agents de raisonner sur ces descriptions. Les langages CDL (*Capability Description Language*) (Wickler, 1999) et LARKS (Sycara *et al.*, 2002) sont des exemples de langages de descriptions de capacités d'agents. On peut noter que cette approche a été reprise dans le cadre des services Web sémantiques (section 3.1.2), en limitant toutefois l'expressivité des langages utilisés.

La représentation statique pose souvent le problème d'équilibre entre la richesse de la représentation et la quantité d'information à communiquer. D'autres approches visent ainsi à construire progressivement une connaissance sur les capacités d'un agent en interagissant avec lui.

e) Synthèse

Intérêts Les systèmes multi-agents fournissent un cadre particulièrement adapté pour garantir la robustesse des applications. Ils regroupent ainsi les intérêts des deux autres approches :

- Le déploiement est facilité par le découplage entre la tâche globale et les capacités locales des agents. Comme dans le cas de la composition de services Web, une application peut donc être définie indépendamment de fonctionnalités particulières et assemblée dynamiquement.
- La dynamicité est prise en compte, aussi bien au niveau de chaque agent que de l'organisation du système. En particulier, les mécanismes de réorganisation permettent de faire face aux changements de situation en réorganisant les tâches des différents agents.
- La tolérance aux pannes est largement favorisée par la décentralisation du système et par l'autonomie de chaque agent. Comme dans le cas des composants, il est possible de remplacer un agent défectueux par un autre agent fournissant les mêmes fonctionnalités. De plus, la décentralisation du système évite l'existence de points critiques dont la panne entraîne l'arrêt de fonctionnement.
- L'évolution de l'environnement et des fonctionnalités est prise en compte, en particulier dans le cas de SMA ouverts. Dans ce cas, les hypothèses sur la présence et sur les caractéristiques des fonctionnalités sont réduites. Le système détermine dynamiquement l'état de l'environnement, qui peut donc évoluer au cours du fonctionnement.

Inconvénients Une limitation majeure à l'utilisation de systèmes multi-agents dans le cadre des environnements attentifs est la complexité de mise en œuvre des solutions existantes. Il est ainsi difficilement envisageable d'imposer l'utilisation d'agents pour fournir toutes les fonctionnalités de l'environnement.

De plus, il apparaît que les mécanismes permettant la réorganisation et l'ouverture du système nécessitent de quantités importantes de ressource de calcul et de mémoire, au niveau de chacune des fonctionnalités. De telles ressources ne sont généralement pas disponibles dans les dispositifs qui composent un environnement attentif.

3.3 Implication des utilisateurs dans l'adaptation

Les applications attentives posent des problèmes particuliers en terme de définition *a priori* des besoins à remplir par une application. Contrairement aux applications classiques, où l'explicitation des besoins est la première étape de la conception, dans un environnement attentif les besoins ne peuvent être explicités par l'utilisateur qu'en fonction du contexte dans lequel l'application est utilisée. Dans ces conditions, une application doit être dynamiquement adaptable aux besoins et au contexte particulier d'un utilisateur. L'adaptation doit cependant être équilibrée entre des mécanismes automatiques, qui épargnent l'attention de l'utilisateur, et des mécanismes de contrôle direct, qui assurent la maîtrise de l'environnement par l'utilisateur. Cependant, de nombreuses solutions présentées dans la section 2.3 considèrent soit que l'utilisateur peut définir complètement le fonctionnement selon ses besoins, soit que le système est capable de déterminer automatiquement les besoins et d'y répondre de manière satisfaisante.

Dans le cadre des approches de composition dynamique d'application, les mécanismes de composition sont mis en œuvre à l'exécution : il est ainsi possible de prendre en compte les aspects précis des besoins et du contexte, afin notamment d'éviter les interférences entre les différentes applications. Il importe de rendre ces mécanismes participatifs et interactifs, et les besoins des utilisateurs peuvent être pris en compte de manière flexible, à plusieurs niveaux :

1. La définition de l'application peut être personnalisée par chaque utilisateur selon ses besoins propres. En effet, on se situe dans une approche *bottom-up*, dans laquelle la définition des besoins n'est pas nécessairement préalable au développement des fonctionnalités élémentaires.
2. Au cours du fonctionnement des mécanismes de composition, l'utilisateur peut être sollicité sur les choix à faire et interagir ainsi avec un mécanisme de composition semi-automatique.
3. L'adaptation de l'application peut être déclenchée non seulement par des décisions automatiques mais aussi par l'utilisateur lui-même. En particulier, un utilisateur peut contrôler l'adaptation de l'application de manière directe (en modifiant ces besoins) ou indirecte (en arrêtant une fonctionnalité qu'il ne souhaite pas utiliser).

Parmi ces trois possibilités d'implication d'un utilisateur dans la composition dynamique d'application, l'implication dans la définition initiale du besoin est la plus répandue, et certaines applications ont été décrites dans la section 2.3.2). C'est aussi la moins flexible, car elle nécessite de pouvoir modéliser explicitement les besoins des utilisateurs. Cette section s'intéresse ainsi à des approches qui permettent l'implication tout au long du fonctionnement de l'application, de manière à prendre en compte l'imprécision et l'évolution des besoins. La section 3.3.1 présente des mécanismes de composition semi-automatique, dans lesquels l'utilisateur est impliqué dans les décisions du système. La section 3.3.2 aborde l'approche des systèmes multi-agents mixtes, dans lesquels les utilisateurs sont explicitement considérés comme des agents du système et participent donc directement à son fonctionnement.

3.3.1 Mécanismes de composition semi-automatique

La plupart des mécanismes de composition d'application présentés dans la section 3.2 sont des mécanismes complètement automatiques. Dans ces systèmes, il est ainsi uniquement possible d'impliquer un utilisateur lors de la définition initiale de l'application à composer. Ce type d'implication est limité car il ne prend pas en compte l'imprécision et l'évolution des besoins.

Cette section présente des approches alternatives qui s'intéressent à la composition semi-automatique d'applications. Cette section présente tout d'abord des travaux portants sur la composition semi-automatique de services Web. Elle s'intéresse ensuite plus généralement à des approches de planification interactive, qui peuvent aussi s'appliquer à la composition dynamique d'application.

a) Composition semi-automatique de services Web

Dans le cadre des services Web, Sirin et al. (Sirin *et al.*, 2004) définissent les principes de fonctionnement d'un système de composition semi-automatique. Ce système est fondé sur la construction graduelle d'une orchestration de services Web. Dans cette approche, chaque nouveau service Web est ajouté successivement, et en suivant plusieurs étapes :

1. définition abstraite de la fonctionnalité recherchée.
2. découverte : le système découvre tous les services correspondant à la description abstraite de la fonctionnalité recherchée (par exemple, un service de réservation de transport).
3. filtrage : parmi les services découverts, certains doivent être filtrés s'ils ne correspondent pas à la tâche (par exemple, s'il n'offre pas de transport pour la destination voulue) ou s'ils violent une préférence de l'utilisateur (par exemple, un prix maximum).
4. compatibilité : parmi les services restants, seuls peuvent être utilisés ceux qui sont compatibles avec les services déjà sélectionnés pour accomplir d'autres étapes de l'orchestration.

Chacune des étapes peut être en partie automatisée grâce à l'utilisation de description sémantique des services Web (exprimé avec OWL-S). En particulier, les incompatibilités sont déterminées à l'aide des informations contenues dans les descriptions et de mécanismes de raisonnement sur les ontologies. La description sémantique est ainsi utilisée pour filtrer automatiquement certaines possibilités, mais ne sélectionne pas automatiquement un seul service possible. Cette approche a l'avantage d'éviter de faire l'hypothèse que les descriptions du service et de la fonctionnalité recherchée sont suffisamment expressives pour effectuer la sélection de manière automatique et déterministe.

b) Planification interactive

De nombreuses solutions pour la composition automatique d'application sont basées sur des techniques de planification. Dans ce domaine, des travaux étudient par ailleurs la problématique de planification interactive, dans laquelle un système de planification interagit avec un utilisateur au cours de son fonctionnement. Dans ces approches, un utilisateur (généralement expert d'un domaine) est aidé plutôt que remplacé par le système automatique.

La nécessité d'intégrer des mécanismes qui permettent un contrôle personnalisé sur la planification est notamment apparue lors de l'application de techniques de planification issues de l'intelligence artificielle à des problèmes réels. O-Plan (Drabble & Tate,

1996) est un exemple de système de planification utilisé pour des applications réelles, en particulier dans le domaine militaire. Afin de répondre à la complexité et à la criticité des décisions, O-Plan peut impliquer un utilisateur (expert) dans le mécanisme de planification. Pour cela, O-Plan propose des alternatives à l'utilisateur lorsqu'il n'est pas en mesure de prendre seul les décisions. Le fonctionnement de O-Plan est fondé sur un réseau de tâches hiérarchiques (HTN), qui permet de décomposer le problème, et sur la définitions de contraintes portants sur les actions choisies.

Nareyek *et al.* (Nareyek *et al.*, 2005) soulignent l'intérêt de l'utilisation de mécanismes de satisfaction de contraintes pour l'application des systèmes de planification à des problèmes réels. En effet, de nombreux aspects des problèmes réels sont difficiles à modéliser en utilisant les formalismes manipulés par les algorithmes de planification, mais peuvent être modélisés plus simplement dans les formalismes des problèmes de satisfaction de contraintes (Nareyek *et al.*, 2005). Nareyek *et al.* détaillent ainsi plusieurs travaux dans lesquels tout ou partie d'un problème de planification est modélisé sous la forme d'un graphe de contraintes. Ils notent que l'utilisation d'une modélisation basée sur les graphes de contraintes est particulièrement intéressante pour gérer l'interactivité. En particulier, il existe des techniques d'explication des mécanismes de raisonnement (Sqalli & Freuder, 1996) et des mécanismes de gestion de graphes de contraintes dynamiques (Verfaillie & Schiex, 1994), grâce auxquels un graphe de contraintes peut-être modifié de manière incrémentale et refléter ainsi un changement du problème de planification. Par ailleurs, cette approche a aussi des avantages pour favoriser la gestion de l'imprécision, de la distribution et la facilité d'utilisation.

Les approches par propagation de contraintes constituent une catégorie particulière de techniques de planification interactive reposant sur une formalisation sous forme de problème de satisfaction de contraintes. Ces techniques consistent à construire la solution d'un problème complexe de manière incrémentale, en introduisant progressivement les contraintes du problème et en interagissant avec l'utilisateur pour les résoudre. Lamma *et al.* (Lamma, 1999) proposent ainsi un framework pour les problèmes de satisfaction de contraintes interactifs (ICSP, *interactive constraint satisfaction problems*) qui entremêlent l'acquisition de valeurs pour chaque variable et la vérification des contraintes. Herakles II (Ambite *et al.*, 2005) est un exemple de système fondé sur un réseau de contraintes conditionnelles et sur la propagation de contraintes, qui combine la planification, l'interaction et l'acquisition d'information.

c) Synthèse

Intérêts Dans le cadre de la composition d'applications, l'utilisation d'une approche de planification interactive est particulièrement intéressante pour permettre une meilleure personnalisation du fonctionnement. L'utilisateur peut en effet non seulement définir le besoin initial, comme dans le cas de la composition, mais il est aussi impliqué dans certaines décisions. Cela lui garantit un contrôle approprié sur le fonctionnement de l'application produite, en laissant le système résoudre automatiquement certains aspects plus précis. En particulier, lorsque des descriptions sémantiques des fonctionnalités sont disponibles, un système semi-automatique peut prendre en charge la résolution de l'interopérabilité entre les fonctionnalités.

Inconvénients Un inconvénient des approches de planification interactive concerne la robustesse de fonctionnement. En effet, le système ne modélise pas nécessairement le besoin auquel l'application répond et il lui est donc impossible de s'adapter dynamiquement pour fournir l'application souhaitée lorsque l'environnement change. La délégation est ainsi relativement faible et le fonctionnement du système peut nécessiter de fréquentes interventions de l'utilisateur, ce qui augmente sa charge cognitive.

3.3.2 Implication des utilisateurs dans un système multi-agent

Les systèmes multi-agents autorisent une approche plus profonde de l'implication des humains dans le fonctionnement du système. Un utilisateur peut en effet être considéré explicitement comme un agent du système. Dans ce cas, ses connaissances, son comportement et ses actions peuvent influencer le comportement des autres agents du système, en particulier des agents logiciels.

Cette section évoque les possibilités offertes par l'implication d'agents humains dans un système multi-agents. Elle s'intéresse tout d'abord à quelques exemples de systèmes mixtes humains-agents. Dans un deuxième temps, elle décrit les nouvelles possibilités offertes par l'approche du Web sémantique dans le cadre de systèmes humain-agents.

a) Systèmes mixtes humain-agent

Certains travaux se sont intéressés à la construction de systèmes mixtes humain-agents, dans lesquels les humains sont explicitement considérés comme des agents du système.

Un des objectifs de l'architecture OAA (*Open Agent Architecture*, (Martin *et al.*, 1999)) est l'intégration « sans couture » des agents logiciels et des utilisateurs humains. Les utilisateurs sont directement et consciemment en interaction avec des agents logiciels. Ils peuvent en particulier leur communiquer des ordres directs, et les agents peuvent leur adresser directement des réponses. OAA fournit aux utilisateurs des moyens de spécifier aux agents les tâches à effectuer, à l'aide de moyens d'interaction naturels tels que la parole. Par exemple, un utilisateur peut téléphoner à un agent pour lui demander la météo prévue dans la ville où il se trouve. Un second aspect de l'intégration est le support pour la collaboration entre humains et agents logiciels : les deux types d'agents peuvent travailler simultanément sur des ressources et des données partagées (Martin *et al.*, 1999).

D'autres travaux se sont intéressé de manière plus fondamentale à la manière dont les humains et les agents peuvent coopérer. Par exemple, Pauchet et al. (Pauchet *et al.*, 2007) proposent un système multi-agents capable de simuler les mécanismes de résolution coopérative de problèmes utilisés par les humains. Une perspective de tels travaux est la production de systèmes multi-agents capables de coopérer de manière intuitive avec des humains. Charton et al. (Charton *et al.*, 2003) s'intéressent aussi à la collaboration entre des agents humains et des agents logiciels, et notamment à l'adaptation des agents logiciels aux agents humains. Cette adaptation est essentielle dans le cas où ces agents sont hétérogènes et ne savent pas *a priori* comment interagir. La solution proposée repose sur des agents médiateurs, capables de s'adapter aux différents agents (humains et logiciels) pour établir une médiation entre eux. L'adaptation est basée sur des séquences d'interactions entre un humain et un agent fournissant un service, tel que la réservation d'un vol. L'agent médiateur utilise une modélisation probabiliste de la tâche à accomplir (la réservation du vol) et interagit avec les autres agents pour déterminer les paramètres du modèle de leurs préférences.

b) La médiation humain-agent dans le Web sémantique

Le Web sémantique (Berners-Lee *et al.*, 2001) a pour objectif de rendre les informations accessibles sur le Web interprétable par des agents logiciels. Pour cela, les connaissances doivent être exprimées de manière formelle, en particulier en utilisant des ontologies.

Une conséquence de la vision du Web sémantique est de faciliter l'interaction entre des agents humains et des les agents logiciels, par l'intermédiaire des connaissances décrites sur le Web. Rousset (Rousset, 2004) indique l'intérêt de considérer le Web

sémantique comme un système pair-à-pair de gestion d'informations (*Peer Data Management System*). Cette vision privilégie une gestion décentralisée de la définition d'ontologies, dans laquelle chaque source d'information peut décrire les informations fournies à l'aide d'ontologies qui lui sont propres. L'objectif des outils du Web sémantique est de fournir une médiation entre ces ontologies hétérogènes, et non de chercher à définir des ontologies suffisamment expressives et génériques pour être réutilisées par tous. Cette approche encourage ainsi la définition d'ontologies simples, qui peuvent être comprises de la même manière par des agents logiciels mais aussi par des humains.

Comme le montre Gruber (Gruber, 2006), cette intégration des humains et des agents logiciels dans le cadre du Web sémantique est notamment rendue possible par les outils et les pratiques existant actuellement dans le cadre du Web social (aussi appelé Web 2.0). Dans le Web social, de nombreux outils informatiques facilitent l'échange de connaissances entre des humains, sous la forme de langage naturel. Cependant, la mise en œuvre de ces échanges à grande échelle repose sur l'utilisation de solutions plus formelles pour exprimer les connaissances.

Dans le domaine du Web sémantique, des solutions similaires existent pour favoriser l'implication des utilisateurs dans l'expression des connaissances. Krötzsch et al. (Krötzsch *et al.*, 2006) décrivent ainsi le Semantic MediaWiki. Il s'agit d'une extension du système MediaWiki, sur lequel repose en particulier l'encyclopédie collaborative Wikipedia. Cette extension permet l'intégration d'outils du Web sémantique dans le cadre de l'utilisation d'un wiki. Noy et al. (Noy *et al.*, 2006) décrivent un outil permettant l'édition et l'évolution d'ontologies dans un environnement collaboratif. Un tel outil simplifie la négociation entre utilisateurs humains concernant les ontologies utilisées pour décrire un domaine, et autorise l'évolution des ontologies en fonction des nouveaux besoins.

c) Synthèse

Intérêts L'approche de l'intégration des humains et des agents logiciels au sein de SMA mixte est intéressante à plusieurs points de vue :

- la personnalisation est permise à la fois par la définition initiale des besoins à remplir par le système et par des ajustements ponctuels des demandes aux agents.
- la délégation est facilitée par les comportements autonomes des agents, qui peuvent prendre certaines décisions et effectuer certaines adaptations sans nécessiter d'interaction directe. En particulier, la notion de but caractéristique de l'approche agent est utile pour modéliser explicitement le transfert d'intention de l'utilisateur au système.
- le contrôle par l'utilisateur est assuré par les mécanismes sociaux des agents, qui permettent de modéliser explicitement la prédominance de l'utilisateur sur les agents. De plus, les solutions apportées par le Web sémantique permettent d'envisager une médiation entre les humains et les agents.
- La gestion des interférences est assurée par les mécanismes de coordination et de négociation entre les agents.

Inconvénient Il n'existe pas véritablement de solution générique permettant l'intégration d'humains et d'agents au sein d'un système mixte. Le domaine du Web sémantique fournit des réalisations plus avancées, mais elles se focalisent généralement sur des activités de recherche d'information.

3.4 Synthèse

Ce chapitre explore différentes approches basées sur la notion de composition dynamique d'application et qui fournissent des solutions intéressantes pour la conception

d'applications attentives.

Tout d'abord, l'approche orientée-service semble intéressante pour permettre *l'intégration de fonctionnalités hétérogènes*. Grâce à l'utilisation de standards et d'une architecture bien définie, il est possible de construire des applications à partir de fonctionnalités hétérogènes présentées comme des services. L'approche adoptée par les services Web sémantiques complète l'approche orientée service en utilisant des techniques de représentation des connaissances pour faciliter l'intégration de fonctionnalités qui n'ont pas été prévues pour fonctionner ensemble. Cependant, on remarque que les solutions développées dans ces deux domaines se limitent à un modèle de service très simple, avec lequel seules des interactions ponctuelles de type requête-réponse sont possibles.

Concernant *l'adaptation à un environnement évolutif*, on a présenté trois approches, qui ont chacune leurs intérêts et leur limitations. Tout d'abord, l'approche de composition automatisée de services Web est intéressante pour faciliter le déploiement automatique d'application en intégrant les fonctionnalités présentes dans l'environnement. De plus, l'utilisation de descriptions explicites des fonctionnalités à l'aide de descriptions sémantiques permet de prendre en compte l'évolution de l'environnement. Cependant, cette approche se fonde sur un modèle de composition orienté processus, qui n'est pas très adapté aux applications attentives. À l'inverse, l'approche de reconfiguration d'applications à base de composant est basée sur un modèle dans lequel les fonctionnalités peuvent interagir sur des périodes plus longues, et la composition peut être révisée au cours du fonctionnement. En revanche, ces approches ne permettent pas de prendre en compte simplement l'évolution de l'environnement. Enfin, il apparaît que l'approche multi-agents est particulièrement adaptée pour concevoir des applications robustes dans un environnement évolutif. Elle présente cependant l'inconvénient d'être plus complexe et de ne pas pouvoir être appliquée aux fonctionnalités dont les ressources sont trop contraintes. Finalement, il semble qu'une combinaison de ces approches soit nécessaire pour la conception d'applications attentives.

Concernant *l'implication des utilisateurs*, on a présenté deux approches qui étendent les solutions étudiées pour l'adaptation dynamique en permettant un meilleur équilibre entre l'autonomie du système et le contrôle par l'utilisateur. Tout d'abord, de nombreux travaux ont étudié des solutions de planification interactive, qui permettent d'impliquer l'utilisateur dans certaines décisions d'un sytsème automatique. Ces solutions peuvent être envisagées pour la conception de mécanismes de composition semi-automatiques, mais présentent l'inconvénient de nécessiter une implication synchrone et relativement forte de l'utilisateur. Les solutions envisageables dans le cas de systèmes mixte humain-agent permettraient une implication plus subtile, où l'utilisateur influence le comportement du système de manière directe ou indirecte. En particulier, il apparaît que certains travaux menés dans le cadre du Web sémantique permettent d'envisager l'utilisation de descriptions sémantiques de haut niveau comme une médiation entre les humains et les agents logiciels. Ce type de médiation ne serait en particulier pas totalement synchrone, dans la mesure où les agents peuvent utiliser les informations précédemment fournies par les utilisateurs.

Chapitre 4

Synthèse de l'état de l'art

Les chapitres de cette partie ont détaillé l'état de l'art sous deux aspects. Tout d'abord, le chapitre 2 a examiné la notion d'application attentive et précisé les problématiques liées à la conception de telles applications. Ensuite, le chapitre 3 a étudié la notion de composition et d'adaptation d'applications dans d'autres domaines de l'informatique, afin de dégager des solutions permettant de répondre aux problématiques des applications attentives.

Ce chapitre synthétise cette étude de l'état de l'art et introduit la proposition détaillée dans la partie suivante.

4.1 Constats

L'étude de l'état de l'art a montré qu'il existe peu de solutions offrant un support générique pour la conception d'applications attentives. La plupart des applications existantes restent des prototypes de laboratoire, développés spécialement dans un environnement particulier. Ainsi, les solutions développées restent cantonnées à un domaine ou une application particulière. Pourtant l'étude menée dans le chapitre 2 indique que la conception d'applications attentives se heurte à plusieurs problématiques génériques :

- les dispositifs qui constituent un environnement domestique sont hétérogènes, spécifiques à cet environnement et évoluent au cours du temps. En conséquence, les concepteurs d'application doivent limiter les hypothèses sur les fonctionnalités disponibles dans un environnement particulier.
- l'environnement d'exécution est distribué, dynamique et imprévisible. En conséquence, les concepteurs d'applications ne peuvent restreindre le cadre de fonctionnement à des situations bien définies.
- la notion de besoin de l'utilisateur est imprécise et évolutive. En conséquence, il est difficile pour des concepteurs d'application de définir *a priori* le fonctionnement d'une application de manière précise. Il est nécessaire de permettre une personnalisation et un contrôle fin des applications par les utilisateurs en cours d'exécution.

Afin de tenir compte de ces caractéristiques, il semble important qu'un environnement attentif domestique soit muni d'une infrastructure offrant un support pour :

- permettre l'*intégration de fonctionnalités hétérogènes*, en la rendant si possible automatique.
- faciliter la *conception d'applications robustes*, qui s'adaptent aux situations rencontrées au cours de leur utilisation.

– fournir aux utilisateurs une *compréhension et un contrôle du fonctionnement* de l'environnement.

Certains des travaux étudiés dans le chapitre 2 s'intéressent à ce type d'infrastructure. Cependant, il est apparu que l'adéquation entre les propositions existantes et les besoins de concepteurs d'applications reste imparfaite. D'un autre côté, il apparaît que de nouvelles solutions peuvent être apportées par diverses technologies issues de différents domaines de l'informatique. En particulier, le chapitre 3 a détaillé comment les trois grandes problématiques des applications attentives peuvent bénéficier des solutions issues des domaines suivants :

L'architecture orientée-service permet le développement de systèmes logiciels dans lesquels les fonctionnalités sont faiblement couplées. Ceci permet une plus grande réutilisabilité des fonctionnalités, ainsi qu'une prise en charge d'une partie de leur hétérogénéité. Les approches de composition de services permettent de concevoir simplement des applications répondant à un besoin précis.

Le Web sémantique fournit des solutions essentielles pour la gestion de l'hétérogénéité et de l'ouverture des systèmes. Grâce à l'utilisation de modèles de connaissances formalisés, il devient possible d'utiliser des outils génériques de raisonnement sur les connaissances et de médiation entre des connaissances hétérogènes.

Les systèmes multi-agents fournissent de nombreuses solutions pour la conception de systèmes robustes et adaptatifs. Ces propriétés sont essentielles pour faire face à la dynamicité des environnements attentifs et à l'imprécision des besoins des utilisateurs. Les systèmes multi-agents permettent de plus d'envisager des systèmes ouverts et évolutifs, dans lesquels de nouvelles fonctionnalités peuvent apparaître au cours du temps.

Les résultats issus de ces domaines et les technologies associées sont assez peu exploités dans les environnements attentifs domestiques, principalement en raison de leur relative complexité de mise en œuvre. L'intégration de ces contributions dans l'infrastructure même d'un environnement attentif est une solution à envisager pour apporter leurs bénéfices de manière transparente aux concepteurs d'applications.

4.2 Architecture de référence

À la lumière de l'étude de l'état de l'art, il devient possible d'esquisser les grandes lignes d'une infrastructure pour les environnements attentifs, qui fournit un support générique pour faire face à certaines des problématiques de conception des applications attentives. La figure 4.1 illustre ainsi une architecture de référence pour les environnements attentifs, et positionne les différents éléments d'une infrastructure générique.

Cette figure présente une distinction entre espace physique et système informatique, caractéristique des environnements d'informatique diffuse dont les environnements attentifs sont un sous-ensemble. L'espace physique comprend les éléments suivants :

Utilisateur Un utilisateur d'un environnement attentif est une personne qui interagit avec des applications par l'intermédiaire de ses actions sur l'espace physique et sur les objets qui le composent. Selon les cas, l'utilisateur peut interagir de manière directe, explicite et consciente, ou bien être assisté par l'environnement de manière indirecte, implicite et périphérique.

Objet communicant Il s'agit de l'élément essentiel de l'environnement attentif, puisqu'il assure le lien entre le monde physique et le monde informatique, entre l'utilisateur et les applications. Les objets communicants couvrent toute sorte de dispositifs électroniques capables de communiquer avec d'autres éléments du

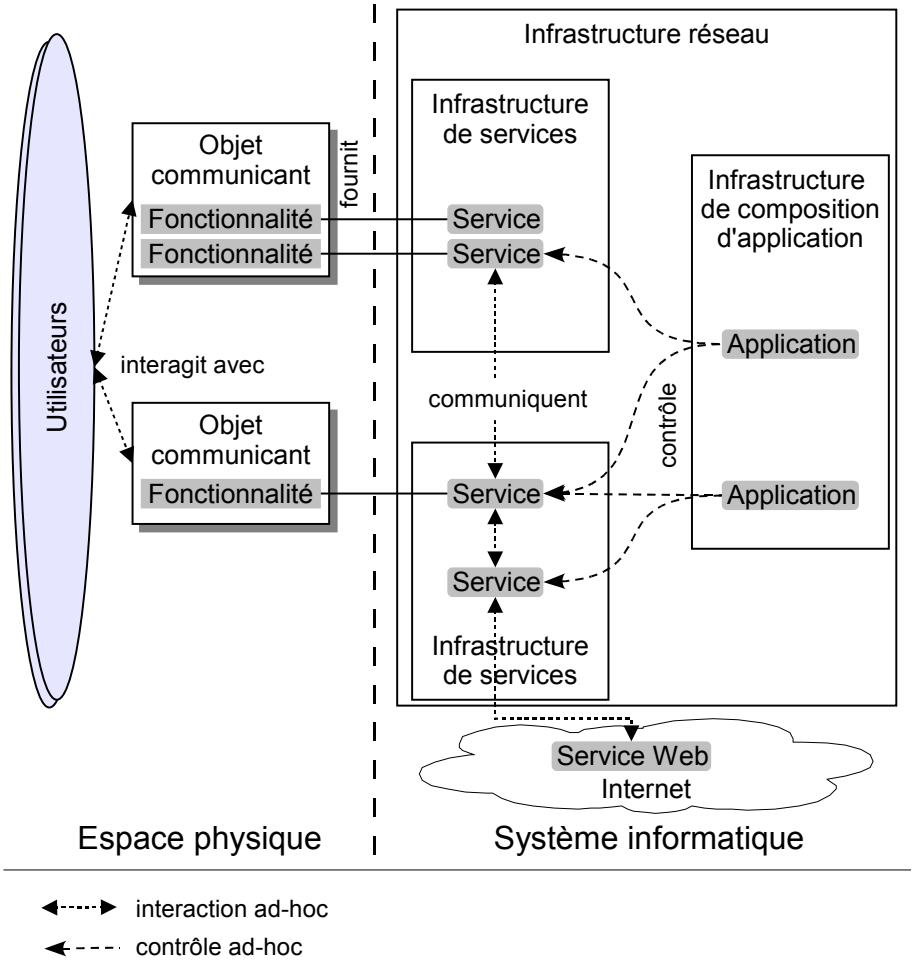

FIGURE 4.1 – Architecture de référence d'un environnement attentif domestique

système, aussi bien au travers d'un réseau filaire que sans fil. On considère en particulier des capteurs, des actionneurs et des dispositifs d'interfaces avec l'utilisateur. Au niveau de l'infrastructure globale, aucune hypothèse ne doit être faite sur le fonctionnement interne de ces objets communicants.

Fonctionnalité Chaque objet communicant de l'environnement a pour finalité de fournir des fonctionnalités élémentaires qui pourront être exploitées par l'environnement attentif. Par exemple, la fonctionnalité d'un capteur de luminosité est de fournir une mesure de la luminosité à l'endroit où il se trouve. Un objet communicant plus élaboré comme un cadre photo numérique peut fournir plusieurs fonctionnalités : afficher des photos ou communiquer des messages textuels à un utilisateur. En général une application intéressante pour l'utilisateur requiert la combinaison de plusieurs fonctionnalités élémentaires.

Le système informatique comprend les entités suivantes :

Service Le service est l'élément de base du système informatique de l'environnement. Le service donne accès à une fonctionnalité accessible sur le réseau domestique.

Pour cela, un service définit les interfaces qui permettent d'utiliser la fonctionnalité, décrit ces interfaces de manière compréhensible par d'autres systèmes et permet la découverte dynamique de la fonctionnalité. Dans un environnement attentif, il est ainsi possible d'utiliser les fonctionnalités des objets communicants au travers des services qu'elles fournissent. Un environnement attentif comporte aussi d'autres types de services, en particulier les services Web, qui sont des services disponibles sur Internet.

Application Une application coordonne un ensemble de fonctionnalités de manière à répondre à un besoin particulier d'un utilisateur. Une application exploite les services fournis par les fonctionnalités des objets communicants et les met en relation de manière à produire un comportement approprié pour répondre au besoin auquel elle s'adresse. Le rôle principal d'une application est donc la coordination de fonctionnalités disponibles dans l'environnement de manière pertinente pour son utilisateur.

Les entités du système informatique reposent sur plusieurs infrastructures, qui apportent des solutions génériques pour les tâches essentielles :

Infrastructure réseau Contrairement aux systèmes informatiques traditionnels, dans lesquels les fonctionnalités sont localisées sur une même machine, les fonctionnalités de l'environnement sont distribuées dans l'espace et les services correspondant sont distribués dans des réseaux informatiques. Ces réseaux ne sont pas nécessairement homogènes : dans un même environnement peuvent cohabiter diverses technologies de communication, en particulier sans fil. Pour former un environnement intégré, ces réseaux doivent cependant être inter-connectés et disposer de dispositifs passerelles, c'est-à-dire capable de communiquer sur plusieurs réseaux. Par ailleurs, ces réseaux ne sont pas nécessairement fiables : des déconnexions ou des débits de communication limités peuvent advenir.

Infrastructures de services Le fonctionnement d'un service repose sur une infrastructure de services, qui offre les moyens nécessaires aux services pour publier leur description et établir des communications avec les autres. On peut noter qu'un même environnement peut reposer sur plusieurs infrastructures de service. En effet, les besoins et les capacités des services sont variés, et il n'existe actuellement pas une infrastructure de services unique, capable de répondre à toutes les situations. Chaque fournisseur de services est libre de faire appel à une infrastructure particulière.

Infrastructure de composition d'application L'infrastructure de composition d'application fournit des solutions génériques pour faciliter la conception d'applications capables de répondre aux problématiques d'un environnement attentif. Elle fournit en particulier des solutions pour sélectionner les fonctionnalités appropriées, les assembler et prendre en compte le contexte de l'environnement. Chaque application peut ainsi être définie de manière indépendante à un environnement particulier, en se focalisant sur le besoin auquel l'application répond. L'infrastructure de composition d'application se charge de la mise en œuvre de l'application.

4.3 Vers une infrastructure de gestion de composition d'applications

Sur la base de l'architecture de référence dérivée de l'état de l'art, nos travaux proposent la réalisation d'une infrastructure de gestion de composition d'applications sous la forme d'un intergiciel de gestion de composition flexible d'application, appelé FCAP (*Flexible Composite Applications*). Cet intergiciel est basé sur le principe

de la « composition flexible », selon lequel une application est composée d'un ensemble de fonctionnalités dont les constituants et l'organisation évoluent en fonction de l'évolution de l'environnement et des besoins. FCAP apporte ainsi aux concepteurs d'applications des solutions pour :

- permettre l'intégration transparente de fonctionnalités fournies par des objets communicants hétérogènes, provenant de fournisseurs divers, et susceptibles d'évoluer au cours du fonctionnement des applications.
- permettre la conception d'applications robustes, capables de s'adapter à un environnement dynamique et imprévisible.
- permettre une personnalisation et un contrôle par les utilisateurs de ces applications, en prenant en compte leurs préférences générales et en leur donnant des moyens intuitifs d'agir sur le comportement de l'environnement.

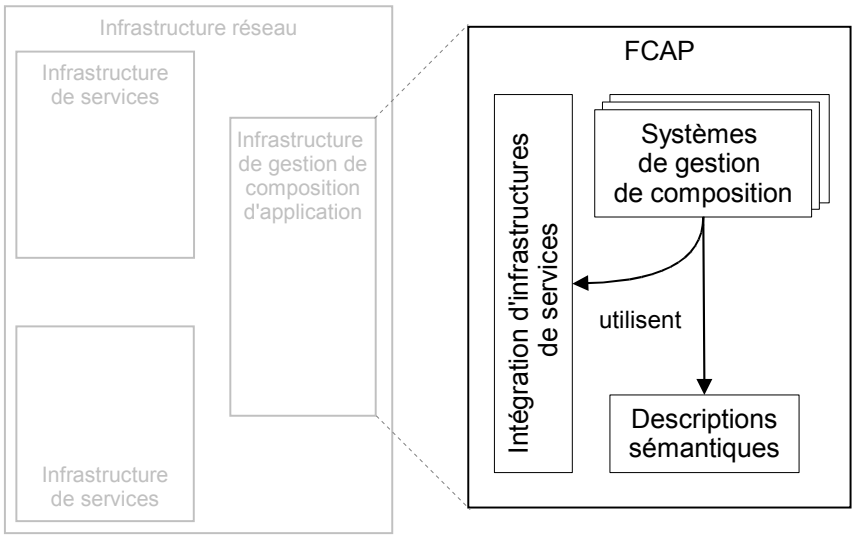

FIGURE 4.2 – Positionnement de FCAP dans l'architecture de référence

La figure 4.2 situe FCAP dans l'architecture de référence et introduit les différents éléments présentés dans les chapitres suivants :

Les chapitres suivants effectuent une présentation progressive de FCAP et de son intégration dans l'architecture de référence.

- Les *concepts fondamentaux* de FCAP sont présentés dans le chapitre 5. Ce chapitre présente aussi les éléments de base de FCAP. En particulier, la section 5.2 présente l'utilisation des descriptions sémantiques de fonctionnalités et la section 5.3 présente le support pour l'intégration avec des infrastructures de services existantes.
- Le *modèle de gestion de composition flexible* selon lequel fonctionne FCAP est détaillé dans le chapitre 6. Ce modèle comprend deux types de mécanismes, qui permettent de réaliser une composition adaptée à partit de descriptions sémantiques des fonctionnalités et de l'application.
- L'*architecture générique des systèmes de gestion de composition* est détaillée dans le chapitre 7. Les systèmes de composition sont l'élément principal de FCAP, et ils mettent en œuvre le modèle de gestion de composition flexible.

Deuxième partie

Proposition d'un intergiciel pour la composition flexible d'application

Chapitre 5

Principes de l'intergiciel FCAP pour la composition flexible d'applications

L'analyse de l'état de l'art des chapitres 2 et 3 a révélé que les différentes approches envisageables pour concevoir des applications attentives présentent des lacunes par rapport aux problématiques soulevées par ce type d'applications. Le chapitre 4 synthétise les atouts et les limitations des différentes solutions existantes et préconise l'élaboration d'une infrastructure d'adaptation pour les applications attentives. Le rôle d'une telle infrastructure est de fournir des mécanismes génériques et mutualisés pour permettre la conception d'applications attentives capables de s'adapter aux conditions rencontrées dans un environnement particulier et pour un utilisateur particulier.

Ce chapitre présente FCAP, un intergiciel pour la composition flexible d'applications fondée sur la notion d'application composite flexible. Dans cette proposition une application est conçue comme un assemblage dynamique des fonctionnalités présentes dans un environnement particulier, et cet assemblage est mis à jour en fonction des situations rencontrées. Pour permettre ce mode de fonctionnement, FCAP repose sur une approche de composition fondée sur la manipulation de descriptions des fonctionnalités et des applications. À partir de ces descriptions, FCAP élabore dynamiquement la description d'un assemblage de fonctionnalités appropriées pour la situation courante, en intégrant des informations issues de sources diverses. FCAP propose ainsi une approche générique pour garantir un couplage faible des fonctionnalités, une robustesse face à la dynamicité de l'environnement et une capacité à évoluer en fonction des besoins des utilisateurs.

La section 5.1 précise la notion d'application composite flexible et le modèle conceptuel sur lequel repose FCAP. La section 5.2 présente l'approche de composition par manipulation de descriptions, qui garantit les propriétés des applications composites flexibles. La section 5.3 le support fourni par FCAP pour l'intégration dans un environnement attentif.

5.1 Notion d'application composite flexible

5.1.1 Applications composites flexibles : modèle conceptuel

Cette section présente le modèle conceptuel dans lequel s'inscrivent les applications composites flexibles. Ce modèle se caractérise par l'absence de la notion d'application en tant qu'ensemble de fonctionnalités assemblées *a priori* pour répondre à un besoin bien défini. A l'inverse, les applications sont des entités abstraites dont la matérialisation dépend de la situation.

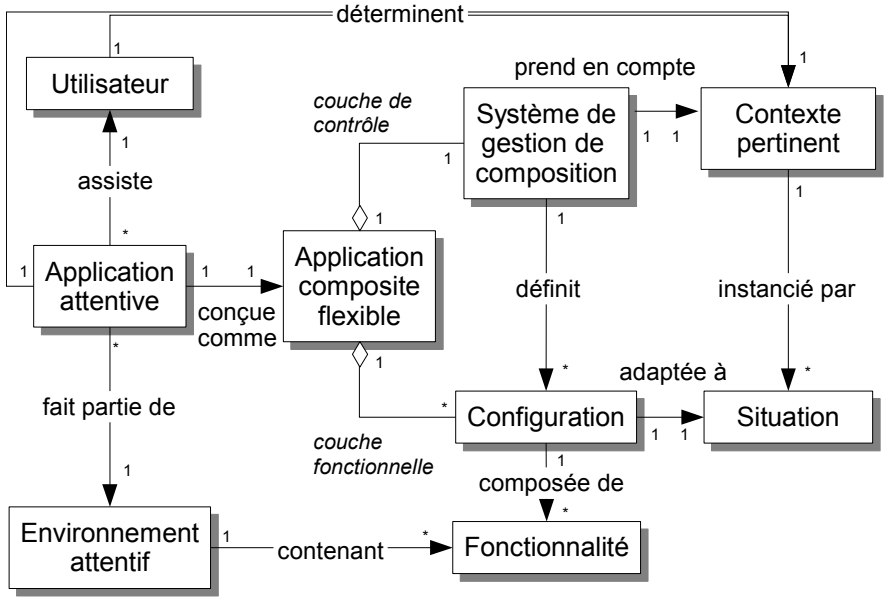

FIGURE 5.1 – Modèle conceptuel des applications composites flexibles

La figure 5.1 représente graphiquement le modèle conceptuel. Parmi les concepts qui composent ce modèle, certains ont déjà été définis :

- Les concepts d'*utilisateur* et de *fonctionnalité* sont assez généraux et ont été précisés dans le chapitre 4.
- Les concepts de *contexte pertinent* et de *situation* sont employés dans de nombreux travaux. On adopte ici la définition très générale suivante (Dey, 2000) : le contexte pertinent est l'ensemble des informations qui caractérisent les entités considérées comme pertinentes pour l'interaction entre l'utilisateur et l'application. Le contexte pertinent est ainsi défini pour un utilisateur et une application attentive donnée, et il peut évoluer au cours du temps. Une situation est une instance particulière d'un contexte, dans laquelle les différents éléments qui composent le contexte prennent une valeur particulière (Rey, 2005).
- Les concepts d'*application attentive* et d'*environnement attentif* ont été définis précédemment (section 2.1.1).

Trois nouveaux concepts sont introduits par ce modèle :

Définition 4 (Application composite flexible) *Une application composite flexible est un système logiciel constitué de composants* faiblement couplés*, dont la nature et l'organisation sont déterminées* dynamiquement *en fonction de la situation dans la-*

quelle l'application s'exécute. La constitution d'une application composite flexible dépend en particulier des composants disponibles dans un environnement donné, de la capacité de ces composants à interagir et de la situation du contexte pertinent pour l'utilisation de l'application. Une application composite flexible peut ainsi réaliser une application attentive adaptée à un utilisateur particulier, dans une situation particulière du contexte.

Définition 5 (Configuration) *La configuration d'une application composite flexible est définie par un ensemble de fonctionnalités et de connexions entre ces fonctionnalités. La pertinence d'une configuration est définie pour une situation donnée, et les configurations prises par une application composite flexible varient en fonction des situations rencontrées.*

Définition 6 (Système de gestion de composition) *Un système de gestion de composition réalise le lien entre les besoins de l'utilisateur, les fonctionnalités effectivement disponibles dans un environnement particulier et les différentes informations de contexte à prendre en compte pour l'adaptation de l'application. Chaque application composite flexible possède son système de gestion de composition,*

5.1.2 Architecture générique d'une application composite flexible

La figure 5.1 fait apparaître qu'une application composite flexible est réalisée par l'agrégation d'un système de gestion de composition et d'un ensemble de configurations. Cette architecture se rapproche de l'architecture générique pour les applications auto-adaptatives proposée par Oreizi et al. ((Oreizy *et al.*, 1999), voir section 3.2.2), mais prend en compte le caractère ouvert et évolutif d'un environnement attentif. On peut ainsi considérer qu'une application composite flexible s'organise en 2 couches :

La couche fonctionnelle est constituée d'un ensemble de fonctionnalités qui sont impliquées dans l'application. Ces fonctionnalités sont divisées en deux catégories. Les *fonctionnalités spécifiques* sont des fonctionnalités développées uniquement dans le cadre d'une application précise. Elles sont en général fortement liées entre elles et réalisent une tâche particulière à l'application. Les *fonctionnalités réutilisables* sont des fonctionnalités qui peuvent être partagées entre plusieurs applications. Elles sont généralement présentes dans l'environnement indépendamment d'une application particulière, afin d'être utilisées de manière opportuniste. Dans un environnement attentif, les fonctionnalités d'interface homme-machine sont généralement des fonctionnalités réutilisables.

La couche de contrôle est constituée par un ensemble d'agents logiciels formant le système de gestion de composition. Ces agents sont donc chargés de réaliser l'adaptation de l'application en fonction de la situation, et chaque agent est responsable d'un aspect particulier de l'adaptation de l'application.

L'architecture précise d'une application composite flexible peut évoluer au cours du fonctionnement. Aucune de ces deux couches n'est en effet complètement définie lors de la conception de l'application, car chacune dépend en partie des conditions rencontrées lors de l'exécution :

- Pour la couche fonctionnelle, seules les fonctionnalités spécifiques sont définies lors de la conception de l'application. Les fonctionnalités réutilisables dépendent de l'environnement et doivent être découvertes à l'exécution. Les interactions entre les fonctionnalités sont elles aussi définies à l'exécution.
- Pour la couche de contrôle, seule une spécification abstraite de l'application recherchée est définie à la conception. Cette spécification sert de base pour générer et organiser dynamiquement un système multi-agents chargé d'effectuer la gestion de composition.

Chacune de ces deux couches est supportée par une infrastructure distincte. La couche fonctionnelle est essentiellement supportée par une ou plusieurs infrastructures de services, qui permettent aux différentes fonctionnalités de rendre leurs services accessibles sur le réseau, de publier leurs descriptions, ainsi que d'accéder à des services proposés par d'autres fonctionnalités. L'intergiciel FCAP constitue une infrastructure pour la couche de contrôle.

5.2 Manipulation de descriptions pour la composition

Les informations manipulées par un système de gestion de composition prennent la forme de descriptions sémantiques. Comme on l'a vu dans le cas des services Web sémantiques (section 3.1.2), l'utilisation de descriptions sémantiques permet de faire face à l'hétérogénéité et à l'ouverture d'un environnement attentif en permettant un découplage fort entre les différentes parties impliquées. La figure 5.2 présente les différents types de descriptions manipulées par un système de gestion de composition :

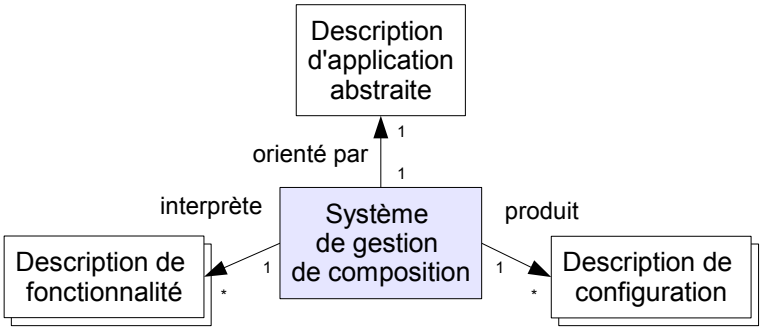

FIGURE 5.2 – Utilisation des descriptions sémantiques

La description d'application abstraite contient la spécification abstraite de l'application à réaliser. Elle définit ainsi l'objectif à atteindre et dirige le comportement du système de gestion de composition. Elle est fournie par les concepteurs de l'application.

Les descriptions de fonctionnalités contiennent des informations sur chacune des fonctionnalités présentes dans l'environnement. Elles sont fournies par les différents concepteurs de fonctionnalités.

Les descriptions de configurations définissent une configuration particulière d'une application attentive. Elles sont produites par le système de gestion de composition, en fonction des situations rencontrées.

FCAP fournit un support générique pour la manipulation de ces descriptions. Ainsi, la section 5.2.1 définit les modèles de descriptions sur lesquels sont basées les descriptions d'applications, de fonctionnalités et de configurations. La section 5.2.2 présente la notion d'espace de description, qui permet de manipuler des descriptions distribuées au travers du système et reliées entre elles. Enfin, la section 5.2.3 précise l'utilisation des descriptions pour la composition.

5.2.1 Modèles de description des applications, des fonctionnalités et des configurations

Afin de permettre aux systèmes de gestion de composition de manipuler les différentes descriptions, FCAP propose des modèles de description de haut-niveau pour les applications, les fonctionnalités et les configurations. Ces modèles définissent les concepts principaux, génériques et applicables à tout environnement attentif. Ces modèles peuvent ainsi être étendus pour décrire plus précisément les caractéristiques d'une application, d'une fonctionnalité, ou d'une configuration.

a) Modèle de description d'application abstraite

Le modèle de description d'application abstraite permet la spécification d'une application de manière abstraite, sans faire référence aux dispositifs et aux fonctionnalités qui peuvent être présents dans l'environnement. La figure 5.3 présente ce modèle de description. Il se compose de trois concepts :

app:AbsFunctionnality représente le concept de fonctionnalité d'une application abstraite (ou fonctionnalité abstraite). Une fonctionnalité abstraite représente une fonctionnalité dont les caractéristiques correspondent aux besoins exprimés par l'application abstraite, mais ne fait pas nécessairement référence à une fonctionnalité réellement fournie par un dispositif.

app:Interaction représente le concept d'interaction entre des fonctionnalités. Des fonctionnalités peuvent interagir pour échanger des informations, ou avoir une influence indirecte l'une sur l'autre. On considère ici qu'une interaction lie toujours deux fonctionnalités.

app:Application représente le concept d'application. Une application comporte généralement plusieurs fonctionnalités abstraites qui interagissent.

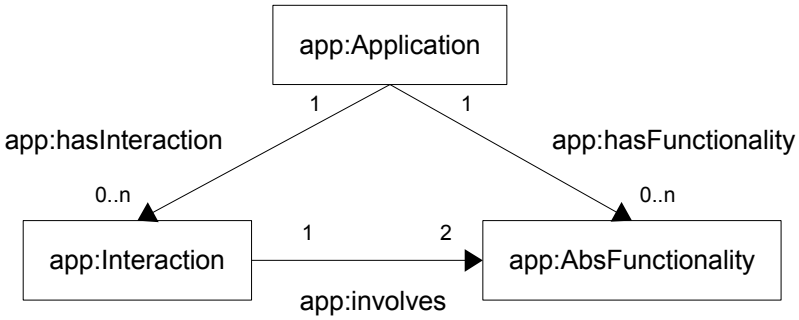

FIGURE 5.3 – Modèle de description d'application

Exemple La figure 5.4 présente une partie de la description d'une application correspondant au scénario de la section 2.1.2. Cette description définit une application irnotif:app, composée de deux fonctionnalités : irnotif:sender et irnotif:notifier. Elle définit de plus que ces deux fonctionnalités doivent interagir au travers d'une interaction irnotif:interaction (sans préciser la nature de cette interaction).

b) Modèle de description de fonctionnalité

Le modèle de description de fonctionnalité est un modèle de haut-niveau, qui s'applique à toute sorte de fonctionnalités présentes dans un environnement attentif. La

```
irnotif:app
    a     app:Application ;
    app:hasFunctionnality irnotif:sender ;
    app:hasFunctionnality irnotif:notifier ;
    app:hasInteraction irnotif:interaction1 .
irnotif:sender
    a     app:AbsFunctionnality .
irnotif:notifier
    a     app:AbsFunctionnality .
irnotif:interaction1
    a     app:Interaction ;
    app:involves irnotif:sender ;
    app:involves irnotif:notifier .
```

FIGURE 5.4 – Partie de la description de l'application

figure 5.5 représente le modèle de description de fonctionnalités. Cette ontologie présente deux classes (ou concepts) essentiels :

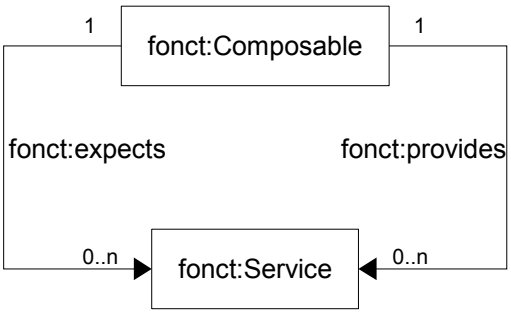

FIGURE 5.5 – Modèle de description de fonctionnalité

fonct:Composable représente le concept de fonctionnalité composable. Une fonctionnalité composable peut être composée avec d'autres fonctionnalités pour former une application.

fonct:Service représente un service grâce auquel il est possible d'invoquer une fonctionnalité. La propriété fonct:provides permet d'exprimer qu'une fonctionnalité fournit un service. La propriété fonct:expects permet d'exprimer qu'une fonctionnalité s'attend à utiliser un service fourni par une autre fonctionnalité.

Exemple Les figures suivantes présentent des descriptions de fonctionnalités du scénario présenté dans la section 2.1.2.

- La figure 5.6 présente une partie de la description de la fonctionnalité offerte par le cadre photo. Elle définit ainsi que la ressource nommée pframe:msg_displayer est une fonctionnalité composable et qu'elle fournit un service, nommé pframe:VisualNotification.
- La figure 5.7 présente une partie de la description de la fonctionnalité offerte par iRider. Elle définit ainsi que la ressource nommée irider:irider est une fonctionnalité composable et qu'elle attend un service, nommée irider:notif_service.

```
pframe:msg_displayer
    a      fonct:Composable ;
    fonct:provides pframe:display_service .
pframe:display_service
    a      fonct:Service .
```

FIGURE 5.6 – Partie de la description du cadre photo

```
irider:irider
    a      fonct:Composable ;
    fonct:expects irider:notif_service .
irider:notif_service
    a      fonct:Service .
```

FIGURE 5.7 – Partie de la description de iRider

c) Modèle de description de configuration

La modélisation des configurations s'appuie sur la modélisation des fonctionnalités et fournit des éléments pour décrire des interactions entre les fonctionnalités. La figure 5.8 représente le modèle de description des configurations. Ce modèle présente deux classes essentielles :

conf:Assembly représente le concept d'assemblage de plusieurs fonctionnalités. Un assemblage implique plusieurs fonctionnalités composables et met en place des connecteurs pour les faire interagir.

conf:Connector représente le concept de connecteur entre des services. Un connecteur permet de mettre en interaction un service fourni par une fonctionnalité composable et un service attendu par une autre fonctionnalité composable.

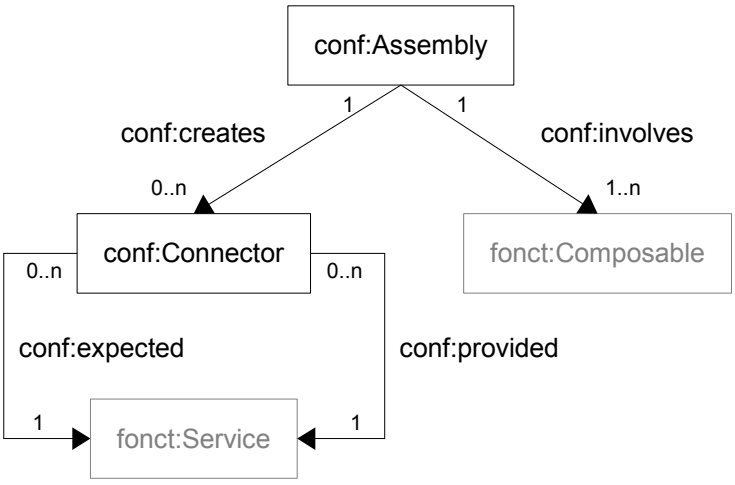

FIGURE 5.8 – Modèle de description des configurations

Exemple La figure 5.9 présente un assemblage issu du scénario présenté dans la section 2.1.2. Elle définit que la ressource nommée notif1:asm1 est un assemblage, qui

implique les ressources pframe:msg_displayer et irider:irider. Cet assemblage crée de plus un connecteur nommé notif1:connector1, qui connecte deux fonctionnalités composables pframe:msg_displayer et irider:irider en considérant pframe:display_service comme service fournit et irider:notif_service comme service attendu.

```
notif1:asm1
    a      conf:Assembly ;
    conf:involves pframe:msg_displayer ;
    conf:involves irider:irider ;
    conf:creates connect:connector .
notif1:connector1
    a      conf:Connector ;
    conf:connects pframe:msg_displayer ;
    conf:connects irider:irider ;
    conf:provided pframe:display_service ;
    conf:expected irider:notif_service .
```

FIGURE 5.9 – Partie de la description d'un assemblage de iRider et du cadre photo

5.2.2 Espace de descriptions

Dans un environnement attentif, les descriptions des différentes entités (fonctionnalités, applications ...) sont par nature issues de sources diverses, réparties dans l'environnement. FCAP propose un mode de gestion des descriptions sur le principe d'un espace partagé : des producteurs publient des informations et des consommateurs obtiennent, regroupent et analysent les informations disponibles selon leurs besoins. Cet *espace de descriptions* possède trois caractéristiques :

Persistance des informations : les informations publiées ne sont pas altérées par leur lecture. Elles restent disponibles jusqu'à ce que leur fournisseur les retire. Ces informations sont de plus accessibles sur le réseau, et identifiées de manière unique.

Distribution : chaque fournisseur est chargé de donner un accès direct aux informations qu'il publie. Chaque consommateur doit obtenir les informations utiles auprès des consommateurs et les stocker lui-même pour les analyser. Il n'existe pas d'espace de stockage centralisé dans lequel toutes les informations seraient disponibles.

Modélisation sémantique : les informations sont toutes exprimées sous la forme de triplets RDF (voir section 3.1.2). La sémantique des termes employés est définie par des ontologies OWL, qui constituent elles-mêmes un type particulier d'information présente dans l'espace de description.

a) Ressources, documents, et descriptions

L'espace de descriptions proposé par FCAP est constitué de trois types d'éléments : les ressources, les documents et les descriptions.

Une *ressource* représente une entité sur laquelle des informations peuvent être exprimées dans des descriptions. Une ressource est identifiée de manière unique par

une URI. Dans la suite, une ressource sera notée sous la forme $\underline{r}$ [1]. Dans certains cas, on pourra expliciter l'URI identifiant une ressource en écrivant $\underline{r}$=http:// example.org/ sample/ uri#fragment ou $\underline{r}$=namespace:fragment (lorsque le préfixe namespace: est défini comme http:// example.org/ sample/ uri#) .

Un *document* regroupe un ensemble d'informations exprimées en RDF, accessible à une adresse précise sur le réseau. Cette adresse est exprimée par une URL, qui sert aussi à identifier le document de manière unique. Dans la suite, un document est noté sous la forme $\underline{D}$, en utilisant un identifiant majuscule afin de le distinguer d'une ressource.

La notion de description est définie comme suit :

Définition 7 (Description) *Une description est un ensemble d'informations portant sur une ressource particulière. Une description est caractérisée par un* document support *et une* ressource *sur laquelle porte la description. Il peut exister plusieurs descriptions pour une même ressource, chaque description ayant un document différent pour support. Par ailleurs, un même document peut servir de support pour décrire plusieurs ressources.*

Dans la suite, la description d'une entité $\underline{r}$ dans un document $\underline{D}$ est désignée par un couple $r = (\underline{r}, \underline{D})$, où $\underline{r}$ est l'URI qui identifie la ressource de manière unique et $\underline{D}$ est l'URL à laquelle est accessible le document. $r_1 = (\underline{r}, \underline{D_1})$ et $r_2 = (\underline{r}, \underline{D_2})$ désignent donc deux descriptions distinctes d'une même ressource, basées sur des documents différents $\underline{D_1}$ et $\underline{D_2}$.

b) Hiérarchisation des documents

L'espace de descriptions est destiné à favoriser la modularité et la réutilisation des connaissances. Pour cela, les documents sont organisés de manière hiérarchique : un document peut englober d'autres documents et contenir ainsi l'ensemble des informations que ceux-ci contiennent. On dira qu'un document *importe* un autre document.

Pour chaque document $\underline{D}$, on note ainsi l'ensemble des documents qu'il importe $Import(\underline{D})$. Par exemple, pour un document $\underline{D}$ qui importe deux documents $\underline{D1}$ et $\underline{D2}$, on a $Import(\underline{D}) = \{\underline{D1}, \underline{D2}\}$. La relation d'import est transitive, et on note l'ensemble complet des documents importé $Import^*(\underline{D})$. Par exemple, $Import^*(\underline{D}) = \{\underline{D1}, \underline{D2}, \underline{D3}\}$, si $Import(\underline{D1}) = \{\underline{D3}\}$, $Import(\underline{D2}) = \emptyset$ et $Import(\underline{D3}) = \emptyset$.

1. On notera que la notation soulignée pour les URI se rapproche de l'apparence habituelle des hyperliens dans les documents HTML.

c) Exemples

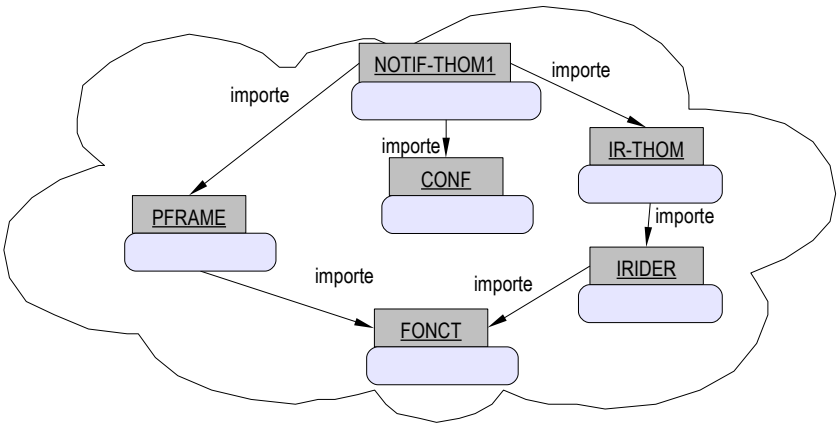

FIGURE 5.10 – Exemple d'espace de description

La figure 5.10 illustre un espace de description comportant quatre documents interdépendants, qui correspondent à un cas du scénario de la section 2.1.2. Les quatre documents sont les suivants :

PFRAME : document support de la description du cadre photo La figure 5.6 présente une partie de la description du cadre photo correspondante, exprimée à l'aide du modèle de description de fonctionnalités. PFRAME est le document contenant cette description. La description de la fonctionnalité du cadre photo est ainsi désignée par (pframe,PFRAME), avec pframe=pframe:msg_displayer, PFRAME=http://.../PictFram et $Import$(PFRAME)={FONCT}.

IRIDER : document support de la description du système de co-voiturage iRider La figure 5.7 présente une partie de la description du système de co-voiturage iRider, exprimée à l'aide du modèle de description de fonctionnalités. IRIDER est le document contenant cette description. La description de la fonctionnalité de iRider est ainsi désignée par (irider,IRIDER), avec irider=irider:irider, IRIDER=http://.../IRider.owl et $Import$(IRIDER)={FONCT}.

IR-THOM : document support d'une description de iRider utilisé par Thom IR-THOM est un document qui enrichit le document IRIDER, en indiquant que Thom est l'utilisateur de iRider. Dans ce document, la description de iRider est désignée par (irider, IR-THOM), où irider=irider:irider et IR-THOM=http://.../IRiderThom.owl. Noter que cette description diffère de la précédente car l'URL du document change, tandis que l'URI de la ressource décrite reste la même. On a de plus $Import$(IR-THOM)={IRIDER}.

NOTIF-THOM1 : description de la composition de iRider et du cadre photo La figure 5.9 présente un assemblage des fonctionnalités précédentes. Cet assemblage est basé sur les documents présentés précédemment. Ainsi, on a : $Import$(NOTIF-THOM1)={ CONF, PFRAME, IR-THOM } et $Import^*$(NOTIF-THOM1)={ CONF, PFRAME, IR-THOM, IRIDER, SERVICE }. Dans ce document, les descriptions du cadre photo et de iRider, ainsi que la description de Thom sont respectivement (pframe, NOTIF-THOM1), (irider, NOTIF-THOM1), (thom, NOTIF-THOM1), où NOTIF-THOM1=http://.../NotifThom1.owl et thom=chezThom:Thom.

d) Opérations dans l'espace de descriptions

L'espace de descriptions n'est pas figé : de nouvelles informations peuvent y être ajoutées, d'autres informations peuvent être retirées. Les informations contenues dans l'espace de descriptions reflètent ainsi l'état de l'environnement attentif, et son évolution.

On considère trois types d'opérations possibles dans l'espace de descriptions :

L'ajout de document permet d'ajouter de nouvelles informations à l'espace de descriptions. Il est en particulier possible d'ajouter des informations à un document déjà existant, en ajoutant un nouveau document qui l'importe et ajoute de nouvelles informations. Il n'est en revanche pas possible de modifier directement un document existant.

Le remplacement de document permet de supprimer des informations de l'espace de description. Dans le cas où une information contenue dans un document n'est plus valide, il n'est pas possible de retirer simplement ce document. En effet, il peut être importé par d'autres documents ou avoir été lu par des consommateurs. La solution consiste à conserver le document en le remplaçant par une nouvelle version.

Le déplacement de document permet de déplacer un document dans la hiérarchie des documents. Cette opération consiste à modifier l'ensemble des documents importés, sans modifier le contenu du document lui-même. Elle peut notamment être utilisée pour propager le remplacement d'un document importé par une nouvelle version.

Ces opérations permettent de transformer et de faire évoluer l'espace de descriptions.

e) Interprétation de descriptions

Une transformation fondamentale de l'espace de descriptions est la publication d'informations obtenues par un raisonnement qui interprète les informations fournies par une ou plusieurs descriptions. Ces informations sont ajoutées à l'espace de descriptions en créant un nouveau document qui importe les documents supports des descriptions considérées. Un tel document est appelé une *interprétation de descriptions*.

À partir des descriptions sur lesquelles portent l'interprétation, il est alors possible de définir de nouvelles descriptions ayant pour support ce nouveau document. Par exemple, sur la base des descriptions $a = (\underline{a}, \underline{A})$ et $b = (\underline{b}, \underline{B})$, une interprétation des descriptions a et b est un document $\underline{C}$ tel que $Import(\underline{C}) = \{\underline{A}, \underline{B}\}$. Les nouvelles descriptions sont $(\underline{a}, \underline{C})$ et $(\underline{b}, \underline{C})$. On dira que $(\underline{a}, \underline{C})$ est une description *enrichie* de $(\underline{a}, \underline{A})$.

5.2.3 Composition basée sur les descriptions

Dans FCAP, la gestion de composition repose sur la manipulation des descriptions des fonctionnalités, des applications et des configurations. Un système de gestion de composition s'appuie sur l'espace de description pour gérer la distribution et la mise à jour des informations contenues dans les descriptions. Cette section introduit comment la gestion de composition s'opère dans le cadre de l'espace de description.

a) Interprétation de fonctionnalités

Dans la section précédente, on a présenté la notion d'interprétation de descriptions, qui consiste à ajouter à l'espace de description de nouveaux documents dont le contenu est dérivé de documents existants, notamment grâce à des mécanismes de raisonnement.

Dans le cas de la gestion de composition, un cas particulier d'interprétation de descriptions est l'*interprétation de fonctionnalités.*

Définition 8 (Interprétation de fonctionnalités) *Une interprétation de fonctionnalités est une interprétation de descriptions portant sur un ensemble de couples constitués d'une fonctionnalité abstraite (définie pour une application) et d'une fonctionnalité réelle (existant dans l'environnement). Une interprétation de fonctionnalités décrit certaines relations existant entre une fonctionnalité abstraite et une fonctionnalité réelle pour chaque couple de fonctionnalités.*

Dans le cas d'une interprétation de fonctionnalités portant sur un seul couple de fonctionnalités, on peut considérer :

- Une description de fonctionnalité abstraite $a = (\underline{a}, \underline{A})$. Par définition, $\underline{A}$ contient le triplet ($\underline{a}$ rdf:type app:Functionnality).
- Une description de fonctionnalité réelle $f = (\underline{f}, \underline{F})$. Par définition, $\underline{F}$ contient le triplet ($\underline{f}$ rdf:type fonct:Composable).
  Une interprétation des fonctionnalités a et f est un document $\underline{I}$ tel que $\{\underline{A}, \underline{F}\} \subset Import(\underline{I})$.

Pour un couple donné de descriptions, il peut exister de nombreuses interprétations différentes. On distingue ainsi différents types d'interprétations, et on note $\underline{\tau}[(f, a)]$ l'interprétation de type τ portant sur les descriptions a et f. De même, pour une interprétation portant sur plusieurs couples de fonctionnalités, on note $\underline{\tau}[(f_1,a_1), (f_2,a_2), \ldots, (f_n, a_n)]$. Afin de simplifier les notations dans la suite, on écrira les formules dans le cas d'une interprétation portant sur un seul couple de fonctionnalités.

Exemples d'interprétation de fonctionnalités La figure 5.11 représente trois interprétations obtenues à partir des descriptions de fonctionnalités abstraites $irn = (\underline{irnotif}, \underline{NOTIF})$, $not = (\underline{notifier}, \underline{NOTIF})$ et des descriptions de fonctionnalités réelles $ir = (\underline{irider}, \underline{IRIDER})$, $pf = (\underline{pframe}, \underline{PFRAME})$:

- $\underline{\text{REALIZE}}[(pf, not)]$ est une interprétation qui décrit que la fonctionnalité réelle *pframe* réalise la fonctionnalité abstraite *notifier*. En général, les interprétations donnent des informations plus complexes sur la relation entre une fonctionnalité abstraite et une fonctionnalité réelle.
- $\underline{\text{CONTEXT}}[(ir, irn)]$ est une interprétation qui décrit que *thom* est l'utilisateur de la fonctionnalité *irider*. Cette information a pu être déduite des informations contenues dans la description abstraite de l'application.
- $\underline{\text{CONNECT}}[(pf, not), (ir, irn)]$ est une interprétation qui décrit la connexion entre les fonctionnalités *pframe* et *irider*.

Utilité d'une interprétation de fonctionnalités Si l'on considère les différents couples de fonctionnalités abstraites/réelles de l'environnement, et différents types d'interprétations possibles, de nombreuses interprétations différentes peuvent être construites et ajoutées à l'espace de description. Dans le cadre de la composition, toutes les interprétations n'ont pas le même intérêt : certaines interprétations permettent de construire une configuration adaptée, d'autres ne sont pas adaptées. On définit ainsi une notion d'utilité d'une interprétation de fonctionnalités

Définition 9 (Utilité d'une interprétation de fonctionnalités) *L'utilité d'une interprétation de fonctionnalités est une mesure qui indique à quel point cette interprétation est intéressante pour obtenir une configuration qui correspond à la description abstraite d'une application et qui est adaptée à la situation courante de l'environnement.*

L'utilité d'une interprétation de fonctionnalités mesure en quelque sorte la proximité entre la description d'une fonctionnalité abstraite et la description d'une fonctionnalité réelle. Dans la suite, l'utilité d'une interprétation $\underline{\tau}[(f, a)]$ est notée $\mu(\underline{\tau}[(f, a)])$.

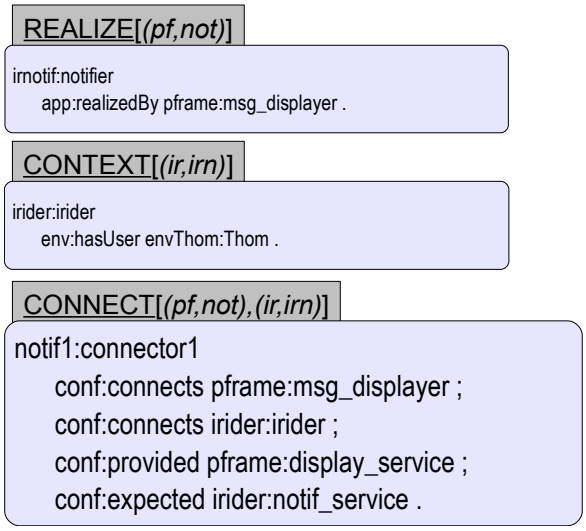

FIGURE 5.11 – Exemple d'interprétations de fonctionnalités

Lorsqu'une interprétation est intéressante pour obtenir une configuration adaptée, on dit que son utilité est positive et on note $\mu(\underline{\tau}[(f,a)]) > 0$. Dans le cas contraire, on dit que son utilité est négative et on note $\mu(\underline{\tau}[(f,a)]) < 0$.

Exemples d'utilité d'interprétations La figure 5.12 présente plusieurs interprétations avec une indication de leurs utilités respectives. On ne précise pas comment est mesurée l'utilité, mais on indique seulement si l'utilité est positive (+) ou négative (-). Pour l'interprétation de type *realize*, l'utilité de l'interprétation est positive si elle permet d'affirmer que la fonctionnalité réelle réalise la fonctionnalité abstraite, et négative dans le cas contraire. Trois cas sont présentés :

- L'interprétation $\underline{\text{REALIZE}}[(pf, not)]$ ajoute des informations qui permettent d'affirmer que *pframe* réalise *notifier*, et son utilité est positive.
- L'interprétation $\underline{\text{REALIZE}}[(ir, irn)]$ n'ajoute pas d'information, mais son utilité est positive, car la description de l'application indique déjà que *irider* réalise *irnotif*.
- L'interprétation $\underline{\text{REALIZE}}[(ir, not)]$ n'ajoute pas d'information, et son utilité est négative, car il n'est pas possible d'affirmer que *irider* réalise *notifier*.

b) Validité d'un ensemble de documents

Dans le cadre de la gestion de composition, un espace de description contient une description d'application abstraite (et les descriptions de fonctionnalités abstraites), des descriptions de fonctionnalités réelles et des interprétations de fonctionnalités. Les diverses interprétations de fonctionnalités associent de manière différentes les fonctionnalités abstraites et réelles, et ont des utilités différentes. À l'intérieur de l'espace de description, on peut considérer des sous-ensembles de documents possédant des propriétés particulières.

Définition 10 (Ensemble de documents valide) *Un ensemble de documents valide D est un sous-ensemble de l'espace de description tel que :*

- *D contient l'ensemble les documents décrivant l'application (et en particulier les fonctionnalités abstraites).*

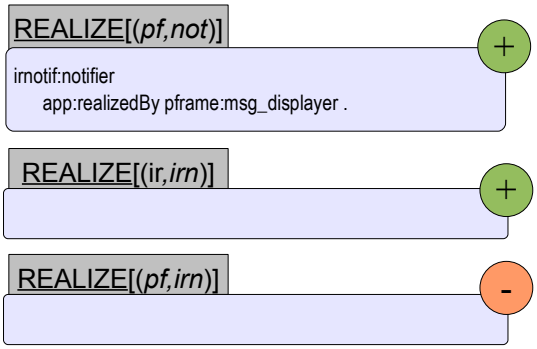

FIGURE 5.12 – Exemple d'utilité attribuée à des interprétations de fonctionnalités

- *D contient un ensemble D_f de documents décrivant des fonctionnalités réelles.*
- *Il existe une unique fonction γ qui associe une description de fonctionnalité réelle à chacune des descriptions de fonctionnalité abstraite, telle que toutes les interprétations appartenant à D associent toujours des fonctionnalités abstraites à une même image par γ, c'est-à-dire $\exists\gamma$ tel que $\underline{\tau}[(f,a)] \in D \Rightarrow f = \gamma(a)$.*
- *Toutes les interprétations appartenant à D ont une utilité positive, c'est-à-dire $\underline{\tau}[(f,a)] \in D \Rightarrow \mu(\underline{\tau}[(f,a)]) > 0$.*

Un ensemble de documents valide est dit *maximal* si toutes les interprétations possibles sont présentes dans D, c'est-à-dire s'il existe une fonction γ telle que pour chaque type d'interprétations τ et pour chaque fonctionnalité abstraite a, l'interprétation de fonctionnalité $\underline{\tau}[(\gamma(a),a)]$ appartient à l'ensemble de documents (cette interprétation est de plus positive, par définition d'un ensemble de document valide).

Exemple En se basant sur les exemples de la section précédente, on a par exemple
- { NOTIF, PFRAME, IRIDER, REALIZE$[(pf,not)]$, REALIZE$[(ir,irn)]$ } est un ensemble de documents *valide*.
- { NOTIF, PFRAME, IRIDER, REALIZE$[(pf,not)]$, REALIZE$[(pf,irn)]$ } *n'est pas* un ensemble de documents valide, car $\mu(\underline{REALIZE}[(pf,irn)]) < 0$.
- { NOTIF, PFRAME, IRIDER, REALIZE$[(pf,not)]$, REALIZE$[(ir,irn)]$, CONTEXT$[(pf,not)]$, CONTEXT$[(ir,irn)]$, CONNECT$[(pf,not),(ir,irn)]$ } est un ensemble de document *valide* et *maximal*.

Illustration graphique Il est utile de représenter l'espace de description graphiquement. On représente les documents, avec l'étiquette associée, ainsi que les relations d'importation entre les documents. Pour clarifier l'illustration graphique lorsque plusieurs descriptions ont le même document support, il est possible de faire apparaître des documents virtuels, qui ne contiennent aucune information, mais apparaissent comme document support de chacune de ces descriptions.

La figure 5.13 représente graphiquement l'ensemble des documents et des interprétations considérées dans les exemples précédents (sans faire apparaître le contenu des documents).

Notion de description contextualisée d'une fonctionnalité Au sein d'un ensemble de descriptions valide, les interprétations font apparaître de nouvelles informations sur les fonctionnalités réelles. Chaque fonctionnalité réelle n'est ainsi pas

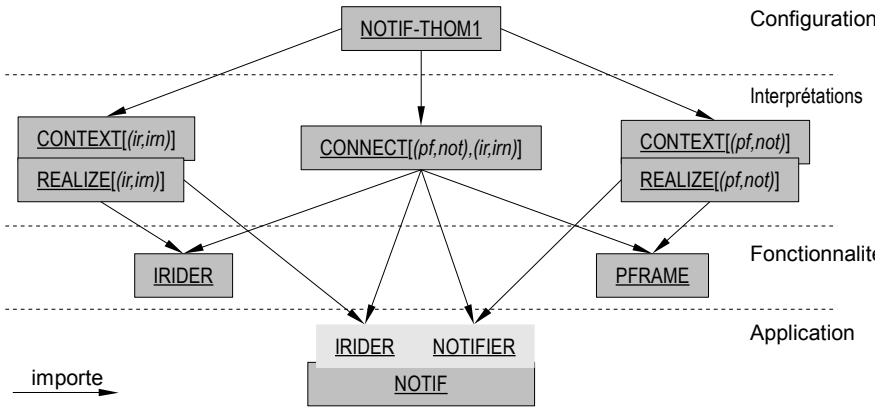

FIGURE 5.13 – Exemple d'ensemble valide dans un espace de description

uniquement décrite par sa description originale, fournie par le concepteur, mais aussi par l'ensemble des informations ajoutées par ces interprétations. On appelle *description contextualisée*, la description d'une fonctionnalité réelle ayant pour support un document qui importe l'ensemble des interprétations portant sur cette fonctionnalité.

Par exemple, dans le cas précédent, la description contextualisée du cadre photo est $pf' = (\underline{pframe}, \underline{PFRAME'})$, avec

$$Import(\underline{PFRAME'}) = \{\underline{REALIZE}(pf, not), \underline{CONTEXT}(pf, not), \underline{CONNECT}[(pf, not), (ir, irn)]\}$$

Il faut noter que, en raison de la transitivité de l'importation, le document PFRAME' importe aussi le document original PFRAME, et contient donc toutes les informations décrivant pframe.

Description de configuration issue d'un ensemble de descriptions valide maximal Dans un ensemble de descriptions valide et maximal, chaque description de fonctionnalité abstraite est associée de manière unique à une fonctionnalité réelle. De plus, cet ensemble de descriptions contient toutes les interprétations qui décrivent la relation entre une fonctionnalité abstraite et une fonctionnalité réelle, et toutes ces interprétations sont positives.

Sur la base de ces interprétations, il est donc possible de construire les descriptions contextualisées des fonctionnalités réelles. Pour chaque fonctionnalité abstraite a et pour chaque fonctionnalité réelle $f = \gamma(a)$, la description contextualisée de f est donnée par $f' = (\underline{f}, \underline{F'})$, où $\underline{F'}$ est un document qui importe toutes les interprétations portant sur $\underline{f}$: $Import(\underline{F'}) = \cup\underline{\tau}[\ldots, (f, a), \ldots]$.

Une configuration adaptée à la situation est obtenue sur la forme d'une description $c = (\underline{c}, \underline{C})$ telle que :

- $\underline{C}$ importe l'ensemble des descriptions contextualisées : $Import(\underline{C}) = \cup\underline{F'}$
- $\underline{C}$ contient tous les triplets ($\underline{c}$ conf:involves $\underline{f}$)

c) Adaptation de la composition

Comme on l'a déjà indiqué, l'espace de description reflète l'environnement et son évolution : il évolue donc constamment, en fonction de l'évolution de l'environnement. Plusieurs cas d'évolution influencent directement la composition :

Ajout/retrait de descriptions de fonctionnalités réelles. Lorsqu'une fonctionnalité apparaît/disparaît dans l'environnement, sa description est ajoutée/retirée de l'espace de description.

Modification d'une interprétation Certaines interprétations dépendent d'informations extérieures aux descriptions des fonctionnalités abstraites/réelles. C'est en particulier le cas pour les interprétations relatives au contexte de l'environnement. Lorsque l'environnement évolue, ces interprétations doivent être modifiées. Le document décrivant l'interprétation est alors retiré de l'espace de description, et remplacé par une nouvelle interprétation mise à jour.

Modification de la description de l'application Pour une application active sur le long terme, il est possible que les besoins de l'utilisateur évoluent au cours du fonctionnement de l'application. La description de l'application peut ainsi subir des modifications au cours du fonctionnement.

Chaque modification de l'espace de description remet potentiellement en cause les ensembles de description valides précédemment obtenus, et les configurations adaptées associées. Un système de gestion de composition doit donc être capable de s'adapter continuellement aux modifications de l'espace de description. Les mécanismes permettant une adaptation dynamique de la composition dans FCAP sont décrits plus précisément dans le chapitre 7.

5.3 Support pour la manipulation de descriptions

Comme décrit dans la section précédente, la gestion de composition repose sur une manipulation de descriptions de plusieurs types d'entités. Pour chaque application, un système de gestion de composition effectue ces manipulations de descriptions pour déterminer une configuration adaptée à la situation courante.

Afin de faciliter la conception d'application, FCAP fournit à l'ensemble des systèmes de gestion de composition un support générique pour la manipulation de descriptions. En particulier, FCAP prend en charge l'intégration des mécanismes de manipulation de description avec l'infrastructure classique d'un environnement attentif. Cette approche permet de masquer certaines particularités d'un environnement attentif aux systèmes de gestion de composition, qui considère uniquement un modèle plus abstrait de l'environnement.

Le support pour la manipulation de description se compose de trois aspects :

La gestion de description FCAP fournit un support pour gérer un espace de description dans un environnement distribué et dynamique. Elle permet ainsi de manipuler des descriptions qui sont disséminées dans le réseau et qui sont modifiées au cours du fonctionnement. Elle est décrite dans la section 5.3.1.

L'acquisition de descriptions FCAP fournit un support pour acquérir des descriptions portant sur les entités présentes dans l'environnement. Elle s'intéresse en particulier à l'acquisition de descriptions des fonctionnalités fournies par les dispositifs de l'environnement, et à l'acquisition d'informations sur le contexte de l'environnement. Elle est décrite dans la section 5.3.2.

La mise en place de configurations FCAP fournit un support pour mettre en place une configuration réalisant une application à un moment donnée. Elle fournit en particulier un support pour le contrôle des interactions entre des fonctionnalités hétérogènes. Elle est décrite dans la section 5.3.3.

5.3.1 Gestion de descriptions

Le système de gestion de descriptions distribuées fournit aux différents éléments de la plate-forme des bases de connaissances dans lesquelles ils peuvent lire et publier des descriptions. Ils peuvent notamment partager des connaissances en publiant de nouvelles descriptions et enrichir des descriptions existantes grâce à un mécanisme de références.

Le système de gestion de descriptions distribuées est utilisé par tous les autres éléments de la plate-forme. Son rôle est de faciliter l'accès et la production de documents de description, qui sont distribués dans un environnement attentif. Ces documents de description peuvent être de plusieurs natures : description de fonctionnalités, description d'informations sur le contexte, description d'applications, ontologies d'un domaine particulier. Les documents de description jouent un rôle fondamental dans la gestion d'applications composites flexibles, car ils véhiculent l'ensemble des informations interprétées et produites par un système de gestion de composition. Le système de gestion de descriptions distribuées permet de manipuler et de partager simplement ces documents. Dans la suite, le terme 3DM (*Documents for Distributed Description Management*) désigne le système de gestion de descriptions distribuées.

a) Architecture du système de gestion de descriptions distribuées

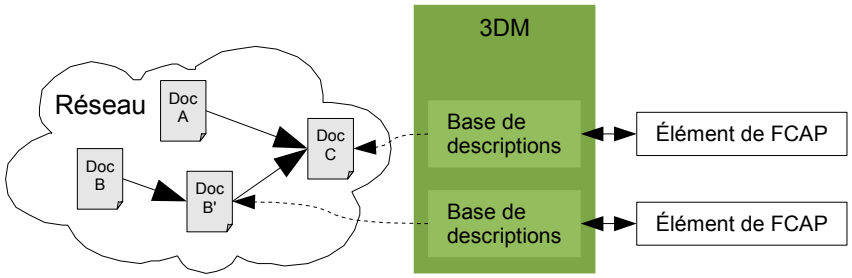

FIGURE 5.14 – Architecture du système de gestion de descriptions distribuées

La figure 5.14 présente l'architecture du 3DM. Il est constitué d'un ensemble de *bases de descriptions*. Chaque base de descriptions est capable de lire des descriptions accessibles sur le réseau et de publier des descriptions. Le système de gestion de descriptions distribuées est ainsi un système pair à pair dans lequel les pairs interagissent indirectement par le biais des descriptions publiées. Chaque base de descriptions est associée à un seul élément de FCAP (système de gestion de composition ou service utilitaire), qui l'utilise pour obtenir et partager ses connaissances.

La figure 5.15 représente l'architecture interne d'une base de descriptions. Chaque base de description est constituée des éléments suivants :

- La **base de connaissances** interne stocke les connaissances contenues dans les descriptions.
- Le **chargeur de documents** obtient les documents sur le réseau et les charge dans la base de connaissance.
- Le **publieur de documents** publie certains documents sur le réseau. Les documents sont ensuite accessibles à une adresse spécifique à la base de description considérée.
- L'**interface de gestion des descriptions** permet aux autres éléments de la plate-forme d'utiliser la base de descriptions pour accéder aux descriptions qui les concernent.

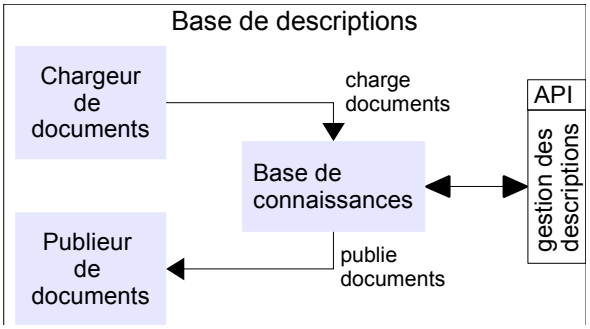

FIGURE 5.15 – Architecture d'une base de descriptions

L'annexe A précise l'API d'une base de descriptions. Dans la suite, ces opérations sont utilisées dans différents algorithmes qui utilisent une base de descriptions.

5.3.2 Acquisition de descriptions

Afin de construire une description de configuration adaptée, l'espace de description doit tout d'abord contenir les descriptions des fonctionnalités présentes dans l'environnement. FCAP assure un support pour faciliter l'acquisition de ces descriptions en exploitant l'infrastructure classique d'un environnement attentif. Cette section décrit les deux aspects de l'acquisition de description. La découverte de descriptions permet l'obtention de descriptions pour les fonctionnalités présentes dans l'environnement, en s'appuyant sur les mécanismes de découverte de services classiques. L'obtention d'information de contexte permet d'obtenir des informations sur les éléments de contexte pertinents de l'environnement, en s'appuyant sur des infrastructures de gestion de contexte existantes.

a) Découverte de descriptions de fonctionnalités

Le découvreur de descriptions de fonctionnalités permet de découvrir les fonctionnalités présentes sur le réseau et d'en obtenir une description sémantique. Les descriptions sémantiques de fonctionnalités se fondent sur une ontologie générique qui définit les concepts de haut niveau, communs à tout type de fonctionnalité (section 5.2.1). Afin de prendre en compte les spécificités de chaque fonctionnalité, la description sémantique comporte des éléments de description spécifiques, qui permettent par exemple de définir un protocole de communication particulier ou un critère de qualité de service pertinent. Ces éléments ne peuvent être interprétés directement par la plate-forme, mais ils sont pris en compte par des composants additionnels dédiés à leur interprétation.

Dans le cadre d'une architecture orientée service, les services sont découverts puis utilisés dynamiquement. Pour cela, chaque infrastructure de services repose sur un mécanisme de découverte particulier, mais aussi sur un langage de description spécifique. Le découvreur de descriptions de fonctionnalités (FDD) a donc deux rôles :

- Intégrer les différents mécanismes de découverte de services proposés par les différentes infrastructures de services. Ceci permet aux systèmes de gestion de composition d'avoir accès aux différents types de services indépendamment des infrastructures de services utilisées.
- Harmoniser les descriptions sémantiques de services pour permettre leur manipulation par les systèmes de gestion de composition.

Pour chaque service découvert sur le réseau, le FDD construit une description sémantique de la fonctionnalité qui le fourni, exprimée à l'aide du langage RDF et de l'ontologie décrite dans la section 5.2.1, figure 5.5. La plate-forme manipule cette description indépendamment de l'infrastructure de services et du langage de description employé pour la publication. Le système de découverte de description agit ainsi comme un médiateur entre les différentes infrastructures et la plate-forme.

Le FDD est caractérisé par les deux aspects suivants :

Découverte passive Le FDD privilégie un mode de découverte passive : un système de gestion de composition s'enregistre auprès du FDD pour recevoir de manière asynchrone les descriptions des services trouvés dans l'environnement. Ce mode de fonctionnement garantit la mise à jour en fonction des apparitions/disparitions de services dans l'environnement. Il est différent de celui de nombreux systèmes, dans lesquels un client doit effectuer une recherche active des services disponibles lors de son démarrage. Bien que la découverte active soit plus simple et plus intuitive, elle n'est pas suffisante pour les environnements dynamiques.

Absence de sélection À la différence de la plupart des mécanismes de découverte de services (section 3.1.1), le FDD n'effectue pas de sélection parmi les descriptions disponibles. On considère en effet que la complexité des environnements ne permet pas de définir un langage suffisamment expressif pour décrire précisément la fonctionnalité recherchée afin de permettre au FDD de sélectionner uniquement la plus appropriée. Le FDD n'offre ainsi aucune garantie que les descriptions qu'il fournit peuvent être utilisées directement par le demandeur. En revanche, pour des questions d'efficacité, il est possible de fournir un *filtre de découverte*, sur la forme d'une description de fonctionnalité recherchée : dans ce cas, le FDD exclut les descriptions dont il peut déterminer qu'elles sont incompatibles avec le filtre.

Architecture du découvreur de service L'architecture du découvreur de service est dirigée par la nécessité de présenter au système de gestion de composition une interface unifiée pour la découverte de services. Ainsi, le découvreur de services est constitué de deux types d'éléments :

Des médiateurs de découverte réalisent une médiation entre le modèle de découverte du FDD et les différents modèles des infrastructures de services présentes dans l'environnement.

L'aggrégateur de découverte est l'interface grâce à laquelle les éléments de la plate-forme utilisent le FDD. Elle définit les opérations qui permettent d'enregistrer une inscription pour recevoir des notifications de découverte et de disparition de descriptions.

Déroulement de la découverte de fonctionnalités La première étape de la découverte de fonctionnalités est l'enregistrement d'une demande de découverte :

1. Un système de gestion de composition enregistre une demande auprès de l'aggrégateur de découverte. Lors de cet enregistrement, le système de gestion de composition peut fournir un filtre, sous la forme description sémantique de la fonctionnalité recherchée.
2. L'aggrégateur de découverte propage cette demande de découverte aux différents médiateurs.
3. Chaque médiateur traduit la demande de découverte pour l'infrastructure de services considérée. En particulier, il peut choisir d'utiliser ou non le filtre, en fonction de sa capacité à traduire les caractéristiques pour l'infrastructure de services.

La découverte de fonctionnalité se déroule selon les étapes suivantes :

1. lorsqu'un médiateur découvre un service potentiellement intéressant, il établit tout d'abord une description sémantique de la fonctionnalité fournissant ce service. La création de cette description sémantique peut se baser sur une description sémantique effectivement fournie par l'infrastructure de services ou sur un modèle de description défini pour cette infrastructure.
2. Le médiateur compare ensuite la description sémantique de la fonctionnalité avec le filtre fourni. Selon les cas, cette comparaison lui permet de rejeter le service ou de le conserver. Cette comparaison permet aussi d'établir la priorité de découverte, qui est d'autant plus important que la description de la fonctionnalité est proche de celle du filtre.
3. Si la fonctionnalité est intéressante, le médiateur stocke sa description dans l'espace de descriptions, puis notifie l'aggrégateur de découverte en lui fournissant les coordonnées de cette description ainsi que la priorité associée.
4. L'aggrégateur de découverte collecte les différentes propositions des médiateurs, et agit en fonction de la priorité associée à la découverte.
   (a) si la priorité est importante, le découvreur unifié informe immédiatement le système de gestion de composition.
   (b) si la priorité est faible, le découvreur stocke la description en vue d'informer le système de gestion de composition de manière différée. Le découvreur n'informe le système de gestion de composition qu'après un temps d'attente qui dépend du nombre de descriptions importantes reçues et du nombre total de descriptions reçues. Ce mécanisme privilégie ainsi le traitement prioritaire des descriptions ayant une meilleure correspondance avec le filtre, tout en laissant la possibilité au système de gestion de composition de traiter d'autres descriptions.
5. Lorsqu'un service précédemment sélectionné disparaît, le médiateur retire la description de fonctionnalité de l'espace de description et notifie l'aggrégateur de découverte de cette disparition. Le découvreur unifié notifie à son tour le système de gestion de composition (s'il lui avait proposé cette description).

Caractéristiques des médiateurs de découverte Le fonctionnement précis de chaque médiateur dépend des caractéristiques des mécanismes de découverte de l'infrastructure de services auquel il s'adresse. La table 5.1 résume les tâches à accomplir par un médiateur selon les caractéristiques de l'infrastructure de services :

b) Obtention d'informations sur le contexte

L'obtention d'informations sur le contexte est aussi un aspect fondamental du support pour l'acquisition de descriptions. En effet, l'ensemble des éléments de contexte pertinents pour une application donnée n'est pas défini *a priori*. Ce n'est qu'au cours du fonctionnement de l'application que le système de gestion de composition peut déterminer quels sont les éléments de contexte pertinents, en prenant notamment en compte les fonctionnalités disponibles. Cette section indique comment un système de gestion de composition peut définir dynamiquement quelles informations sur le contexte sont pertinentes pour l'application considérée. En revanche, les mécanismes de gestion d'information de contexte, qui permettent d'obtenir et d'agréger des informations issues de sources diverses disséminées dans un environnement attentif, sortent du cadre de ce travail. D'autres travaux (Euzenat *et al.*, 2008; Ricquebourg *et al.*, 2007) détaillent de tels mécanismes et précise comment les informations issues de capteurs peuvent être rendues disponibles sous la forme de descriptions sémantiques.

Caractéristiques		Exemples	Tâches du médiateur
Interaction	synchrone	SLP, UDDI	Le médiateur rend la découverte asynchrone en répétant périodiquement la découverte.
	asynchrone	Amigo, UPnP	Le médiateur gère éventuellement un bail.
Distribution	centralisé	SLP, UDDI,	Le médiateur connaît l'adresse du registre de services auquel s'adresser.
	décentralisé	JXTA, Amigo,	Le médiateur connaît le protocole définit par l'infrastructure de services pour effectuer la découverte.
Modèle de description	syntaxique	SLP, UPnP,	Le médiateur connaît le format de description utilisé et le transforme en une description sémantique basée sur l'ontologie de la plate-forme.
	sémantique	Amigo	Le médiateur connaît l'ontologie de description utilisée et utilise un alignement d'ontologie pour préciser les concepts utilisés en utilisant l'ontologie de la plate-forme.

TABLE 5.1 – Tâches d'un médiateur de découverte

Par nature, les informations sur le contexte sont dynamiques : elles évoluent constamment au cours du fonctionnement d'une application. L'obtention des informations sur le contexte fonctionne selon un principe de souscription/notification. À tout moment du fonctionnement d'une application, le système de gestion de composition peut s'inscrire pour recevoir des informations sur un élément de contexte particulier. On appelle *intérêt sur le contexte* une requête dans laquelle un système de gestion de composition décrit une information de contexte qui l'intéresse.

Pour exprimer un intérêt sur le contexte, FCAP adopte le langage SPARQL (W3C, 2008). SPARQL est un standard W3C qui permet d'exprimer des requêtes sur des graphes RDF à l'aide d'une syntaxe similaire à SQL (qui est utilisé pour interroger des bases de données classiques).

Le registre des intérêts sur le contexte permet au système de gestion de composition d'enregistrer ses intérêts pour les informations de contexte, de manière à être tenu informé de l'évolution de ces éléments de contexte.

Exemple La figure 5.16 illustre un exemple de description d'information sur le contexte. L'intérêt sur le contexte exprimé par le système de gestion de composition prend la forme SPARQL suivante : SELECT ?room WHERE { envThom:Thom env:isLocatedIn ?room }. Cette requête exprime que le système de gestion de composition souhaite savoir dans quelle pièce se trouve Thom.

La figure présente deux états de l'espace de description, contenant les documents suivants :

ENVIRONMENT définit une ontologie générique pour l'expression d'informations sur le contexte physique. Dans cette ontologie, env:Person exprime le concept de personne, env:Room le concept de pièce et env:Device le concept de dispositif. env:isLocatedIn exprime la propriété d'une personne d'être dans une pièce et env:perceives exprime la propriété d'une personne de percevoir un dispositif.

ENV-THOM définit une description de la maison de Thom, qui utilise les concepts de l'ontologie générique du contexte. Cette description indique que envThom:Thom est une personne, que envThom:LivingRoom, envThom:Kitchen et envThom:Bedroom sont des pièces.

LOCATION-142 est une description particulière du contexte de Thom dans une situation donnée. Elle exprime que Thom est dans la cuisine.

LOCATION-143 est une description particulière du contexte de Thom à un autre situation. Elle exprime que Thom est dans le salon.

Dans chacune des situations, le registre d'intérêt sur le contexte informe le système de gestion de composition. Dans la première situation, la description du contexte pertinent pour répondre à la requête est *room*=(kitchen,LOCATION-142) (en notant kitchen=envThom:kitchen). Dans la seconde situation, la description du contexte pertinent est *room*=(living,LOCATION-143) (en notant living=envThom:LivingRoom).

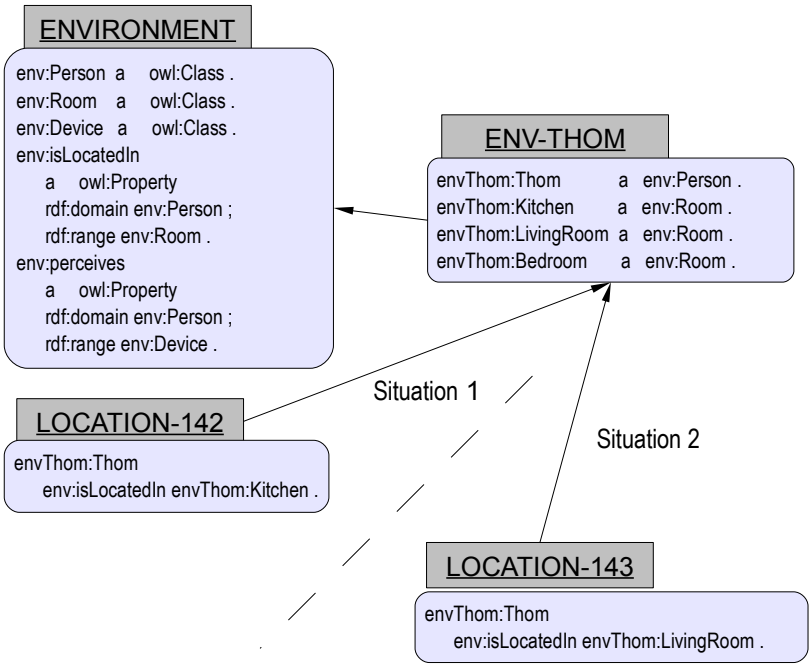

FIGURE 5.16 – Exemple d'obtention d'informations sur le contexte

5.3.3 Mise en place de configurations

Le troisième aspect du support fourni par FCAP concerne la mise en place de configuration, qui permet de réaliser une configuration choisie par un système de gestion de composition. Lorsque le système de gestion de composition a déterminé la description d'une configuration adaptée, le support fourni par FCAP permet d'effectuer les opérations nécessaires à la mise en place effective de cette configuration dans l'environnement.

Le support pour la mise en place de configurations est constitué de trois composantes :

Le contrôle générique de fonctionnalité permet de manipuler une fonctionnalité de manière abstraite, indépendamment de l'infrastructure de services sur laquelle

elle repose. Le contrôle de fonctionnalité se focalise sur quelques opérations fondamentales pour la composition, et offre au système de gestion de composition une interface unifiée pour réaliser ces opérations.

Le contrôle générique de connecteur permet de manipuler les connecteurs de manière abstraite, indépendamment des modes de communication employés. Le contrôle de connecteur se focalise sur quelques opérations fondamentales pour la composition, et offre au système de gestion de composition une interface unifiée pour réaliser ces opérations.

L'assemblage de configuration permet d'assembler une configuration selon une description fournie par le système de gestion de composition. L'assemblage de configuration établit les connexions souhaitées entre les fonctionnalités et gère l'évolution entre les configurations successives.

a) Contrôle générique de fonctionnalités

De manière générale, les fonctionnalités présentes dans un environnement attentif sont hétérogènes, et la manière dont elles peuvent être invoquées varie selon les infrastructures de services. FCAP considère une vue relativement abstraite des fonctionnalités, définie par le modèle de description de fonctionnalité (section 5.2.1, figure 5.5). Cependant, l'utilisation d'une fonctionnalité nécessite généralement des informations plus précises. Ces informations sont présentes dans la description mais peuvent être ignorées par les mécanismes de composition génériques. Elles sont uniquement interprétées au moment de la mise en place d'une composition. Cette section décrit les différents éléments qui permettent à un système de gestion de composition de manipuler des fonctionnalités de manière générique.

Notion de contrôleur de fonctionnalité Afin de rendre générique les mécanismes de contrôle de fonctionnalité, l'approche proposée par FCAP se décompose en deux étapes. Tout d'abord, un *contrôleur de fonctionnalité* est généré en utilisant les informations contenues dans la description de la fonctionnalité. Ensuite, les opérations de composition sont réalisées par l'intermédiaire de ce contrôleur de fonctionnalité. Pour chaque opération de composition générique, le contrôleur de fonctionnalité invoque les services fournis par la fonctionnalité, en utilisant des opérations spécifiques à celle-ci.

Tous les contrôleurs de fonctionnalité présentent la même interface générique, qui se compose des opérations suivantes :

Initialisation de l'utilisation permet de mettre la fonctionnalité dans un état correspondant aux besoins de l'application. Les opérations d'initialisation sont par exemple la fourniture d'un identifiant d'utilisateur, la mise en place d'une session. Pour effectuer cette opération, le contrôleur doit généralement utiliser des informations contenues dans la description, telle que l'identifiant de l'utilisateur.

Obtention d'une référence pour un service fourni permet d'obtenir un ensemble d'information qui permet d'accéder au service (son adresse réseau ...). Cet ensemble d'information est appelé référence du service. La manière dont est exprimée cette référence dépend de l'infrastructure de services et de la fonctionnalité considérée.

Attachement à un service attendu permet d'établir une connexion entre deux fonctionnalités, en attachant la référence du service fourni par l'une au service attendu par l'autre. L'attachement à un service attendu s'effectue à l'aide d'opérations fournies par la fonctionnalité. La forme de ces opérations dépend de l'infrastructure de services et de la fonctionnalité considérée.

Détachement d'un service attendu permet de supprimer une connexion lorsqu'il faut cesser d'utiliser un service fourni. Là encore, le détachement d'un service attendu s'effectue à l'aide de certaines opérations fournies par la fonctionnalité.

Finalisation de l'utilisation permet de finaliser l'utilisation, lorsque la fonctionnalité n'est plus utilisée. Selon le cas, le contrôleur peut alors invoquer le service pour finaliser une session, relâcher des ressources, etc.

Les générateurs de contrôleur Pour obtenir un contrôleur correspondant à chacune des fonctionnalités, FCAP utilise des générateurs de contrôleur. Un générateur de contrôleur est spécifique à une certaine catégorie de fonctionnalités, voire à une fonctionnalité particulière, et fournit les opérations suivantes :

Inspection d'une description Pour une description donnée, un système de gestion de composition ne sait pas *a priori* quel générateur de contrôleur s'applique. Cette opération permet de demander à chacun des générateurs disponibles d'inspecter la description afin de déterminer sa capacité à générer un contrôleur pour la fonctionnalité décrite.

Génération d'un contrôleur Lorsqu'un générateur s'applique pour une description, cette opération permet d'obtenir le contrôleur pour la fonctionnalité décrite.

Comme FCAP fait des hypothèses minimales, l'implémentation des différents contrôleurs et des différents générateurs de contrôleur doit être adaptée aux différents cas de fonctionnalités rencontrées. En particulier, les descriptions contiennent des informations spécifiques qui sont exploitées par un générateur de contrôleur, tout en étant ignorées par les mécanismes de composition génériques.

Le générateur universel de contrôleurs de fonctionnalité Afin de simplifier l'utilisation des mécanismes de génération de contrôleur, FCAP propose un générateur universel de contrôleur. Un système de gestion de composition s'adresse uniquement à ce générateur universel, qui redirige ensuite les demandes de génération vers les générateurs spécifiques. Pour cela, il fait tout d'abord inspecter la description par les différents générateurs spécifiques disponibles, puis en demande la génération au générateur le plus approprié. Il fournit ensuite le contrôleur de fonctionnalité résultant au système de gestion de composition.

b) Contrôle générique de connecteurs

De la même manière, FCAP propose un mécanisme de contrôle générique des connecteurs. Ce mécanisme permettent à un système de gestion de composition de contrôler l'établissement de connexion entre des fonctionnalité en faisant abstraction des protocoles réseau employés, ainsi que des éventuelles adaptations nécessaires entre les services fournis et attendus par les fonctionnalités.

Notion de contrôleur de connecteur Afin de fournir une approche générique pour le contrôle de connecteur, FCAP propose une approche en deux étapes. Tout d'abord, un *contrôleur de connecteur* est généré en utilisant les informations contenues dans la description du connecteur. Ensuite, les opérations de composition sont réalisées par l'intermédiaire de ce contrôleur de connecteur. Pour chaque opération de composition générique, le contrôleur de connecteur invoque les opérations nécessaires sur les fonctionnalités concernées. Dans le cas où une adaptation entre les services fournis et attendus par les fonctionnalités est nécessaire, le contrôleur de connecteur prend aussi en charge la mise en place des adaptateurs.

L'interface générique des contrôleurs de connecteur présente les opérations suivantes :

Initialisation de l'utilisation prépare la connexion, éventuellement en initialisant des services distants qui seront utilisés au cours des interactions entre les services concernés par la connexion (relais, adaptateurs ...).

Mise en place de la connexion connecte les fonctionnalités. Cette opération exploite les contrôleurs des fonctionnalités reliées par le connecteur pour effectuer les opérations de composition.

Suppression de la connexion déconnecte les fonctionnalités. Cette opération exploite les contrôleur des fonctionnalités reliées par le connecteur pour effectuer les opérations de composition.

Finalisation de l'utilisation termine l'utilisation de la connexion, en libérant éventuellement des ressources mobilisées pour son fonctionnement.

Les générateurs de contrôleur Pour obtenir un contrôleur spécifique pour un connecteur donné, FCAP utilise des générateurs de contrôleur. Un générateur s'appuie sur la description du connecteur pour fournir les opérations permettant la mise en place de connexions. Pour générer un contrôleur de connecteur, les contrôleurs des fonctionnalités reliées par ce connecteur doivent de plus être fournis au générateur. Chaque générateur de connecteur fournit les opérations suivantes

Inspection d'une description Cette opération permet de demander à chacun des générateurs disponibles d'inspecter la description afin de déterminer sa capacité à générer un contrôleur pour le connecteur décrit.

Génération d'un contrôleur Lorsqu'un générateur s'applique pour une description, cette opération permet d'obtenir le contrôleur pour le connecteur décrit. Cette opération nécessite que les contrôleurs des fonctionnalités reliées par le connecteur soient fournis.

Le générateur universel de contrôleur de connecteur Comme précédemment, FCAP propose un générateur universel pour les contrôleurs de connecteur. Un système de gestion de composition s'adresse uniquement à ce générateur universel, qui redirige ensuite les demandes de génération vers les générateurs spécifiques. Pour cela, il fait tout d'abord inspecter la description par les différents générateurs spécifiques disponibles, puis en demande la génération au générateur le plus approprié. Il fournit ensuite le contrôleur résultant au système de gestion de composition.

c) Assemblage de configurations

Au cours du fonctionnement d'une application composite flexible, plusieurs configurations sont successivement mises en place, en fonction de la situation de l'environnement. L'assembleur de configuration permet de construire et de modifier dynamiquement les assemblages de fonctionnalités correspondant aux configurations choisies par le système de gestion de composition. Il s'appuie pour cela sur la description de la configuration à réaliser, pour laquelle il détermine une procédure de mise à jour par rapport à l'assemblage existant (s'il en existe un). La mise à jour s'effectue en invoquant les contrôleurs des fonctionnalités participant à l'assemblage.

Interaction avec un système de gestion de composition Pour chaque nouvelle configuration à mettre en place, le système de gestion de composition de l'application fournit la configuration à atteindre à l'assembleur de services. Celui-ci doit alors effectuer les opérations de connexion et de déconnexion nécessaires pour atteindre la configuration souhaitée à partir de la situation courante.

Enregistrement d'une application Avant de mettre en place une configuration, un système de gestion de composition doit s'enregistrer auprès de l'assembleur,

en fournissant un identifiant pour son application. Cet identifiant permet d'identifier l'application concernée : en effet, l'assembleur de services maintient à jour un modèle de la configuration courante pour chacune des applications composites flexibles en fonctionnement dans l'environnement. Lorsqu'une nouvelle configuration est demandée, l'assemblage de services utilise ce modèle de la configuration courante pour déterminer les opérations à effectuer. Il met ensuite son modèle de la configuration courante à jour.

Définition d'une configuration à mettre en place Pour mettre en place une configuration, un système de gestion de composition fournit ainsi à l'assembleur de services une description de la configuration souhaitée. L'assembleur de services interroge alors l'usine à contrôleur pour obtenir les contrôleurs nécessaires, puis effectue les opérations de connexion/déconnexion nécessaires à la mise en place de la configuration souhaitée.

Une configuration à mettre en place est décrite par :

- un ensemble F de descriptions de fonctionnalités à assembler. Une description de fonctionnalité est généralement la description enrichie d'une description de fonctionnalité découverte, dans laquelle des informations spécifiques à l'application ont été ajoutées par les mécanismes d'interprétations.
- un ensemble C de descriptions des connexions à établir entre les services fournis et attendus de ces fonctionnalités. Ces descriptions sont générées par certains mécanismes d'interprétation.

Notification de la configuration courante En raison des imprévus qui peuvent survenir dans l'environnement, une configuration mise en place pour une application peut subir des modifications non souhaitées (perte de connexion, panne d'une fonctionnalité). Dans certains cas, l'assembleur de services peut être averti de ces modifications et modifier son modèle de la configuration courante en conséquence. Il doit alors chercher à atteindre de nouveau la configuration souhaitée. S'il n'y parvient pas, il doit avertir le système de gestion de composition concerné par ces modifications.

Déroulement de l'assemblage de fonctionnalités Au cours du fonctionnement de l'application, la mise en place d'une configuration particulière se déroule de la manière suivante :

1. le système de gestion de composition demande la mise en place de la configuration, en fournissant les ensembles de fonctionnalités F et de connecteurs C.
2. l'assembleur de configuration détermine l'évolution à effectuer par rapport à la configuration précédente. En notant F' l'ensemble de fonctionnalités précédente et C' l'ensemble de connecteurs précédent, il détermine ainsi :

- – l'ensemble $F^+ = F \setminus (F \cap F')$ des fonctionnalités ajoutées à la configuration.
- – l'ensemble $F^- = F' \setminus (F \cap F')$ des fonctionnalités retirées de la configuration.
- – l'ensemble $C^+ = C \setminus (C \cap C')$ des connecteurs ajoutés à la configuration.
- – l'ensemble $C^- = C' \setminus (C \cap C')$ des connecteurs retirés de la configuration.

1. pour chacune des nouvelles fonctionnalités décrites dans F^+, l'assembleur de configuration obtient les contrôleurs de fonctionnalités correspondants. Il utilise ces contrôleurs pour initialiser l'utilisation de chaque fonctionnalité.
2. pour chacun des nouveaux connecteurs décrits dans C^+, l'assembleur de configuration obtient les contrôleurs de connecteurs correspondants. Pour cela, il doit fournir les contrôleurs des différentes fonctionnalités mises en relations par chaque connecteur. Il utilise les contrôleurs de connecteur pour initialiser la mise en place de connexion.
3. l'assembleur de configuration effectue enfin la mise en place de la configuration :

(a) pour chaque connecteur de C^-, il utilise le contrôleur correspondant pour déconnecter les fonctionnalités reliées par le connecteur.

(b) pour chaque connecteur de C^+, il utilise le contrôleur correspondant pour connecter les fonctionnalités reliées par le connecteur.

(c) pour chaque fonctionnalité de F^-, il utilise le contrôleur correspondant pour finaliser l'utilisation de la fonctionnalité.

5.4 Synthèse

Ce chapitre a décrit une approche générique pour la composition flexible d'applications dans les environnements attentifs. L'approche FCAP est basée sur la notion d'*application composite flexible* : une application est considérée comme un système de fonctionnalités faiblement couplées dont la configuration dépend des fonctionnalités disponibles, de la situation de l'environnement et du besoin de l'utilisateur. Ce chapitre présente la notion de *système de gestion de composition*, qui est chargé de déterminer la configuration d'une application en fonction de la situation. L'élément principal du support fourni par FCAP est l'*espace de description*, qui permet d'effectuer la gestion d'une composition en manipulant des descriptions. On définit de plus les mécanismes grâce auxquels FCAP s'intègre aux infrastructures existant dans les environnements attentifs.

L'approche proposée par FCAP se caractérise principalement par le découplage entre les fonctionnalités, les besoins et les stratégies d'adaptation. Ce découplage est assuré par l'utilisation de l'espace de descriptions, dans lequel les informations relatives aux différentes entités sont modélisées sous la forme de descriptions sémantiques. À l'aide de l'espace de descriptions, le système de gestion de composition d'une application peut manipuler ces informations pour élaborer une configuration adaptée en fonction de la situation. Cette approche prend ainsi en compte les principales problématiques de conception des applications attentives :

L'hétérogénéité des fonctionnalités est prise en compte par la définition d'un modèle de description de haut-niveau, qui peut être étendu pour exprimer les particularité de différents types de fonctionnalités. Les informations pertinentes pour décrire les fonctionnalités ne sont ainsi pas restreintes *a priori*, et de nombreux aspects peuvent être intégrés à l'espace de description. FCAP prévoit des mécanismes pour l'interprétation des aspects de descriptions spécifiques, ainsi que pour l'intégration de différentes infrastructures de services.

L'évolution de l'environnement est prise en compte par la mise à jour continuelle de l'espace de descriptions pour intégrer les nouvelles informations. Cette mise à jour est facilitée par la notion de document de description, qui isole les différents éléments de description des fonctionnalités et de l'environnement. Les informations peuvent donc être mis à jour indépendamment et sont dynamiquement intégrées par les mécanismes de gestion de composition.

L'imprécision des besoins des utilisateurs est prise en compte par la notion de description d'application abstraite, qui découple la définition des besoins et les fonctionnalités présentes dans un environnement.

Ce chapitre a présenté une vision globale de FCAP, en précisant ces principes et son intégration dans l'architecture de référence. Les chapitres suivants s'intéressent plus précisément au fonctionnement des systèmes de gestion de composition. Tout d'abord, le chapitre 6 détaille les mécanismes de gestion de composition qui permettent la manipulation de descriptions en vue de l'élaboration de la description d'une configuration adaptée pour réaliser une application dans une situation donnée. Ensuite, le chapitre 7 présente une architecture générique pour les systèmes de gestion de composition.

Chapitre 6

Mécanismes de gestion de composition flexible d'applications

Le chapitre 5 décrit les principes généraux de l'approche FCAP. Il décrit en particulier l'approche de composition par manipulation de descriptions, qui constitue l'élément essentiel de la flexibilité de FCAP. Pour chaque application considérée, un système de gestion de composition interprète des descriptions de fonctionnalités et d'applications et construit dynamiquement la description d'une configuration adaptée à la situation courante. Cette approche assure l'intégration de fonctionnalités hétérogènes, l'adaptation continue de l'application aux conditions rencontrées et la possibilité de modifier les besoins des utilisateurs au cours du fonctionnement.

Ce chapitre décrit plus précisément les mécanismes de gestion flexible de composition. Grâce à ces mécanismes, un système de gestion de composition est capable de manipuler les descriptions afin de déterminer une configuration adaptée. Deux catégories de mécanismes entrent en jeu :

Les mécanismes d'interprétation extraient des informations sur les fonctionnalités à partir de leurs descriptions. Ces mécanismes sont réalisés par des algorithmes d'interprétation indépendants. Chaque algorithme prend en entrée des descriptions de fonctionnalités et fournit en sortie un document décrivant l'interprétation effectuée et une estimation de l'utilité de cette interprétation.

Le mécanisme de résolution détermine une configuration adaptée à une situation donnée. Ce mécanisme repose sur la construction d'un problème de satisfaction de contraintes associé à une application. Un algorithme générique de résolution de contrainte permet ainsi d'examiner les configurations possibles pour déterminer la configuration plus appropriée dans une situation donnée. Il s'appuie sur les résultats fournis par les algorithmes d'interprétation pour obtenir des informations spécifiques sur les différentes contraintes.

La section 6.1 présente plusieurs mécanismes d'interprétation de description qui peuvent être utilisés pour la composition d'applications attentives. Ces différents mécanismes illustrent les possibilités offertes par cette approche et décrivent les modes de fonctionnements principaux des mécanismes d'interprétation. La section 6.2 présente comment la recherche d'une configuration adaptée est réalisée à l'aide d'un problème de satisfaction de contraintes. Cette section détaille comment le problème de satisfaction de contraintes est exprimé à partir d'une description d'application et comment la description d'une configuration adaptée à une situation est ensuite produite.

6.1 Mécanismes d'interprétation

Comme on l'a vu dans le chapitre précédent, la gestion de composition repose sur différents types d'interprétations de fonctionnalités. Pour chaque type d'interprétation, un mécanisme particulier est utilisé pour obtenir une interprétation à partir de descriptions de fonctionnalités abstraites et réelles. Pour chaque mécanisme d'interprétation, les éléments suivants doivent être définis :

Les éléments de descriptions considérés sont les éléments de descriptions qui sont pris en charge par le mécanisme considéré. Les informations sur ces éléments sont extraites des descriptions de fonctionnalités fournies en paramètre.

Le modèle de description de l'interprétation définit les éléments de descriptions qui permettent d'exprimer le résultat de l'interprétation produite par l'algorithme.

L'algorithme d'interprétation définit les opérations effectuées pour réaliser l'interprétation. L'algorithme se déroule généralement en trois phases : la lecture des informations pertinentes dans les descriptions de fonctionnalités, le traitement des informations et l'écriture de l'interprétation résultante. Pour les opérations de lecture et d'écriture, les algorithmes présentés utilisent en particulier les opérations fournies par une base de connaissance 3DM, décrites dans l'annexe A.

En plus d'une description de l'interprétation, chaque mécanisme fournit une estimation de l'utilité de cette interprétation. L'utilité d'une interprétation est ici exprimée à l'aide d'une notion de coût et de coût limite. Pour une interprétation de type τ, le coût est noté Δ_τ et le coût limite est noté λ_τ. L'utilité d'une interprétation $\underline{\tau}[(f,a)]$ est calculée par $\mu(\underline{\tau}[(f,a)]) = (\lambda_\tau/\Delta_\tau[(f,a)]) - 1$. L'utilité est ainsi positive lorsque le coût est inférieur au coût limite et négative lorsqu'il est supérieur. On considère que l'utilité est infinie lorsque le coût est nul. Ce mode de calcul permet d'une part de discriminer les utilités positives et négatives, et d'autre part de comparer les utilités entre elles lorsqu'elles sont positives.

Cette section détaille ces éléments pour trois mécanismes d'interprétation :
- La section 6.1.1 présente un mécanisme d'interprétation portant sur la description du type des fonctionnalités à l'aide de concepts définis dans des ontologies.
- La section 6.1.2 présente un mécanisme d'interprétation portant sur les informations disponibles sur le contexte physique.
- La section 6.1.3 présente un mécanisme d'interprétation portant sur la correspondance entre les services fournis et attendus par deux fonctionnalités.

6.1.1 Interprétation des types de fonctionnalités

Un des intérêts de l'utilisation de descriptions sémantiques est la possibilité d'exploiter les techniques de raisonnement basées sur la logique de description. La mise en correspondances sémantique (*semantic matchmaking*) est notamment utilisée dans le cas des services Web sémantiques (section 3.1.2). Ces techniques reposent sur l'utilisation d'ontologies pour comparer les concepts employés pour décrire les ressources, à l'aide d'une notion de *distance sémantique*. Cette section décrit un mécanisme d'interprétation fondé sur ce principe : il interprète la correspondance entre une fonctionnalité abstraite et une fonctionnalité réelle.

a) Éléments de description considérés

Dans une description, on qualifie une ressource $\underline{r}$ d'*instance* d'une classe (ou concept) $\underline{t}$ dans un document $\underline{D}$ lorsque $\underline{D}$ contient le triplet ($\underline{r}$ rdf:type $\underline{t}$). Pour

comparer deux ressources, on peut considérer l'ensemble des classes dont elles sont des instances. Des ressources qui sont instances des mêmes classes peuvent être considérées comme proches. Des ressources qui ne sont instances d'aucune classe commune représentent vraisemblablement des entités très différentes.

Exemple Pour l'exemple de la section 2.1.2, une partie de la description du cadre photo a été représentée dans la figure 5.6. La figure 6.1 précise plusieurs types pour la fonctionnalité, exprimés à l'aide de deux ontologies de domaine. L'ontologie notée multimedia définit des concepts liés aux fonctionnalités multimédia. Ici, la description indique que la fonctionnalité pframe:msg_displayer permet d'afficher des images (media:Displayer). L'ontologie notée notification définit des concepts liés aux fonctionnalités de notification. Ici, la description indique que pframe:msg_displayer permet d'effectuer des notifications visuelles (notification:MsgNotifier). De même, une partie de la description de l'application a été représentée dans la figure 5.4. La figure 6.1 précise un type pour la fonctionnalité de notification, exprimées à l'aide de l'ontologie notification.

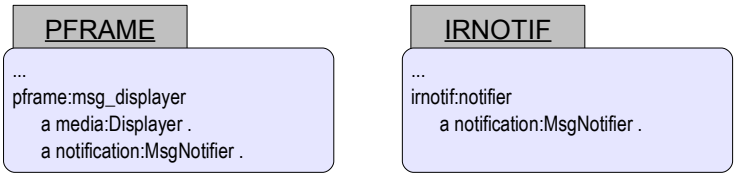

FIGURE 6.1 – Partie de descriptions indiquant les types de fonctionnalité

Il est possible de définir diverses fonctions fournissant une indication de la proximité entre deux classes (ou concepts) d'une ontologie. Dans la suite, on se base sur la fonction `compareConcepts`() suivante. Cette fonction calcule le degré de correspondance d'une classe $\underline{t'}$ par rapport à une classe $\underline{t}$, en considérant successivement les cas où $\underline{t'}$ est égale à $\underline{t}$, équivalente à $\underline{t}$, sous-classe de $\underline{t}$ ou s'il existe une classe dont $\underline{t}$ et $\underline{t'}$ sont sous-classes. Noter que cette fonction n'est pas symétrique : la classe $\underline{t}$ sert de référence par rapport à laquelle on évalue la classe $\underline{t'}$.

b) Modèle de description de l'interprétation

L'algorithme d'interprétation produit un document décrivant les correspondances entre la fonctionnalité abstraite et la fonctionnalité réelle à l'aide de l'ontologie représentée dans la figure 6.2.

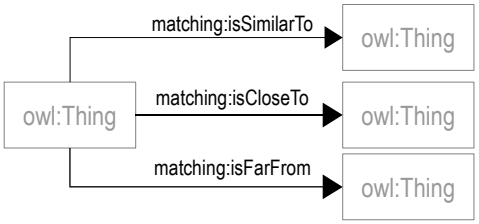

FIGURE 6.2 – Ontologie de description des correspondances

Les propriétés définies dans cette ontologie représentent trois degrés de similarité :

matching:isSimilarTo indique que la fonctionnalité réelle est similaire à la fonctionnalité abstraite.

Fonction `compareConcepts`($\underline{t}, \underline{t'}$: Ressource) → d : degré de correspondance

Entrées : une ressource $\underline{t}$, une ressource $\underline{t'}$, représentant des concepts
Sorties : un degré de correspondance d

```
1  début
3      si t = t' alors
4          retourner 0;
6      sinon si 3db.ask(t' owl:equivalentClass t) alors
7          retourner 0;
9      sinon si 3db.ask(t' rdfs:subClassOf t) alors
10         retourner 5;
12     sinon si ∃u | 3db.ask(t', rdfs:subClassOf; u) et 3db.ask(t,
       rdfs:subClassOf, u) alors
13         retourner 10;
15     sinon
16         retourner 1000;
17     fin
18 fin
```

matching:isCloseTo indique que la fonctionnalité est proche de la fonctionnalité abstraite mais pas exactement similaire.

matching:isFarFrom indique que la fonctionnalité est éloignée de la fonctionnalité abstrait.

Exemple La figure 6.3 représente la description obtenue pour l'interprétation de la correspondance entre la fonctionnalité de notification et le cadre photo, dans l'hypothèse où le cadre photo est proche de la fonctionnalité abstraite.

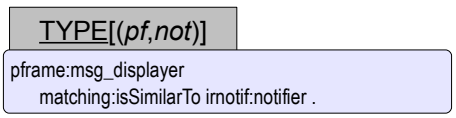

FIGURE 6.3 – Exemple de description de correspondance

c) Algorithme d'interprétation

L'algorithme d'interprétation vise à comparer les types définis pour une fonctionnalité réelle aux types définis pour la fonctionnalité abstraite considérée. Pour cela, il utilise une fonction `matchIndividual`(), qui compare deux ressources sur la base de leurs types. Le déroulement de la fonction est le suivant :

1. lecture de l'ensemble des types des ressources $\underline{a}$ et $\underline{f}$.
2. pour chacun des types de $\underline{a}$, recherche du type $\underline{f}$ de dont le degré de correspondance est le plus petit.
3. sommation des degrés de correspondances minimaux obtenus pour chaque type de $\underline{a}$.

L'algorithme d'interprétation `InterpretTypes` (page 114) se déroule en trois étapes :

Fonction `matchIndividual`($\underline{a}, \underline{f}$: Ressource) → d : degré de correspondance

Entrées :
une ressource $\underline{a}$ issue d'une description de fonctionnalité abstraite,
une ressource $\underline{f}$ issue d'une description de fonctionnalité réelle
Sorties : un degré de correspondance *cost*

```
1  début
2      T_a ← 3db.selectAll(a, rdf:type, ?t);
3      T_f ← 3db.selectAll(f, rdf:type, ?u);
4      cost ← 0;
5      pour tous les t ∈ T_a faire
6          d_t ← +∞;
7          pour tous les t' ∈ T_b faire
8              d_t,t' ← compareConcepts(t,t');
9              si d_t,t' < d_t alors d_t ← d_t,t';
10             si d_t = 0 alors sortir boucle ;
11         fin
12         cost ← cost + d_t;
13     fin
14     retourner d;
15 fin
```

1. lecture des descriptions de la fonctionnalité abstraite et de la fonctionnalité réelle.
2. comparaison des types des deux fonctionnalités, à l'aide de la fonction `matchIndividual`().
3. création d'un document et écriture de l'interprétation en fonction du degré de correspondance. Ce document indique la correspondance entre la fonctionnalité abstraite et la fonctionnalité réelle, choisie en fonction de la valeur de Δ_τ. En particulier, si Δ_τ est inférieur à un certain seuil Δ_{max}, on considère que la fonctionnalité réelle est proche de la fonctionnalité abstraite, sinon on considère qu'elle est éloignée.

6.1.2 Interprétation de conditions de contexte

Les contraintes portant sur le contexte permettent de caractériser certains éléments du contexte dans lequel une fonctionnalité peut fonctionner. Ces contraintes sont exprimées à l'aide de *conditions de contexte* : ce sont des conditions qui impliquent des entités de l'environnement et qui doivent être vérifiées pour autoriser l'utilisation du service. Les entités sur lesquelles portent ces conditions sont appelées *paramètres de contexte*. Certains paramètres peuvent être des variables, qui doivent alors être instanciées de manière à valider les conditions.

L'objectif du mécanisme d'interprétation de condition de contexte est de déterminer si les conditions apparaissant sur les fonctionnalités (abstraites et réelles) sont vérifiées dans la situation courante. La vérification des conditions de contexte contenues dans les descriptions peut être réalisée à l'aide d'un système de gestion de d'information de contexte (section 5.3.2).

Algorithme 3 : `InterpretTypes`

Données :
une description de fonctionnalité abstraite ($\underline{a}, \underline{A}$)
une description de fonctionnalité réelle ($\underline{f}, \underline{F}$)
Résultat : un document $\underline{\tau}$ décrivant l'interprétation, un coût Δ_τ

```
1  début
2      3db.load(A);
3      3db.load(F);
4      Δτ ← matchIndividual(a,f);
5      τ ← 3db.createDoc();
6      3db.addImport(τ, F);
7      3db.addImport(τ, A);
8      si Δτ = 0 alors
9          3db.write(τ, f, matching:isSimilarTo, a);
10     sinon si Δτ ≤ Δmax alors
11         3db.write(τ, f, matching:isCloseTo, a);
12     sinon
13         3db.write(τ, f, matching:isFarFrom, a);
14     fin
15     setInterpretation(τ, Δτ)
16 fin
```

a) Éléments de description considérés

L'expression de conditions sur le contexte nécessite l'introduction d'un modèle de description de ces conditions. La figure 6.4 représente le modèle utilisé. Ce modèle se base sur l'ontologie SWRL (*Semantic Web Rule Language*, (W3C, 2004b)), qui définit des concepts permettant la descriptions de règles.

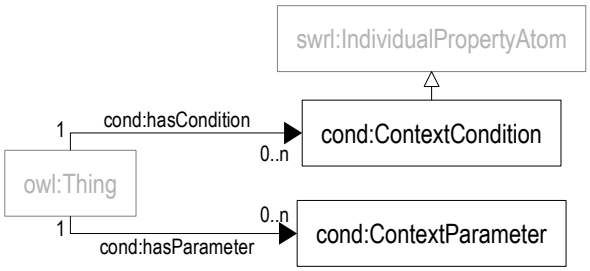

FIGURE 6.4 – Ontologie des conditions sur le contexte

Cette ontologie définit les deux concepts suivants :

cond:ContextParameter représente un paramètre de contexte. N'importe quelle ressource peut être un paramètre de contexte (de type cond:ContextParameter). Un paramètre variable (c'est-à-dire dont la valeur n'est pas définie dans la description) est à la fois instance de cond:ContextParameter et de swrl:Variable.

cond:ContextCondition représente une condition de contexte. Elle est définie comme une sous-classe de swrl:IndividualPropertyAtom, qui représente une clause SWRL portant sur la valeur d'une propriété. Les arguments de la condition doivent être des instances de cond:ContextParameter (mais peuvent aussi être instances de swrl:Variable).

Exemple La figure 6.5 présente une condition de contexte portant sur le cadre photo. Cette condition indique que l'utilisateur de la fonctionnalité fournie par le cadre photo doit percevoir ce cadre photo.

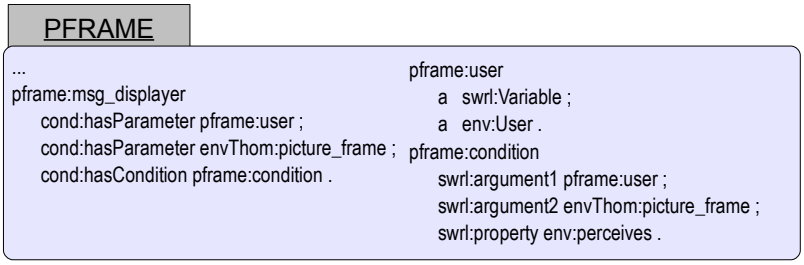

FIGURE 6.5 – Éléments de description de conditions de contexte pour le cadre photo

La figure 6.6 présente uniquement un paramètre de contexte portant sur la fonctionnalité abstraite de notification.

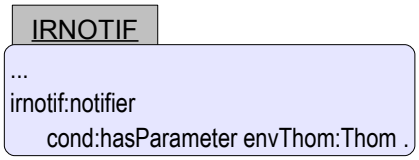

FIGURE 6.6 – Éléments de description de conditions de contexte pour la fonctionnalité abstraite de notification

b) Modèle de description de l'interprétation

Le résultat de l'interprétation de conditions de contexte est décrit par
- les correspondances éventuelles entre les paramètres de contexte des descriptions de fonctionnalités abstraites et réelles, ainsi que les correspondances éventuelles entre un paramètre de contexte variable et une entité (non-variable) de l'environnement. Ces correspondances sont exprimées à l'aide de l'ontologie de correspondance présentée dans la section précédente (section 6.1.1, figure 6.2).
- des informations sur la validité de la condition. Ces informations sont exprimées à l'aide de l'ontologie représentée dans la figure 6.7 :

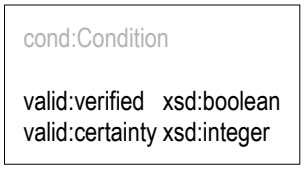

FIGURE 6.7 – Ontologie de vérification des conditions de contexte

Cette ontologie définit deux propriétés qui caractérisent la vérification d'une condition :

valid:verified indique si la condition est vérifiée dans la situation courante.

valid:certainty indique le degré de certitude de la vérification (entre 0 et 100). Dans certain cas, la condition est peut-être vérifiée, mais la vérification n'a pas pu être déterminée de manière certaine.

Exemple La figure 6.8 présente la description de la vérification de la condition de contexte portant sur le cadre photo s'il est utilisé comme dispositif de notification pour iRider. Deux informations y sont notées :

1. le paramètre variable pframe:user est identifié au paramètre constant envThom:Thom.
2. la condition (pframe:user ctxt:canPerceive envThom:picture_frame), transformée en (envThom:Thom ctxt:canPerceive envThom:picture_frame) (après instanciation de la variable pframe:user) est vérifiée dans la situation courante.

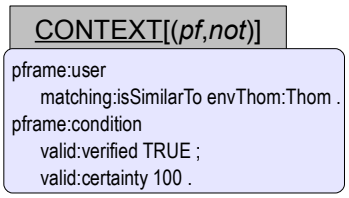

FIGURE 6.8 – Exemple de description de la vérification d'une condition

c) Algorithme d'interprétation

L'algorithme d'interprétation `InterpretContextCondition` (page 117) évalue une condition de contexte en trois étapes :

1. identification de la correspondance des paramètres de contexte dans les descriptions de fonctionnalité abstraite et de fonctionnalité réelle.
2. instanciation des paramètres variables pour lesquels il existe une correspondance avec un paramètre fixé.
3. évaluation de la validité de la condition.
4. écriture des résultats dans un document d'interprétation

Tout d'abord, l'étape d'identification de la correspondance utilise une fonction `matchIndividualSet`(), qui permet de déterminer des correspondances entre deux ensembles de ressources similaires. Ici, cette fonction est utilisée avec les ensembles de paramètres de contexte définis pour la fonctionnalité abstraite et pour la fonctionnalité réelle. La fonction `matchIndividualSet`() utilise la fonction `matchIndividual`() définie précédemment (section 6.1.1) pour déterminer les correspondances entre les

Algorithme 4 : `InterpretContextCondition`

Données :
une description de fonctionnalité abstraite $(\underline{a}, \underline{A})$
une description de fonctionnalité réelle $(\underline{f}, \underline{F})$
Résultat : un document $\underline{\tau}$ décrivant l'interprétation, un coût Δ_τ

```
1  début
2      3db.load(A);
3      3db.load(F);
4      P_a ← 3db.selectAll(a, cond:hasParameter, ?p);
5      P_f ← 3db.selectAll(f, cond:hasParameter, ?p);
6      M ← matchIndividualSet(P_a, P_f);
7      c ← 3db.select(f, cond:hasCondition, ?c);
8      s ← 3db.select(c, swrl:argument1, ?s);
9      p ← 3db.select(c, swrl:property, ?p);
10     o ← 3db.select(c, swrl:argument2, ?o);
11     s' ← instanciateVariableParameter(s,M);
12     o' ← instanciateVariableParameter(o,M);
13     query ← expressSPARQLQuery(s', p, o');
14     Enregistrement de la requête query;
15     tant que vrai faire
16         R ← réception d'une réponse à la requête query;
17         τ ← 3db.createDoc();
18         3db.addImport(τ, F);
19         3db.addImport(τ, A);
20         si R = ∅ alors
21             Δ_τ = 1000;
22             3db.write(τ, c, valid:Verified, false);
23         sinon
24             Δ_τ = 0;
25             3db.write(τ, c, valid:Verified, true);
26             writeParameterMatch(τ, M, R);
27         fin
28         setInterpretation(τ, Δ_τ);
29     fin
30 fin
```

Fonction `matchIndividualSet`(A: ensemble de ressources, F: ensemble de ressources) $\rightarrow$ $Matches$: ensemble de correspondances

Entrées :
un ensemble de ressources $A = \{\underline{a}_1, \ldots, \underline{a}_n\}$ issues d'une description de fonctionnalité abstraite,
un ensemble de ressources $F = \{\underline{f}_1, \ldots, \underline{f}_m\}$ issues d'une description de fonctionnalité réelle
Sorties : un ensemble de correspondances $Matches = \{(\underline{a}, \underline{f}, cost)\}$, où $cost$ est le degré de correspondance entre les ressources $\underline{a}$ et $\underline{f}$.

```
1  début
2      pour tous les a ∈ A faire
3          cost_a ← 1000;
4          match_a ← null;
5          pour tous les f ∈ F faire
6              cost_{a,f} ← matchIndividual(a, f);
7              si cost_{a,f} < cost_a alors
8                  cost_a ← cost_{a,f};
9                  match_a ← f;
10             fin
11         fin
12         Ajouter (a, match_a, cost_a) à Matches;
13     fin
14     retourner Matches
15 fin
```

éléments de chacun des ensembles. Elle met ainsi en correspondance les couples pour lesquels le degré de correspondance est le plus faible. Ici, cette fonction est utilisé pour déterminer si des paramètres de contexte de la fonctionnalité abstraite correspondent aux paramètres de contexte de la fonctionnalité réelle. En particulier, lorsque les conditions de contexte comprennent des variables, on peut instancier ces variables avec les paramètres de contexte définis dans la description de fonctionnalité abstraite.

L'algorithme effectue ensuite l'instanciation des paramètres variables à l'aide une fonction `instanciateVariableParameter`(). Dans l'expression de la condition, le sujet $\underline{s}$ et l'objet $\underline{o}$ peuvent être de type swrl:Variable. Dans ce cas où l'un de ces paramètres de contexte est une variable, la fonction `instanciateVariableParameter`() exploite l'ensemble de correspondances fournies par la fonction `matchIndividualSet`() pour tenter de remplacer cette variable par un paramètre de contexte non variable.

À partir des informations contenues dans la description, l'algorithme construit ensuite une requête SPARQL. Cette requête est destinée à être prise en charge par le registre d'intérêts sur le contexte, dont le fonctionnement est détaillé dans la section 5.3.2. La fonction `expressSPARQLQuery`() indique comment une requête est formulée à partir du triplet ($\underline{s'}$ $\underline{p}$ $\underline{o'}$). Plusieurs cas peuvent se présenter :

- si aucun paramètre n'est une variable, la requête SPARQL est de la forme `ASK { `$\underline{s'}$ $\underline{p}$ $\underline{o'}$` }`.
- si le sujet est une variable, la requête SPARQL est de la forme `SELECT ?s WHERE {?s `$\underline{p}$ $\underline{o'}$` }`.
- si l'objet est une variable, la requête SPARQL est de la forme `SELECT ?o WHERE { `$\underline{s'}$ $\underline{p}$`?o }`.
- si le sujet et l'objet sont des variables, la requête SPARQL est de la forme `SELECT ?s, ?o WHERE {?s `$\underline{p}$`?o }`.

La requête est ensuite soumise au registre d'intérêts sur le contexte. La réponse à cette requête est traitée en boucle : à chaque modification du contexte pertinent, le registre d'intérêts sur le contexte fournit une nouvelle réponse. La réponse R du registre d'intérêts sur le contexte est un ensemble de résultats (instanciation des variables). Pour simplifier, on ne considère pas ici d'évaluation de la confiance des résultats.

La dernière étape consiste à écrire les résultats dans un document d'interprétation. Dans le cas où le résultat R est vide, on considère que la condition n'est pas vérifiée et le coût est élevé. Dans le cas contraire, on considère que la condition est vérifiée, et on indique dans le document les valeurs appropriées pour les paramètres variables. La fonction `writeParameterMatch`() permet d'écrire les correspondance entre les paramètres dans le document $\underline{\tau}$, en utilisant à la fois les correspondances obtenues par la fonction `matchIndividualSet`() et les résultats de la requête.

6.1.3 Interprétation de correspondance de services

Dans un environnement attentif, les fonctionnalités ne sont pas nécessairement conçues pour fonctionner ensemble. Les services fournis et attendus par deux fonctionnalités ne sont pas nécessairement compatibles. Lors de la composition, il faut ainsi vérifier si deux fonctionnalités sont effectivement capables d'interagir. Cette section présente un mécanisme d'interprétation simple qui vérifie que deux fonctionnalités peuvent interagir par l'intermédiaire des services qu'elles fournissent et attendent.

a) Éléments de description considérés

Pour ce mécanisme simple, les éléments de descriptions considérés sont les services fournis et attendus par deux fonctionnalités, et plus particulièrement les types de ces services. Lorsque l'une des fonctionnalités fournit un service de même type qu'un service attendu par l'autre fonctionnalité, on considère qu'une interaction est possible

entre ces services. Aucun modèle de description particulier n'est donc nécessaire pour ce mécanisme d'interprétation.

La figure 6.9 illustre les éléments de description considérés dans le cas des fonctionnalités iRider et cadre photo. Ici, le service irider:otif_service attendu par iRider et le service pframe:display_service fourni par le cadre photo ont exactement le même type.

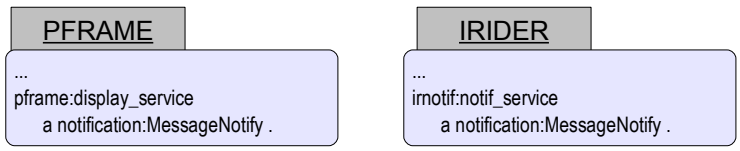

FIGURE 6.9 – Éléments de description des types de services pour les fonctionnalités iRider et cadre photo

b) Modèle de description de l'interprétation

La description de l'interprétation consiste principalement à décrire le connecteur qui permet aux fonctionnalités d'interagir. Cette description s'appuie sur le modèle de description présenté dans la section 5.2.1 (figure 5.8).

La figure 6.10 illustre ainsi la description produite dans le cas des fonctionnalités iRider et cadre photo. Cette description définit simplement un connecteur entre le service irider:notif_service attendu par iRider et le service pframe:display_service fourni par le cadre photo.

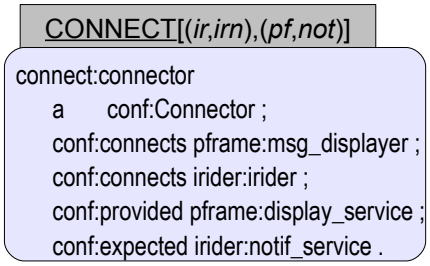

FIGURE 6.10 – Description de l'interprétation de capacité d'interaction entre les fonctionnalités iRider et cadre photo

c) Algorithme d'interprétation

L'algorithme d'interprétation de la correspondance de services `InterpretServiceMatching` (page 122) repose sur une fonction `matchServices`(), qui détermine les paires de services fournis et attendus qui se correspondent. Cette fonction compare les types des services fournis par la fonctionnalité $\underline{p}$ et les services attendus par la fonctionnalité $\underline{e}$ à l'aide de la fonction `matchIndividualSet`() qui permet de déterminer la correspondance entre deux ensembles de ressources (section 6.1.1). On considère qu'il est possible de mettre en place une connexion entre ces fonctionnalités lorsque les types du service fourni et du service attendu sont égaux.

À l'aide de cette fonction, l'algorithme d'interprétation `InterpretServiceMatching` consiste à tester les possibilités d'interaction dans les deux directions possibles, car on ne sait pas *a priori* quelle fonctionnalité est susceptible de fournir un service

Fonction `matchServices`

Entrées :
une ressource $\underline{p}$ décrivant une fonctionnalité pour laquelle on considère les services fournis,
une ressource $\underline{e}$ décrivant une fonctionnalité pour laquelle on considère les services attendus
Sorties : un ensemble de connexions possibles entre les services fournis et attendus

```
1  début
2      S_p ← 3db.selectAll(p, fonc:provides, ?s);
3      S_e ← 3db.selectAll(e, fonc:expects, ?s);
4      M ← matchIndividualSet(S_e, S_p);
5      pour tous les (s_e, s_p, cost) ∈ M faire
6          si cost = 0 alors
7              Ajouter (s_e, s_p) à Result;
8          fin
9      fin
10     return Result;
11 fin
```

attendu par l'autre. Le coût résultant est le minimum des coûts obtenus dans ces deux cas, et le document d'interprétation résultant est formé en combinant les documents fournis par la fonction `matchServices`().

6.2 Mécanisme de résolution

Les algorithmes d'interprétation tels que présentés dans la section précédente permettent d'enrichir progressivement les descriptions fournies et d'obtenir une description de la configuration adaptée. Cependant, parmi toutes les interprétations possibles, seules quelques-unes permettent de d'obtenir des configurations adaptées à la situation.

Cette section présente la modélisation de la recherche d'une configuration adaptée sous la forme d'un problème de satisfaction de contraintes (CSP, *Constraints Satisfaction Problem*). Comme souligné dans l'état de l'art (notamment section 3.3.1) le cadre des problèmes de satisfaction de contraintes fournit une modélisation générique bien adaptée aux problèmes réels et simplifie l'implication des utilisateurs.

La section 6.2.1 pose tout d'abord les définitions et notations pour les réseaux de contraintes. La section 6.2.2 propose ensuite une formulation de la recherche d'une configuration adaptée sous la forme d'un CSP, puis la section 6.2.3 étend cette modélisation pour prendre en compte l'évolution. Enfin, La section 6.2.4 détaille comment la description d'une configuration adaptée est obtenue à partir des solutions du CSP.

6.2.1 Réseau de contraintes : définition et notations

Définition 11 (Réseau de contraintes) *Un réseau de contraintes est défini par un triplet $R = (X, V, C)$, où :*

- $X = \{x_1, \ldots, x_n\}$ *est un ensemble de* n variables.
- V *est un ensemble de* valeurs. *Chaque variable $x_i \in X$ peut recevoir toute valeur $v \in V$.* [1]

1. Dans cette définition, on ne distingue pas un domaine de valeur V_i différent pour chaque

Algorithme 7 : `InterpretServiceMatching`

Données :
deux descriptions de fonctionnalités abstraites $(\underline{a_1}, \underline{A_1})$ et $(\underline{a_2}, \underline{A_2})$
deux descriptions de fonctionnalités réelles $(\underline{f_1}, \underline{F_1})$ et $(\underline{f_2}, \underline{F_2})$
Résultat : un document $\underline{\tau}$ décrivant l'interprétation, un coût Δ_τ

1 **début**
2 `3db.load`($\underline{A_1}$), `3db.load`($\underline{A_2}$);
3 `3db.load`($\underline{F_1}$), `3db.load`($\underline{F_2}$);
4 $M_{1,2} \leftarrow$ `matchServices`($\underline{f_1}$, $\underline{f_2}$);
5 $M_{2,1} \leftarrow$ `matchServices`($\underline{f_2}$, $\underline{f_1}$);
6 $\underline{\tau} \leftarrow$ `3db.createDoc`();
7 `3db.addImport`($\underline{\tau}$, $\underline{F_1}$), `3db.addImport`($\underline{\tau}$, $\underline{F_2}$);
8 `3db.addImport`($\underline{\tau}$, $\underline{A_1}$), `3db.addImport`($\underline{\tau}$, $\underline{A_2}$);
9 **si** $M_{1,2} = \emptyset$ et $M_{2,1} = \emptyset$ **alors**
10 $\Delta_\tau = 1000$;
11 **sinon**
12 $\Delta_\tau =$ coût minimal dans $M_{1,2}$ et $M_{2,1}$;
13 **pour tous les** $(\underline{s_e}, \underline{s_p}) \in M_{1,2}$ **faire**
14 `writeConnector`($\underline{\tau}$, $\underline{s_e}$, $\underline{s_p}$);
15 **fin**
16 **pour tous les** $(\underline{s_e}, \underline{s_p}) \in M_{2,1}$ **faire**
17 `writeConnector`($\underline{\tau}$, $\underline{s_e}$, $\underline{s_p}$);
18 **fin**
19 **fin**
20 `setInterpretation`($\underline{\tau}, \Delta_\tau$)
21 **fin**

– *C est un ensemble de* contraintes. *Chaque contrainte $c \in C$ est définie sur un ensemble de variables $X_c \subset X$ par un ensemble Γ_c de fonctions $\gamma_c : X_c \to V$. Chaque γ_c associe une valeur autorisée à chacune des variables de X_c.*

L'attribution d'une valeur aux variables d'un sous-ensemble $U \subseteq X$ est exprimée par une fonction $\gamma : U \to V$, appelée *instanciation*. On dira qu'une variable $x_i \in X$ est *instanciée* dans γ par la valeur $v \in V$ si et seulement si $x_i \in U$ et $\gamma(x_i) = v$. Dans ce cas, on notera aussi l'instanciation sous la forme $(x_i, v) \in \gamma$. On note Γ l'ensemble de toutes les instanciations possibles de X dans V.

Afin de simplifier les notations, on note $[X]$ le tuple des variables tel que $[X] = [x_1, \ldots, x_n]$. $[X_c]$ le tuple des variables d'une contrainte c (en respectant l'ordre des index des variables).

On note $\gamma[U]$ le tuple de valeurs correspondant à une instanciation γ définie sur $U \subset X$. Chaque tuple de valeurs défini par l'application de γ sur un sous-ensemble $T \subset U$, est noté $\gamma[T]$. Dans la suite, on emploie la même notation $\gamma[U]$ pour désigner :
– la fonction $\gamma : U \to V$,
– le tuple de valeurs obtenu par l'application de γ sur $[U]$,
– l'ensemble des couples $\{(x, \gamma(x))\}_{x \in U}$.

On s'autorise à manipuler ces objets comme des ensembles, et à noter par exemple $\forall T \subset U, \gamma[T] \subset \gamma[U]$.

Pour chaque contrainte, l'ensemble des tuples de valeurs autorisées pour une contrainte c est noté $\Gamma_c[X_c]$. Dans le modèle considéré, les contraintes sont définies en *intention* : l'ensemble des tuples autorisés est donné par une fonction caractéristique. On appelle *évaluation* de la contrainte l'application de cette fonction caractéristique à un tuple de valeurs. L'évaluation d'un tuple de valeurs permet donc de déterminer si ce tuple satisfait la contrainte ou non.

Une instanciation γ définie sur $U \subseteq X$ est dite *complète* si $U = X$, et *partielle* sinon. Étant donné une contrainte c, une instanciation γ *satisfait* la contrainte si $\gamma[X_c] \in \Gamma_c[X_c]$ et *viole* la contrainte si $\gamma[X_c] \notin \Gamma_c[X_c]$. Une instanciation est *cohérente* si elle ne viole aucune contrainte de C. Une instanciation *complète* et *cohérente* est une *solution* du réseau de contraintes. La résolution d'un CSP consiste à exhiber les solutions d'un réseau de contrainte, s'il en existe. Un problème d'optimisation de contraintes consiste à déterminer la solution optimale selon un critère particulier.

Graphe de contraintes Étant donnée une contrainte c, le cardinal de X_c, noté $|X_c|$ est appelé *arité* de la contrainte. Dans le cas où l'on considère uniquement des contraintes d'arité inférieure ou égale à 2, le graphe des contraintes $(X, \{X_c\}_{c \in C})$ permet de donner une représentation graphique simple au réseau de contraintes. La figure 6.11 représente un tel graphe de contraintes : les nœuds ronds représentent les variables et les nœuds carrés représentent les contraintes. Chaque nœud est étiqueté par le nom de la variable ou de la contrainte qu'il représente.

variable x_i

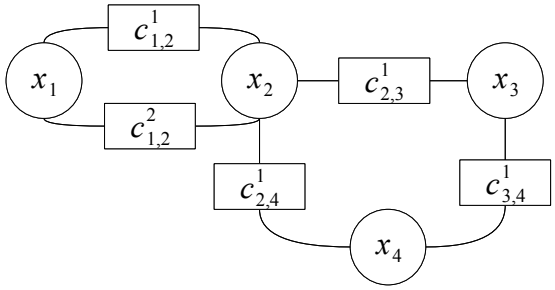

FIGURE 6.11 – Exemple de réseau de contraintes

Dans cette représentation graphique, on désigne chaque contrainte $c \in C$ à l'aide une étiquette choisie en fonction des variables contenues dans X_c de la manière suivante :

- les contraintes telles que $X_c = \{x_i, x_j\}$ où $i, j \in [1..n]$ et $i < j$ sont notés $c^k_{i,j}$ avec $k \in [1..k_{i,j}]$ où $k_{i,j}$ est le nombre total de contraintes portant sur x_i et x_j.
- les contraintes telles que $X_c = \{x_i\}, i \in [1..n]$ sont notés $c^k_{i,i}$ avec $k \in [1..k_{i,i}]$ où $k_{i,i}$ est le nombre total de contraintes portant uniquement sur x_i.

Dans la suite, on considérera uniquement des contraintes d'arité inférieure ou égale à 2.

La représentation graphique permet aussi de représenter une instanciation particulière, dans laquelle une valeur est associée à chaque variable et une évaluation est associée à chaque contrainte. La figure 6.12 représente un état particulier du problème. Dans cet état, la valeur v_{14} est attribuée à x_1, v_{53} est attribuée à x_2 et x_4, et v_{62} est attribuée à x_3. Les contraintes $c^1_{1,2}$ et $c^1_{2,4}$ ne sont pas satisfaites et sont représentées barrées d'une croix.

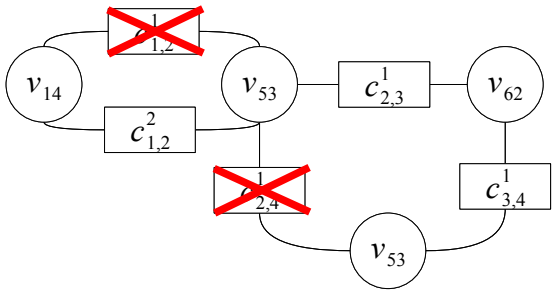

FIGURE 6.12 – Exemple d'état du réseau de contraintes

6.2.2 Formulation de la recherche de configuration adaptée

Pour formuler le réseau de contrainte associé à une application, on considère un espace de description (voir section 5.2.2) qui contient :

- la description de l'application, qui définit les fonctionnalités abstraites qui la composent. Chaque fonctionnalité abstraite possède une description $(\underline{a_i}, \underline{A_i}), i \in [1..N]$, où N est le nombre de fonctionnalités abstraites de l'application.
- des descriptions de fonctionnalités réelles $(\underline{r_j}, \underline{R_j}), j \in [1..M]$, où M est le nombre de fonctionnalités réelles disponibles dans l'environnement.

– des interprétations de descriptions, fournies par des mécanismes d'interprétation τ_l. Chaque algorithme d'interprétation est caractérisé par sa fonction de coût Δ_{τ_l} et son coût limite λ_{τ_l}.

Les besoins d'une application sont modélisés sous la forme d'un réseau de contraintes $R = (X, V, C)$ dans lequel :

– une *variable* a_i est associée à chaque fonctionnalité abstraite décrite par $(\underline{a_i}, \underline{A_i})$. L'ensemble des variables est donc $X = \{a_i\}_{i\in[1..N]}$.
– une *valeur* r_j est associée à chacune des fonctionnalités réelles disponible dans l'environnement. L'ensemble des valeurs est donc $V = \{r_j\}_{j\in[1..M]}$.
– une *contrainte* $c^{\tau}_{X_c}$ est associée à un tuple $(X_c, \Delta_\tau, \lambda_\tau)$, où les éléments du tuple sont les suivants :
  – $X_c \subset X$ est l'ensemble des variables sur lesquelles porte la contrainte. Cet ensemble est aussi associé à un sous-ensemble de descriptions de fonctionnalités abstraites $(\underline{a_i}, \underline{A_i})$.
  – Δ_τ est la fonction de coût associée à un mécanisme d'interprétation de type τ. Elle associe à chaque instanciation $\gamma[X_c] : X_c \rightarrow V$ le coût obtenu lorsque l'algorithme d'interprétation est appliqué aux descriptions de fonctionnalités abstraites et réelles correspondantes.
  – λ_τ est un coût limite, qui définit l'ensemble $\Gamma_c[X_c]$ des tuples autorisés pour c.

Dans cette formulation, la satisfaction d'une contrainte $c\tau_{X_c}$ par une instanciation γ est déterminé par l'intermédiaire de la fonction de coût de l'algorithme d'interprétation.

$$\gamma[X_c] \in \Gamma_c[X_c] \Leftrightarrow \Delta_\tau(\gamma[X_c]) \leq \lambda_\tau$$

a) Caractérisation des contraintes

Après avoir introduit comment les besoins d'une application composite flexible sont modélisés par un réseau de contraintes, cette section décrit les principales caractéristiques des contraintes et permet de préciser comment elles modélisent les différents aspects des applications.

Portée On s'intéressera principalement à des contraintes d'arité 1 ou 2. Chacune de ses arités traduit une réalité particulière pour une application composite flexible :

– une contrainte d'arité $|X_c| = 1$ porte sur une seule variable, c'est-à-dire sur une seule fonctionnalité. Cette contrainte influe donc directement sur la sélection d'une fonctionnalité réelle pour réaliser une fonctionnalité abstraite, indépendamment des autres fonctionnalités intervenant dans l'application. On nommera ce type de contrainte une *contrainte locale*.
– une contrainte d'arité $|X_c| = 2$ porte sur deux variables, c'est-à-dire sur deux fonctionnalités. Ce type de contrainte dénote les interactions entre les fonctionnalités. En particulier, elles permettent d'exprimer les contraintes d'interopérabilité entre des fonctionnalités. On nommera ce type de contrainte une *contrainte partagée*.

Dans la suite, on parlera de *portée* (locale ou partagée) d'une contrainte dans une application.

On notera C^l l'ensemble des contraintes locales et C^s l'ensemble des contraintes partagées. En utilisant la convention d'étiquetage des contraintes défini précédemment (section 6.2.1), C^l contient les contraintes de type $c^k_{i,i}$ et C^s contient les contraintes de type $c^k_{i,j}, i \neq j$. Ces ensembles sont inclus dans C, disjoints ($C^l \cap C^s = \emptyset$) et complémentaires ($C^l \cup C^s = C$).

Évaluation et domaine de satisfaction L'ensemble $\Gamma_c[X_c]$ des tuples de valeurs qui satisfont une contrainte est défini à l'aide du coût et du coût limite associé

à chaque tuple de valeur. La fonction de coût et le coût limite font apparaître trois sous-ensembles parmi les tuples de valeurs possibles :

- l'ensemble $\Gamma_c^0[X_c]$ des tuples dont le coût est nul correspondent aux instanciations qui satisfont *parfaitement* la contrainte.
$$\gamma[X_c] \in \Gamma_c^0[X_c] \Leftrightarrow \Delta_\tau(\gamma[X_c]) = 0$$
- l'ensemble $\Gamma_c^-[X_c]$ des tuples dont le coût est inférieur au coût limite correspondent aux instanciations qui satisfont *suffisamment* la contrainte.
$$\gamma[X_c] \in \Gamma_c^-[X_c] \Leftrightarrow \Delta_\tau(\gamma[X_c]) \leq \lambda_\tau$$
- L'ensemble $\Gamma_c^+[X_c]$ des tuples dont le coût est supérieur au coût limite correspondent aux instanciations qui ne satisfont *pas suffisamment* la contrainte.
$$\gamma[X_c] \in \Gamma_c^+[X_c] \Leftrightarrow \Delta_\tau(\gamma[X_c]) \geq \lambda_\tau$$

  On a alors $\Gamma_c[X_c] = \Gamma_c^-[X_c]$ et $\Gamma_c^0[X_c] \subset \Gamma_c^-[X_c]$.

Il est possible de distinguer les contraintes pour lesquelles $\lambda_\tau = 0$. Dans ce cas, $\Gamma_c^0[X_c] = \Gamma_c^-[X_c]$, et un tuple doit satisfaire parfaitement la contrainte pour être autorisé. On parlera de *contrainte absolue* lorsque $\lambda_\tau = 0$, et de *contrainte souple* lorsque $\lambda_\tau > 0$.

L'utilisation de la fonction de coût et du coût limite apporte une grande flexibilité qui permet de traduire l'imprécision des connaissances sur le besoin. Pour chaque contrainte, la fonction de coût permet d'ordonner les différentes possibilités, tandis que le coût limite permet d'exclure certaines possibilités.

6.2.3 Évolution d'un réseau de contraintes

Les environnements attentifs sont caractérisés par leur dynamicité et par les variations des besoins des utilisateurs. Pour répondre à ces problématiques, le réseau de contraintes qui modélise les besoins d'une application attentive est capable d'évoluer au cours du temps, en fonction des modifications du contexte et des besoins des utilisateurs.

Cette section présente tout d'abord la modélisation de cette évolution, puis introduit plusieurs cas d'évolution d'un réseau de contraintes, illustrés à l'aide d'exemples.

a) Modélisation de l'évolution d'un réseau de contraintes

L'évolution d'un réseau de contrainte est modélisé sous la forme d'une suite de réseaux, indexée par un indice t :

$$R_t = (X_t, V_t, C_t)$$

Chaque réseau R_{t+1} résulte d'une évolution élémentaire du réseau R_t. Les évolutions élémentaires considérées sont :

- l'ajout/retrait d'une valeur
- ajout/retrait de contrainte
- ajout/retrait de variable (et donc des contraintes associées)

Concernant les contraintes, il est aussi nécessaire de prendre en compte deux autres évolution possibles :

- la modification de la fonction de coût Δ_τ
- la modification du coût limite λ_τ

Ces modifications peuvent cependant se ramener au retrait de la contrainte suivi de l'ajout d'une contrainte similaire dans laquelle la fonction de coût ou le coût limite sont modifiés.

Chacune des évolutions élémentaires affecte la structure du réseau de manière locale. Ainsi, il est souvent possible de déterminer les solutions du problème associé à un réseau de contraintes R_{t+1} en partant des solutions du problème associé à R_t.

b) Caractérisation de l'évolution du réseau de contraintes

Cette modélisation de l'évolution d'un réseau de contraintes permet de rendre compte de diverses possibilités d'évolution d'une application attentive.

Apparition/disparition d'une fonctionnalité Le cas le plus simple est l'apparition/disparition d'une fonctionnalité. Par exemple, le cadre photo utilisé pour notifier Thom peut être éteint au cours du fonctionnement de l'application. Inversement, un nouveau dispositif de notification peut être installé dans l'environnement.

L'apparition/disparition de fonctionnalités est modélisée simplement par l'ajout/retrait d'une variable du réseau de contraintes. Plusieurs cas peuvent alors se présenter :

- une fonctionnalité qui n'était pas utilisée disparaît : il ne se passe rien.
- une fonctionnalité qui était utilisée disparaît : le système doit rechercher d'autres solutions du problème de satisfaction de contraintes, qui étaient peut être moins intéressantes précédemment.
- une fonctionnalité apparaît mais ne permet pas de trouver une composition plus appropriée : il ne se passe rien
- une fonctionnalité apparaît et permet de trouver une composition plus appropriée : le système met à jour la configuration de l'application pour utiliser la nouvelle fonctionnalité.

Évolution d'une information sur le contexte Le cas le plus courant est l'évolution d'une information sur le contexte. Par exemple, lorsque Thom se déplace dans une autre pièce, le cadre photo n'est plus le dispositif de notification le plus approprié.

L'évolution d'une information sur le contexte est modélisée par la modification de la fonction de coût Δ_τ, pour les mécanismes d'interprétation de type τ qui sont influencés par cette évolution. Cette modification impacte les contraintes associée à ces mécanismes d'interprétation, et donne lieu à une nouvelle résolution du CSP.

Ajustement dynamique de préférences Il est possible de laisser un utilisateur ajuster ses préférences au cours du fonctionnement de l'application. Par exemple, Thom peut décider de ne pas imposer d'adaptation au contexte pour le choix de dispositif de notification.

L'ajustement de telles préférences est modélisé par une modification du coût limite λ_τ pour les mécanismes d'interprétation concernés. Comme dans le cas précédent, cette modification impacte les contraintes de type τ, et nécessite une nouvelle résolution du CSP.

Modification de l'application abstraite Au cours du fonctionnement de l'application, le besoin de l'utilisateur peut être légèrement modifié. Par exemple, Thom peut souhaiter recevoir des notifications vocales plutôt que textuelles.

La modification du besoin de l'utilisateur se traduit par une modification de la description abstraite de l'application. Une telle modification impacte les variables et les contraintes du CSP, et nécessite une nouvelle résolution.

6.2.4 Obtention de la configuration adaptée

Après avoir présenté le réseau de contrainte associé à une application, cette section présente comment la résolution du CSP associé à une application permet la génération d'une configuration adaptée pour la situation de l'environnement.

Cette section précise tout d'abord les critères selon lesquels une configuration est appropriée est choisie, en particulier pour les cas où plusieurs solutions du CSP

existent. Elle détaille comment construire la description de la configuration correspondant à une solution du CSP.

a) Critère de choix de la configuration adaptée

Dans l'approche proposée, la configuration adaptée est la configuration obtenue à partir de la meilleure solution du CSP associé à une application. Cette section définit plus précisément les critères de choix de la meilleure solution du CSP.

On définit l'ensemble $\Gamma[X]$ des instanciations satisfaisantes pour le CSP comme l'ensemble des instanciations satisfaisantes pour chacune des contraintes $c \in C$:

$$\gamma[X] \in \Gamma[X] \Leftrightarrow \forall c \in C, \gamma[X_c] \in \Gamma_c[X_c]$$

Plusieurs cas peuvent se présenter :
- Si $\Gamma[X]$ ne contient aucun élément, il n'existe aucune solution au CSP.
- Si $\Gamma[X]$ contient un seul élément, c'est l'unique solution du CSP.
- Si $\Gamma[X]$ contient plusieurs éléments, il existe plusieurs solutions. Dans ce cas, on cherche à comparer les solutions pour choisir les meilleures.

Afin de comparer différentes solutions possibles, on définit un ordre partiel sur $\Gamma[X]$. Pour cela, on considère la fonction

$$\begin{array}{rccc} M: & \Gamma[X] \times \Gamma[X] & \rightarrow & P(C) \\ & (\gamma_1, \gamma_2) & \rightarrow & \{c \in C | \Delta_c(\gamma_1[X_c]) > \Delta_c(\gamma_2[X_c])\} \end{array}$$

$M(\gamma_1, \gamma_2)$ est l'ensemble des contraintes pour lesquelles γ_1 a un coût plus élevé que γ_2. Le cardinal de cet ensemble, $|M(\gamma_1, \gamma_2)|$ est le nombre de contraintes pour lesquelles γ_1 a un coût plus élevé que γ_2. On considère de plus la fonction $S(\gamma_1, \gamma_2) = \sum_{c \in M(\gamma_1, \gamma 2)} \Delta_c(\gamma_1[X_c]) - \Delta_c(\gamma_2[X_c])$. Cette fonction caractérise l'écart global de coût entre les deux instanciations. Par définition, on a toujours $S(\gamma_1, \gamma_2) > 0$.

À l'aide de ces fonctions, deux instanciations sont ordonnées de la manière suivante :

$$\gamma_1[X] > \gamma_2[X] \quad \Leftrightarrow \quad \begin{array}{c} |M(\gamma_1, \gamma_2)| > |M(\gamma_2, \gamma_1)| \\ OU \\ |M(\gamma_1, \gamma_2)| = |M(\gamma_2, \gamma_1)| \text{ ET } S(\gamma_1, \gamma_2) \geq S(\gamma_1, \gamma_2) \end{array}$$

Cela signifie que l'instanciation γ_1 est considérée comme plus coûteuse que l'instanciation γ_2 s'il existe plus de contraintes pour lesquelles le coût de γ_1 est plus élevé que celui de γ_2, ou bien s'il en existe autant mais que le coût global de γ_1 est plus élevé que celui de γ_2. Notons que cet ordre privilégie une comparaison contrainte par contrainte par rapport à une comparaison globale basée sur la somme de l'ensemble des coûts des contraintes : une telle somme n'a en effet que peu de sens car il n'est pas possible de garantir une certaine uniformité dans l'ordre de grandeur des coûts des contraintes (ce qui conduit à des aberrations pour une instanciation pour laquelle une seule contrainte à un coût beaucoup plus élevé que les autres). Notons par ailleurs qu'il ne s'agit pas d'un ordre total. Dans le cas où $|M(\gamma_1, \gamma_2)| = |M(\gamma_2, \gamma_1)|$ et $S(\gamma_1, \gamma_2) = S(\gamma_1, \gamma_2)$, on a à la fois $\gamma_1[X] > \gamma_2[X]$ et $\gamma_1[X] > \gamma_2[X]$, sans avoir $\gamma_1[X] = \gamma_2[X]$.

À l'aide de cet ordre, on détermine l'ensemble $\Gamma^-[X]$ des instanciations de $\Gamma[X]$ dont le coût est le plus faible

$$\Gamma^-[X] = \{\gamma[X] \in \Gamma[X] | \forall \gamma'[X] \in \Gamma[X], \gamma'[X] > \gamma[X]\}$$

L'ensemble $\Gamma^-[X]$ est l'ensemble des meilleures solutions de $\Gamma[X]$. Il peut contenir plusieurs éléments : dans ce cas, une discrimination plus fine est laissée au contrôle des utilisateurs du système.

Dans certains cas, il est intéressant de considérer le sous-ensemble $\Gamma^0[X] \subset \Gamma[X]$ des solutions parfaites, pour lesquelles le coût de toutes les contraintes est nul :

$$\Gamma^0[X] = \{\gamma[X] \in \Gamma[X] | \forall c \in C, \gamma[X_c] \in \Gamma^0_c[X_c]\}$$

Dans le cas où $\Gamma^0[X] \neq \emptyset$, on a alors simplement $\Gamma^-[X] = \Gamma^0[X]$. Comme précédemment, si $\Gamma^0[X]$ contient plusieurs éléments, il n'est pas possible de discriminer une solution optimale du problème.

b) Construction de la description d'une configuration adaptée

À partir d'une solution du CSP associé à une application, on construit la description d'une configuration adaptée. Cette description est obtenue en combinant plusieurs descriptions :

- la description abstraite de l'application (et donc les descriptions des fonctionnalités abstraites).
- les descriptions des fonctionnalités réelles choisies.
- les interprétations de fonctionnalités correspondant aux contraintes du CSP.

On considère une solution γ^0 du CSP (X, V, C). Pour simplifier les notations, on note chacune des valeurs choisies en utilisant le même indice que la variable qu'elle instancie. Ainsi, $\$\forall x_i \in X, \gamma^0(x_i)$ est noté v_i. La description de la fonctionnalité réelle correspondant à cette valeur est notée $f_i = (\underline{f_i}, \underline{F_i})$.

Pour chaque contrainte, on doit considérer l'interprétation correspondant aux valeurs choisies. Ainsi, pour une contrainte d'arité 1 définie par $(X_c = \{x_i\}, \Delta_\tau, \lambda_\tau)$, l'interprétation correspondante est $\underline{\tau}[(f_i, a_i)]$. Pour une contrainte d'arité 2 définie par $(X_c = \{x_i, x_j\}, \Delta_\tau, \lambda_\tau)$, l'interprétation correspondante est $\underline{\tau}[(f_i, a_i), (f_j, a_j),]$. On note T_i l'ensemble des interprétations portant sur une instanciation $[(f_i, a_i)]$ et $T_{i,j}$ l'ensemble des interprétations portant sur une instanciation $[(f_i, a_i), (f_j, a_j)]$.

Pour chaque fonctionnalité choisie, on obtient la *description contextualisée* de la fonctionnalité en combinant les interprétations portant sur cette fonctionnalité, c'est-à-dire en créant une nouvelle description dont le document support importe toutes ces interprétations. La description contextualisée d'une fonctionnalité f_i est ainsi :

$$f'_i = (\underline{f}_i, \underline{F'_i}) \text{ avec } Import(\underline{F'_i}) = \bigcup_{T_i} \underline{\tau}[(f_i, a_i)]$$

Pour chaque couple de fonctionnalités en interaction choisie, on obtient la *description de connexion* du couple de fonctionnalités en combinant les interprétations portant sur ce couple de fonctionnalités. La description contextualisée d'un couple de fonctionnalités (f_i, f_j) est ainsi :

$$c_{i,j} = (\underline{c}_{i,j}, \underline{C}_{i,j}) \text{ avec } Import(\underline{C}_{i,j}) = \bigcup_{T_{i,j}} \underline{\tau}[(f_i, a_i), (f_j, a_j)]$$

Finalement, pour une application dont la description est notée $a = (\underline{a}, \underline{A})$, on obtient la description de la configuration adaptée en combinant les descriptions contextualisées des fonctionnalités et les descriptions de connexion des couples de fonctionnalités. Cette description est ainsi :

$$a' = (\underline{a}, \underline{A'}) \text{ avec } Import(\underline{A'}) = \bigcup_i \underline{F'_i} \cup \bigcup_{i,j} \underline{C}_{i,j}$$

6.3 Synthèse

Ce chapitre a présenté les mécanismes flexibles de gestion de composition. Ces mécanismes se divisent en deux groupes : les mécanismes d'interprétation et le mécanisme de résolution. Un mécanisme d'interprétation est consacré à l'interprétation

spécifique de certains éléments de descriptions de fonctionnalités. Il fournit ainsi une expertise sur un aspect particulier des fonctionnalités et sur le rôle de cet aspect dans la composition. Le mécanisme de résolution est chargé de déterminer la description d'une configuration adaptée pour l'application dans une situation donnée. Ce mécanisme repose sur la définition d'un problème de satisfaction de contraintes associé à une application. Ce problème de satisfaction de contraintes intègre les différentes contributions des mécanismes d'interprétation pour élaborer une solution globale.

Les mécanismes de gestion de composition présentent deux caractéristiques principales :

L'indépendance entre les différents mécanismes est fondamentale. D'une part, chacun des mécanismes d'interprétation de description est indépendant des autres et offre un point de vue particulier sur les descriptions considérées. D'autre part, le mécanisme de résolution agrège ces différentes interprétations de manière à fournir une solution cohérente.

L'évolution des interactions entre les différents mécanismes est possible. En effet, il n'existe pas un unique algorithme de composition capable de déterminer la configuration adaptée dans tout les cas. Selon les situations, les différents mécanismes de gestion de composition doivent être organisés et articulés différemment pour rendre compte des évolutions de l'environnement.

En conséquence de ces caractéristiques, il reste à définir une architecture permettant d'organiser ces différents mécanismes de manière flexible au sein d'un système de gestion de composition. L'architecture d'un système de gestion de composition doit préserver les caractéristiques d'indépendance des mécanismes d'interprétation et la possibilité d'évolution du problème de contraintes en fonction de la situation. Le chapitre suivant propose une architecture multi-agents pour les systèmes de gestion de composition. L'utilisation d'une approche multi-agents garantit ces propriétés et apporte ainsi une réponse appropriée aux problématiques des applications attentives.

Chapitre 7

Architecture multi-agents d'un système de gestion de composition flexible

Le chapitre 6 décrit les différents mécanismes qui permettent à un système de gestion de composition de déterminer dynamiquement une configuration adaptée pour une application. Une des caractéristiques fondamentales de ces mécanismes est la possibilité d'évolution du problème de satisfaction de contraintes associé à une application, qui permet de refléter les évolutions de l'environnement et des besoins des utilisateurs. Cette flexibilité est nécessaire pour prendre en compte la dynamicité, l'ouverture et l'imprécision des besoins dans les environnements attentifs. Le système de gestion de composition adapte ainsi constamment l'assemblage de fonctionnalité utilisé suivant les conditions rencontrées.

Ce chapitre détaille l'architecture d'un système de gestion de composition FCAP. Cette architecture est basée sur un système multi-agents dans lequel les différents agents réalisent les différents mécanismes présentés précédemment. L'utilisation d'une approche multi-agents garantit l'indépendance du fonctionnement des différents mécanismes de gestion de composition et la possibilité d'organiser les interactions entre ces différents mécanismes de manière flexible. Les capacités d'évolution et de réorganisation d'un système multi-agents sont essentielles pour répondre efficacement aux problématiques des environnements attentifs, en particulier concernant la dynamicité et l'évolution de l'environnement et des besoins des utilisateurs.

La section 7.1 présente l'architecture générale du système multi-agents, en détaillant les différents types d'agents et leurs interactions. La section 7.2 détaille le fonctionnement des agents de résolutions, chargés de mettre en place une configuration adaptée en résolvant le problème de contraintes. La section 7.3 présente le fonctionnement des agents d'interprétation, qui fournissent au système les mécanismes d'interprétation de description grâce auxquels une configuration adaptée peut être déterminée.

7.1 Système de gestion de composition à base d'agents

Cette section présente le système multi-agents qui fournit un mécanisme flexible de résolution du problème de contraintes associé à une application. Dans cette approche,

les variables et les contraintes du problème sont représentées par des agents. Les interactions entre ces agents, et donc l'organisation du réseau de contraintes, sont mises en place dynamiquement.

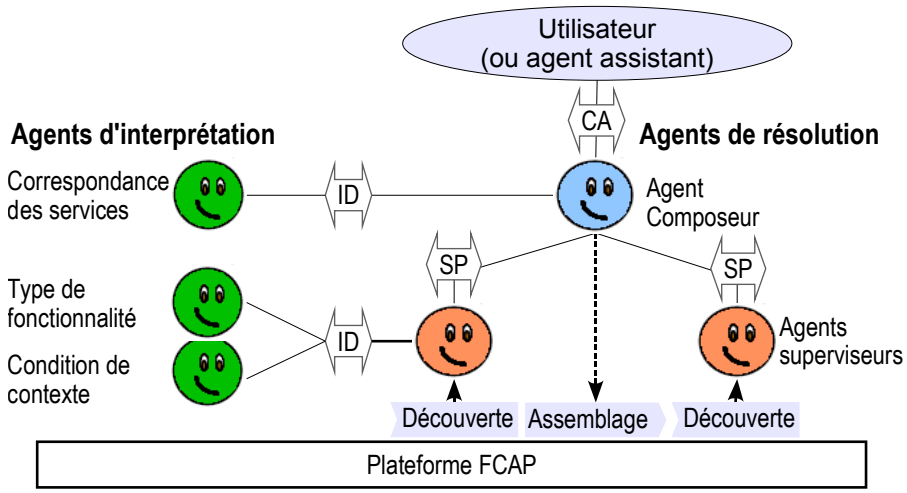

FIGURE 7.1 – Exemple d'architecture d'un système de gestion de composition

La figure 7.1 illustre l'architecture générale d'un système de gestion de composition, dont les différents constituants sont décrits dans les sections suivantes. La section 7.1.1 introduit les différents types d'agents présents dans un système de gestion de composition. La section 7.1.2 détaille les protocoles d'interaction grâce auxquels les différents agents interagissent et mettent en place l'organisation du système.

7.1.1 Types d'agents du système de gestion de composition

La figure 7.1 présente l'architecture globale d'un système multi-agents chargé de gérer la composition d'une application. Cette architecture fait intervenir deux catégories d'agents : les agents de résolution et les agents d'interprétation.

Les agents de résolution ont pour mission la résolution du problème de satisfaction de contraintes (CSP) associé à une application. Pour chaque application composite flexible, il existe une équipe d'agents de résolution. Cette catégorie est composée de deux types d'agents :

Agent composeur d'application Un agent composeur possède une vue globale de l'application. Il est chargé de mettre en place une configuration correspondant à la description abstraite de l'application qui lui est fournie. À l'aide de cette description, il détermine la structure globale de l'application et organise le réseau de contraintes correspondant. Pour chacune des fonctionnalités abstraites de l'application, il charge un agent superviseur de lui fournir des descriptions appropriées. À partir de cet ensemble, il termine la résolution du CSP et choisit la configuration à mettre en place. La section 7.2.3 décrit le fonctionnement d'un agent composeur plus en détail.

Agent superviseur de fonctionnalité Un agent superviseur possède une vue locale sur l'une des fonctionnalités abstraites de l'application. Il est chargé de

fournir à l'agent composeur des descriptions de fonctionnalités appropriées pour réaliser la fonctionnalité voulue. Pour cela, il doit non seulement sélectionner les meilleures fonctionnalités présentes dans l'environnement, mais il doit aussi fournir une description permettant d'intégrer ces fonctionnalités dans l'application. Sa tâche se décompose en trois aspects : découvrir des descriptions de fonctionnalités, obtenir des interprétations des descriptions et sélectionner les fonctionnalités appropriées. Les interprétations de descriptions sont obtenues en interrogeant différents agents d'interprétation présents dans le système. Les trois tâches sont menées de manière continue et réactive : l'agent s'adapte à l'évolution de l'environnement et maintient un ensemble de descriptions valides dans le contexte courant. La section 7.2.2 décrit le fonctionnement d'un agent superviseur plus en détail.

Les agents d'interprétation Un agent d'interprétation (ou interpréteur) possède une connaissance spécifique sur un aspect de description des fonctionnalités. Il repose sur un mécanisme d'interprétation (section 6.1) qui lui permet de fournir des informations sur les fonctionnalités. Un agent d'interprétation a la capacité d'interagir avec les agents de résolution, de collecter les informations nécessaires à l'interprétation et de réagir aux modifications de l'environnement. Il se charge ainsi de mettre en œuvre le mécanisme d'interprétation à chaque fois qu'une interprétation doit être produite ou révisée. Les informations fournies par un agent d'interprétation prennent la forme de documents d'interprétation, tels que définis dans la section 5.2.2. La section 7.3 décrit les propriétés génériques des agents d'interprétation et détaille plusieurs réalisations typiques. Dans la figure 7.1, trois agents d'interprétation sont représentés à titre d'exemple :

L'interpréteur de types de fonctionnalité repose sur le mécanisme d'interprétation des types de fonctionnalité présenté dans la section 6.1.1.

L'interpréteur de conditions de contexte repose sur le mécanisme d'interprétation des conditions sur le contexte présenté dans la section 6.1.2.

Interpréteur de correspondance des services repose sur le mécanisme d'interprétation de la correspondance entre les services présenté dans la section 6.1.3.

Ces différents agents sont détaillés dans la section 7.3.2. D'autres exemples d'agents d'interprétation sont détaillés dans le chapitre 8.

Dans cette architecture, on peut noter une distinction fondamentale entre les deux catégories d'agents :

- Les agents de résolution sont des *agents génériques*. Ce sont des agents dont le comportement est défini indépendamment de l'application et de l'environnement. Leur rôle est d'assurer la résolution du problème de satisfaction de contraintes. Ils sont capables de manipuler n'importe quelle description d'application ou de fonctionnalité exprimée à l'aide des modèles de description définis par FCAP (section 5.2.1). En revanche, ils ne peuvent pas interpréter seuls le contenu précis des descriptions, exprimé à l'aide d'ontologies particulières.
- Les agents d'interprétation sont des *agents spécifiques à un environnement*. Il en existe plusieurs types, chacun étant spécialisé dans l'interprétation d'une caractéristique particulière. Le comportement du système dans un environnement particulier dépend des agents d'interprétation qui s'y trouvent. En revanche, ces agents sont généralement indépendants d'une application.

7.1.2 Protocoles d'interaction

Cette section définit plus précisément les protocoles d'interactions qui apparaissent sur la figure 7.1. Chacun de ces protocoles est détaillé sous la forme d'un diagramme de

séquence qui indique les différents messages échangés entre les parties. Les messages sont exprimés à l'aide des langages FIPA-ACL et FIPA-SL (voir section 3.2.3). La signification des messages est décrite ici de manière informelle, mais leur contenu précis est explicité dans la section 7.2.1.

Les interactions entre ces différents types d'agents sont régies par quatre protocoles d'interaction principaux :

`ComposeApplication` (noté `CA` sur la figure 7.1) est le protocole qui permet à un utilisateur (ou à un agent assistant qui le représente dans le système) d'interagir avec un agent composeur. Ce protocole permet de définir la tâche de composition de l'application.

`SuperviseFunctionality` (noté `SP` sur la figure 7.1) est le protocole qui permet à un agent composeur et à un agent superviseur d'échanger des informations sur la tâche de supervision d'une fonctionnalité abstraite.

`InterpretDescription` (noté `ID` sur la figure 7.1) est le protocole qui permet à un agent superviseur et à un agent d'interprétation d'échanger des informations sur une interprétation de fonctionnalités.

`DescribeSolution` (non représenté sur la figure 7.1) est le protocole qui permet à l'ensemble des agents de construire collectivement une description de la configuration adaptée à partir des solutions du problème de contraintes.

Le reste de cette section décrit les quatre protocoles d'interaction.

a) Le protocole de composition d'application

Le protocole `ComposeApplication` implique un agent assistant (extérieur au système) et un agent composeur d'application. Son déroulement est représenté par la figure 7.2 :

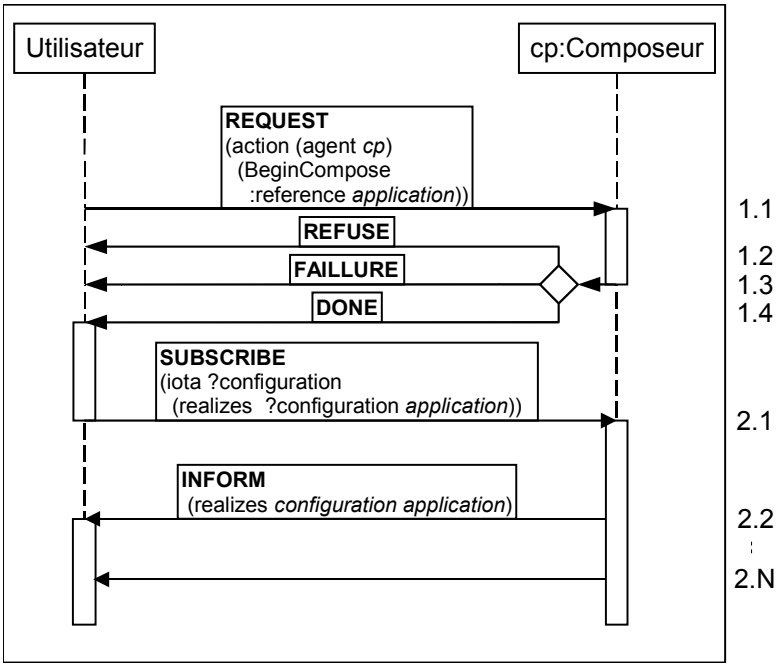

FIGURE 7.2 – Protocole d'interaction `ComposeApplication`

Ce protocole se déroule en deux phases :

1. *Affectation d'une application* : au cours de cette première phase, l'assistant demande au composeur de prendre en charge la composition d'une application. Pour ce faire, il lui fournit la description abstraite de l'application à réaliser (1.1). Le composeur examine cette description et réalise les opérations d'initialisation pour la recherche de solutions. Il informe ensuite l'assistant du succès (1.4) ou de l'échec (1.3) de l'initialisation. Le composeur peut aussi refuser de réaliser cette composition (1.2), par exemple s'il est déjà occupé à en réaliser une autre. En cas de succès de l'initialisation, le composeur prend l'engagement de poursuivre l'activité de composition de l'application.
2. *Surveillance des résultats* : tout au long du fonctionnement de l'application, l'assistant peut surveiller les configurations mises en place pour la réaliser. Pour cela, il s'inscrit auprès du composeur de manière à recevoir les descriptions de configurations qui réalisent l'application demandée (2.1). A chaque modification de configuration, le composeur fournit une description de la configuration choisie pour réaliser le comportement souhaité de manière adaptée à la situation (2.2 ... 2.N). La description de configuration peut éventuellement être vide, ce qui traduit l'impossibilité de réaliser l'application dans la situation courante.

b) Le protocole de supervision de fonctionnalité

Le protocole `SuperviseFunctionality` concerne un agent composeur et un agent superviseur. Son déroulement est représenté par la figure 7.3 :

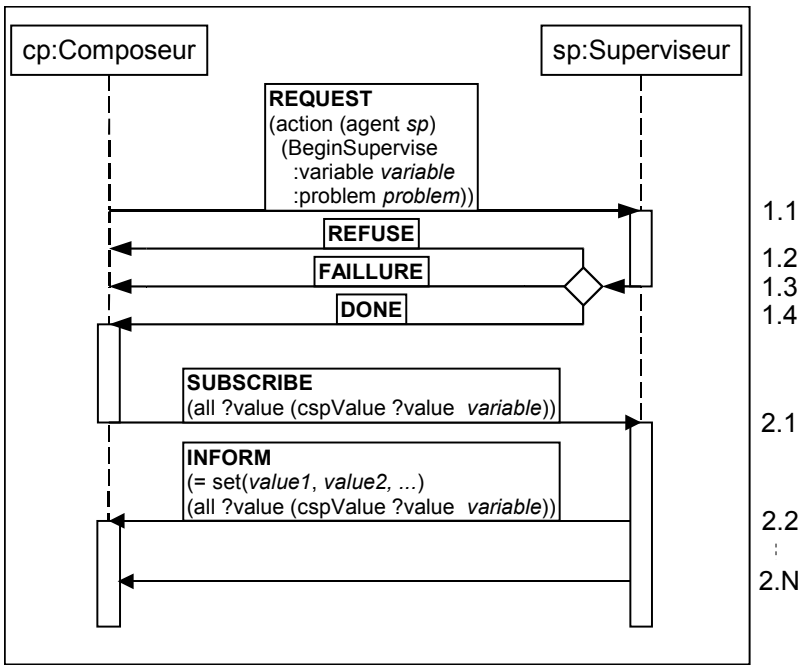

FIGURE 7.3 – Protocole d'interaction `SuperviseFunctionality`

Ce protocole se déroule en deux phases :

1. *Affectation d'une fonctionnalité abstraite* : le composeur demande tout d'abord au superviseur de prendre en charge une des fonctionnalités nécessaires à l'application (1.1). Pour cela, le composeur fournit une description de la fonctionnalité abstraite concernée (cette description est extraite de la description globale de

l'application). Le superviseur démarre alors la recherche de fonctionnalités appropriées pour réaliser la fonctionnalité abstraite, et informe le composeur du succès (1.4) ou de l'échec (1.3) de cette initialisation. Le superviseur peut aussi refuser de prendre en charge cette fonctionnalité abstraite (1.2).

2. *Obtention de propositions de fonctionnalités* : une fois l'initialisation effectuée, le composeur s'inscrit pour recevoir des propositions de fonctionnalités appropriées (2.1). Au fur et à mesure, le superviseur fournit au composeur une liste de description des fonctionnalités appropriées dans la situation courante (2.2 ... 2.N).

c) Le protocole d'interprétation de description

Le protocole `InterpretDescription` implique un agent de résolution (composeur ou superviseur) et un nombre indéfini d'agents d'interprétation. Il s'agit d'un protocole d'interaction de groupe : l'agent de résolution n'adresse pas un message à un agent bien déterminé, mais à un groupe de discussion dont les membres ne sont pas définis. Ainsi, n'importe quel agent d'interprétation peut s'intéresser à un groupe de discussion et décider de fournir des informations à un superviseur ou à un composeur sans que celui-ci ne lui ait spécifiquement demandé. Le déroulement de ce protocole est présenté dans la figure 7.4 (avec un seul agent d'interprétation)

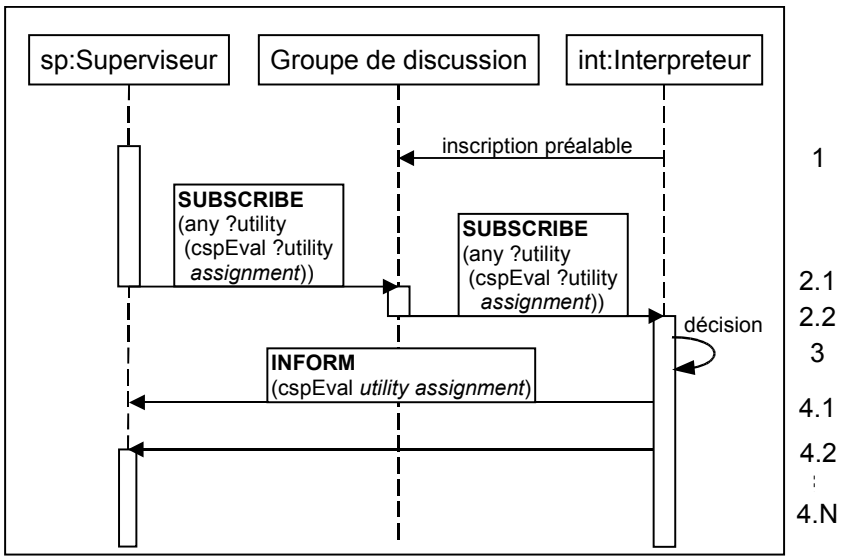

FIGURE 7.4 – Protocole d'interaction `InterpretDescription`

Ce protocole se déroule en quatre phases :

1. *Enregistrement des agents d'interprétation* : à tout moment un agent d'interprétation peut s'inscrire auprès du médiateur pour recevoir les demandes d'interprétations à effectuer (1).
2. *Publication d'une demande d'interprétation* : l'agent de résolution publie une instanciation pour laquelle il souhaite obtenir des interprétations (2.1). Une instanciation est définie comme une liste de couples (description de fonctionnalité abstraite, description de fonctionnalité réelle). Lorsqu'une demande d'interprétation est publiée, elle est accessible à tous les agents d'interprétation. Par ailleurs, elle est directement transmise aux agents d'interprétation qui se sont préalablement inscrits (2.2).

3. *Décision d'interprétation* : chaque agent d'interprétation décide s'il souhaite proposer une interprétation correspondant à cette instanciation (3).
4. *Déclaration d'évaluations* : chaque agent d'interprétation peut décider de s'intéresser à une instanciation publiée et indiquer une ou des contraintes qu'il connaît sur cette instanciation. Dans ce cas, il envoie directement au superviseur concerné un ou des messages contenant les informations sur une contrainte dont il veut l'informer (4.1 ... 4.N). Chaque message décrit une contrainte en indiquant notamment son type et le coût qui lui est associé.

d) Le protocole de description de solution

Le protocole `DescribeSolution` implique tous les agents du système. Il permet de reconstituer la description de la configuration choisie à partir des différentes interprétations de fonctionnalités fournit par différents agents d'interprétation. Le déroulement de ce protocole est présenté dans la figure 7.5.

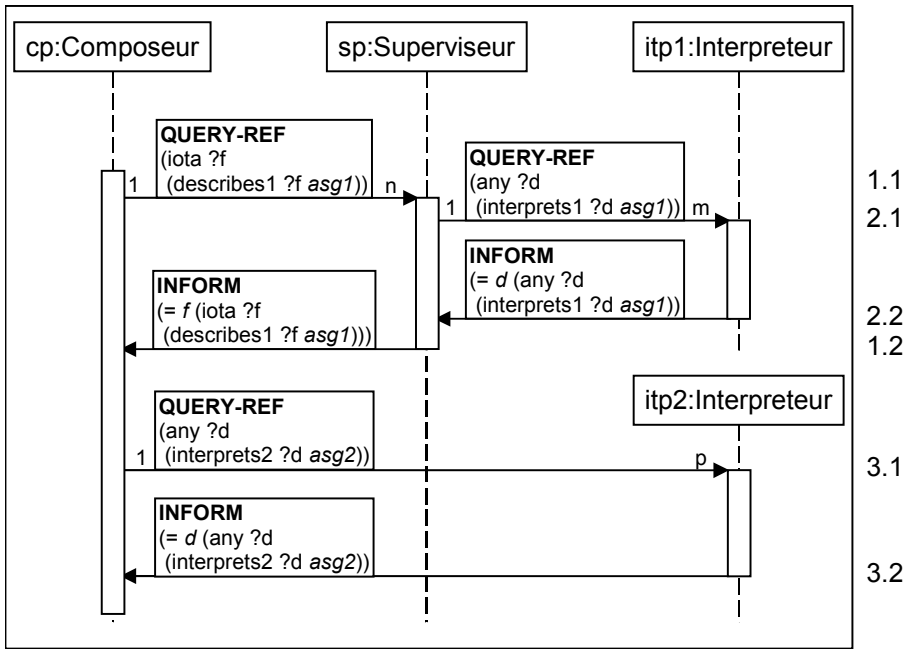

FIGURE 7.5 – Protocole DescribeSolution

Ce protocole se déroule en trois phases :

1. *Requête de description contextualisée des fonctionnalités individuelles* : le composeur demande à chacun des n superviseurs une description contextualisée de la fonctionnalité choisie (1.1).
2. *Requête de descriptions des interprétations locales* : chaque superviseur demande à chacun des m agents d'interprétation ayant proposé une interprétation pour la fonctionnalité choisie de décrire son interprétation (2.1). Après la réponse des agents d'interprétation (2.2), Il agrège les interprétations obtenues et fournit au composeur une description contextualisée de la fonctionnalité qui le concerne (1.2).
3. *Requête de descriptions des interprétations partagées* : le composeur demande à chacun des p agents d'interprétation ayant proposé une interprétation portant

sur plusieurs fonctionnalités de décrire son interprétation (3.1). Il collecte ensuite les réponses des agents d'interprétation (3.2).

A l'issue du déroulement de ce protocole, l'agent composeur agrège finalement les interprétations obtenues auprès des superviseurs (1.2) et celles obtenues auprès des agents d'interprétation (3.2) pour former une description de la configuration choisie.

7.2 Agents de résolution

Le cœur du système de gestion de composition d'une application est constitué par une équipe d'agents de résolution. La mission de cette équipe est la mise en place de configurations adaptées aux différentes situations rencontrées.

Cette section présente le fonctionnement interne des deux types d'agents de résolution. La section 7.2.1 présente tout d'abord l'architecture générique des agents de résolution. La section 7.2.2 détaille le fonctionnement d'un agent superviseur. La section 7.2.3 détaille le fonctionnement d'un agent composeur.

7.2.1 Architecture d'un agent de résolution

Les agents de résolution (superviseurs et composeurs) sont des agents rationnels, dont le comportement est dirigé par des buts. Cette section présente une architecture d'agent simple, inspirée des travaux existants sur les architectures d'agents. Dans les sections suivantes, les comportements précis des agents superviseurs et composeurs sont explicités dans le cadre de cette architecture.

La figure 7.6 illustre l'architecture d'un agent de résolution. Elle se compose de trois élément principaux :

La base de croyance stocke les connaissances internes de l'agent (appelées croyances). Les croyances représentent les informations dont l'agent dispose sur le monde extérieur, et notamment sur les autres agents. Dans la suite, les croyances sont exprimées à l'aide du langage FIPA-SL et de plusieurs ontologies.

La librairie de plans contient un ensemble de plans que l'agent utilise pour atteindre ses buts, ou pour réagir aux évènements qui surviennent dans l'environnement.

Le moteur d'exécution des plans coordonne l'exécution des plans de l'agent en fonction des croyances, des intentions et des évènements.

En plus de ces éléments principaux, chaque agent de résolution possède une *librairie de fonctions*, qui sont en particulier dédiées à la manipulation d'information dans l'espace de description. Ces fonctions sont utilisées dans les plans et s'appuient sur les mécanismes de *gestion de description* décrits dans la section 5.3.1.

a) La base de croyances

La base de croyance contient les croyances de l'agent. Les croyances modélisent la connaissance que possède l'agent sur l'environnement et sur les autres agents. Les croyances sont exprimés à l'aide du langage FIPA-SL ((FIPA, 2002c)). FIPA-SL est aussi utilisé par les agents pour exprimer des messages et échanger leurs connaissances. Par ailleurs, FIPA-SL permet d'exprimer les croyances d'un agent sur ses intentions et sur les intentions des autres agents.

Modèle commun des croyances Les connaissances internes des agents de résolution sont modélisées en utilisant le langage FIPA-SL et une ontologie qui définit les concepts et les prédicats manipulés par l'agent. Cette section détaille l'ontologie commune aux deux types d'agents de résolution. Ces concepts et prédicats sont en

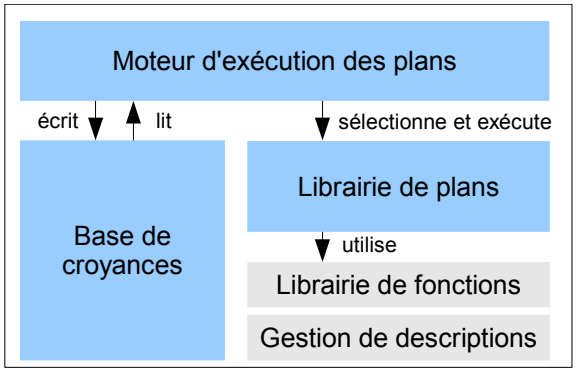

FIGURE 7.6 – Architecture d'un agent de résolution

particulier utilisés dans les messages échangés entre les agents de résolution. Les tables 7.1 et 7.2 détaillent ces concepts et prédicats. Noter que les concepts Assign1 et Assign2 ainsi que les prédicats cspEval1 et cspEval2 permettent de simplifier les notations dans un cadre restreint aux contraintes dont l'arité est inférieure ou égale à 2.

b) La librairie de plans

Les différentes capacités des agents sont réalisées par des plans. La bibliothèque de plans contient l'ensemble des plans dont dispose un agent. Deux catégories de plans peuvent être distinguées :

Les plans réactifs sont des plans qui permettent de réagir à des évènements particuliers.

Les plans orientés par un but sont des plans qui permettent de satisfaire un but de l'agent. Ces plans sont fréquemment organisés de manière hiérarchique : un plan peut définir des sous-buts pour lesquels d'autres plans seront activés. Ceci permet une décomposition du comportement de l'agent.

Un plan est constitué de deux parties :

La condition d'activation définit quelles intentions ou évènements entraînent une activation du plan. La condition d'activation peut contenir des variables qui sont alors des paramètres du plan.

Le corps du plan définit l'enchaînement des opérations effectuées par le plan. Le plan enchaîne des opérations primitives selon une procédure qui peut comprendre des branchements.

Dans la suite, on présente les plans des agents de résolution à l'aide d'une représentation graphique inspirée des diagrammes d'activité UML (OMG, 1999). Les paragraphes suivants détaillent les représentations utilisées pour les différents types de condition d'activation et les différentes opérations primitives.

La condition d'activation La condition d'activation est le premier élément d'un plan. On distingue trois types de condition d'activation. La table 7.3 détaille ces trois types ainsi que la représentation graphique utilisée dans les définitions de plans.

Pour tous les types de conditions, il est possible d'indiquer des paramètres variables : dans ce cas, la condition d'activation est remplie s'il existe des valeurs des paramètres pour lesquels la condition est remplie. Ces valeurs de paramètres peuvent être utilisées dans le plan. Dans la suite, les paramètres variables sont précédés d'un

Expression SL	Signification
(Description :res [r:URI] :doc [d:URL])	désigne la description $r = (\underline{r}, \underline{D})$ d'une ressource d'URI $\underline{r}$ dans un document d'URL $\underline{D}$.
(CSP :ref [r:Description])	désigne un réseau de contraintes associé à l'application décrite par la description r.
(Variable :desc [a:Description])	désigne la variable associé à la description a d'une fonctionnalité abstraite.
(Value :desc [f:Description])	désigne la valeur associée à la description f d'une fonctionnalité.
(Assign1 :val [v:Value] :var [x:Variable])	désigne une instanciation dans laquelle une seule variable est instanciée.
(Assign2 :val1 [v:Value] :val2 [w:Value] :var1 [x:Variable] :var2 [y:Variable])	désigne une instanciation dans laquelle deux variables sont instanciées.
(Utility :type [t:STRING] :date [d:STRING] :cost [c:INTEGER] :limit [l:INTEGER])	désigne l'utilité d'une interprétation, et indique le type t de l'interprétation, la date d, le coût c et le coût limite l.
(BeginCompose :reference [d:Description])	désigne l'action de commencer la composition basée sur l'application décrite par d.
(BeginSupervise :variable [v:Variable] :problem [p:CSP])	désigne l'action de commencer la supervision d'une variable v du problème p.

TABLE 7.1 – Concepts manipulés par tous les agents de résolution

Expression SL	Signification
(cspVariable [x:Variable] [p:CSP])	indique que x est une variable du réseau de contraintes p.
(cspValue [v:Value] [x:Variable])	indique que v est une valeur possible de la variable x.
(cspChoice [v:Value] [x:Variable])	indique que la variable x est instanciée par la valeur v.
(cspEval1 [u:Utility] [asg1:Assign1])	indique que l'utilité u caractérise une interprétation des fonctionnalités de l'instanciation $asg1$ (d'arité 1).
(cspEval2 [u:Utility] [asg2:Assign2])	indique que l'utilité u caractérise une interprétation des fonctionnalités de l'instanciation $asg2$ (d'arité 2).
(describes1 [d:Description] [asg1:Assign1])	indique que la description d est une description contextualisée des fonctionnalités de l'instanciation $asg1$.
(interprets1 [d:Description] [asg1:Assign1])	indique que la description d est une interprétation des fonctionnalités de l'instanciation $asg1$.
(interprets2 [d:Description] [asg2:Assign2])	indique que la description d est une interprétation des fonctionnalités de l'instanciation $asg2$.
(compose [cp:Agent] [p:CSP])	indique que l'agent cp compose une application avec le problème p.
(supervise [sp:Agent] [x:Variable])	indique que l'agent sp supervise la variable x.

TABLE 7.2 – Prédicats manipulés par tous les agents de résolution

caractère $, par exemple $x. Si leur valeur est utilisée dans le plan, elle est indiquée en italique, par exemple x.

On peut noter que les deux types de plans considérés (réactif ou orienté par un but) se distinguent par la condition d'activation utilisée : les plans réactifs sont activés par une perception ou un message, les plans orientés par un but sont activés par un l'activation du but correspondant.

Les opérations primitives d'un plan Le corps d'un plan est constitué d'opérations primitives, enchaînées selon une procédure particulière. Les opérations primitives sont regroupées en cinq catégories : interagir avec l'environnement, manipuler les croyances, manipuler les intentions, interagir avec d'autres agents, et exécuter des fonctions internes. Les tableaux 7.4 et 7.5 détaillent les différentes opérations primitives.

c) Le moteur d'exécution

Le moteur d'exécution est un système très simple : il réagit à la présence de certaines informations dans la base de croyance pour sélectionner le plan correspondant et exécuter ses opérations.

La *sélection d'un plan* est réalisée en surveillant les conditions d'activation des différents plans. Lorsqu'une condition d'activation est remplie, le plan correspondant est sélectionné et exécuté. Dans cette architecture simple, on ne détaille pas de mécanismes d'arbitrage permettant de résoudre d'éventuels conflits entre les plans. Les plans présentés dans la suite sont conçus de manière à éviter les ambiguïtés et les conflits. Ils peuvent ainsi être exécutés dans un ordre indéterminé et en parallèle.

L'*exécution d'un plan* consiste à effectuer les opérations primitives suivant la procédure définie par le corps du plan. Diverses constructions de branchements peuvent être exprimées, suivant les notations des diagrammes d'activités UML. La figure 7.7 donne un exemple de plan comportant un branchement conditionnel et un branchement parallèle. L'exécution de ce plan est la suivante :

1. La condition d'activation est vérifiée. Par exemple, il existe un ensemble A tel que condition(A) est vraie. Le paramètre variable $X prend alors la valeur A.
2. Le branchement conditionnel est testé : si A contient un unique élément a, l'opération operation2 est effectué. Dans le cas contraire, pour chaque élément x de A, l'opération operation1(x) est effectuée. On considère que les opérations pour chaque élément sont effectuées en parallèle. La fin du branchement est atteinte lorsque toutes les opérations sont terminées.
3. Les deux opérations operation3 et operation4 sont effectuées en parallèle. Ces deux opérations sont des interactions avec l'extérieur (par exemple envoyer un message ou effectuer une action). L'opération operation3 est bloquante. Le plan ne se poursuit que lorsqu'elle est terminée (par exemple, une réponse a été reçue). L'opération opération4 n'est pas bloquante : l'agent ne vérifie pas que l'opération c'est effectivement terminée avant de continuer l'exécution.

La *terminaison d'un plan* consiste à effectuer les opérations de gestion associées au plan. En particulier, pour un plan orienté par un but, la terminaison correcte du plan entraîne l'ajout de croyances indiquant que le but est satisfait.

Exécution de plans standards pour la gestion des connaissances En plus des plans spécifiquement définis pour l'agent, le moteur d'exécution est capable d'exécuter des plans standards pour la gestion des connaissances et le partage de connaissance entre les agents. En particulier, un message reçu par l'agent peut être

Représentation graphique	Description de la condition
Perception *expression*	Le plan est activé lorsque les informations que l'agent perçoit sur son environnement correspondent à l'expression indiquée. Les perceptions d'un agent de résolution concernent en particulier l'apparition/disparition de fonctionnalités dans l'environnement.
Réception message *agent PERFORMATIF (expression SL)*	Le plan est activé lorsque l'agent reçoit un message de la part l'agent indiqué avec le performatif indiqué et un contenu correspondant à l'expression SL indiquée. Le nom de l'agent émetteur et le performatif peuvent être des paramètres variables.
Activation but *(expression SL)*	Le plan est activé lorsque le but correspondant à l'expression SL indiquée est activé.

TABLE 7.3 – Types de condition d'activation des plans d'un agent de résolution

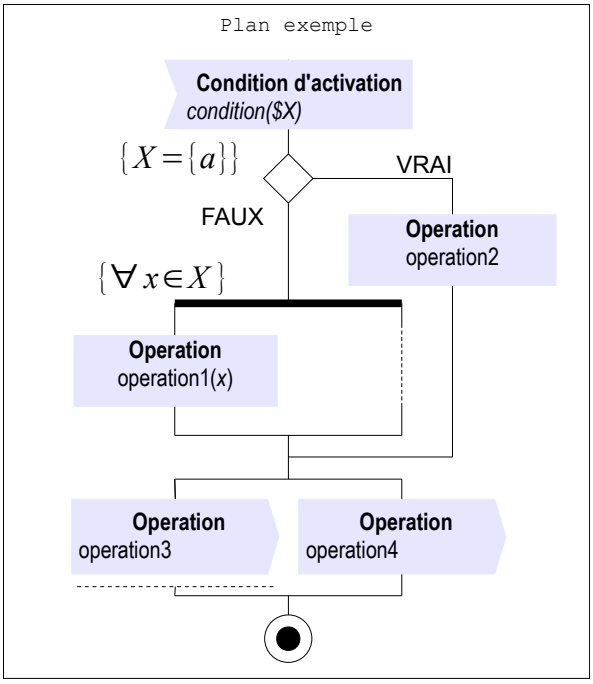

FIGURE 7.7 – Exemple de plan

Interaction avec l'environnement	
Effectuer action *expression*	exécute l'action décrite par l'expression indiquée. Cette primitive est bloquante : elle attend la fin de l'action pour se terminer. Elle peut éventuellement retourner un résultat.
Percevoir *expression*	effectue une perception active, c'est-à-dire cherche à obtenir une information sur l'environnement. Cette primitive est bloquante et retourne l'information perçue.
Manipulation des croyances	
Ajouter croyance *(expression SL)*	ajoute à la base de croyances la croyance correspondant à l'expression SL fournie.
Enlever croyance *(expression SL)*	retire de la base de croyances la croyance correspondant à l'expression SL fournie.
Interroger croyance *(expression SL)*	interroge la base de croyances concernant la croyance correspondant à l'expression SL indiquée. Dans l'expression de la croyance, il peut exister des termes variables. Si l'expression de la croyance contient des termes variables, cette primitive retourne un ensemble de termes correspondants dans la base de croyances.

TABLE 7.4 – Les primitives des plans d'un agent de résolution

Manipulation des intentions	
Adopter but *TYPE (expression SL)*	adopte un but décrit par l'expression SL fournie. Le type de but indiqué précise s'il s'agit d'un but persistant ou non. Le type ATTEINDRE désigne un but non persistant, c'est-à-dire un but qui disparaît dès qu'un plan a permis d'atteindre un état où le but est satisfait. Le type MAINTENIR désigne un but persistant, c'est-à-dire que l'état où le but est satisfait doit être maintenu. Lorsque ce n'est plus le cas, un nouveau plan doit être effectué pour essayer d'atteindre le but de nouveau.
Abandonner but *TYPE (expression SL)*	abandonne le but décrit par l'expression SL fournie.
Prévoir but *date TYPE (expression SL)*	prévoit d'adopter le but décrit à une date ultérieure indiquée. Cette primitive permet aux agents de gérer la dynamicité de l'environnement attentif en retardant certains traitements.
Interaction avec d'autres agents	
Envoyer message *agent PERFORMATIF (expression SL)*	envoie à l'agent indiqué un message avec le performatif et le contenu indiqués.
Recevoir message *agent PERFORMATIF (expression SL)*	attend la réception d'un message avec le performatif et le contenu indiqués de la part de l'agent indiqué. Le nom de l'agent émetteur et le performatif peuvent être des paramètres variables. Le contenu peut aussi être partiellement défini et contenir des termes variables.
Initier discussion *PEFORMATIF (expression SL)*	initie une discussion de groupe commençant par un message avec le performatif et le contenu indiqués.
Exécution de fonctions internes	
Executer fonction *expression fonction*	Exécute la fonction indiquée.

TABLE 7.5 – Les primitives des plans d'un agent de résolution (suite)

traité automatiquement par un plan standard, qui interroge la base de croyance et fournit une réponse. Dans la suite, on considère notamment qu'il existe des plans standards pouvant prendre en charge des messages dont les performatifs sont les suivants :

INFORM Lorsque l'agent reçoit un message INFORM, un plan standard ajoute les informations contenues dans ce message à sa base de croyances.

QUERY, QUERY-REF Lorsque l'agent reçoit un message QUERY ou QUERY-REF, un plan standard interroge la base de croyances pour répondre à la requête exprimée.

SUBSCRIBE Lorsque l'agent reçoit un message SUBSCRIBE ou QUERY-REF, un plan standard met en place un mécanisme pour notifier l'agent demandeur lorsque la réponse à la requête change dans la base de croyances.

Dans la suite, on indiquera simplement lorsqu'un plan standard intervient dans le déroulement du comportement d'un agent, sans détailler les opérations effectuées par ce plan standard.

7.2.2 Agent superviseur de fonctionnalité

Un agent superviseur prend en charge la supervision d'une fonctionnalité seule, qui correspond à une variable du CSP. Son rôle est de définir un ensemble de valeurs possibles pour cette variable, en tenant compte des fonctionnalités disponibles dans l'environnement et des contraintes locales sur la variable. Cette section détaille le fonctionnement de l'agent superviseur en précisant son modèle de croyances, les perceptions et actions dont il dispose puis l'ensemble des plans qu'il utilise.

a) Perceptions et actions d'un agent superviseur

Le superviseur utilise uniquement la découverte de descriptions fournie par la plateforme. On définit ainsi les deux actions et les deux perceptions :

registerDiscovery([f: Description]) est l'action de s'inscrire auprès du découvreur de descriptions en spécifiant la description f comme filtre.

deregisterDiscovery([f: Description]) est l'action de suspendre l'inscription basée sur le filtre f.

discovered([d: Description], [f: Description]) est la perception déclenchée par la découverte d'une description de fonctionnalité.

disappeared([d: Description], [f: Description]) est la perception déclenchée par la disparition d'une description de fonctionnalité.

La réception de perceptions nécessite l'exécution préalable des actions d'inscription. Les perceptions peuvent ensuite apparaître à tout moment, sans nécessiter une recherche active de la part de l'agent.

b) Croyances et intentions internes d'un agent superviseur

En plus du modèle de croyances défini dans la section 7.2.1, un agent superviseur utilise les prédicats suivants :

(allowed [asg1:Assign1]) représente que l'instanciation asg1 est autorisée (c'est-à-dire que l'agent ne connait pas de contrainte qui l'interdise explicitement).

L'agent superviseur peut définir les intentions suivantes :

(evaluating [asg1:Assign1]) désigne l'intention d'effectuer l'évaluation d'une instanciation asg1 de la variable, c'est-à-dire de chercher à obtenir des interprétations pour cette instanciation.

(analysed [asg1:Assign1]) désigne l'intention d'analyser l'instanciation `asg1` de la variable, c'est-à-dire de déterminer si cette instanciation est permise en fonction des interprétations connues.

(solved [x:Variable]) désigne l'intention de déterminer l'ensemble des valeurs possibles pour la variable `x`.

c) Plans d'un agent superviseur

Cette section détaille les différents plans que l'agent superviseur utilise pour gérer ses connaissances et ses actions. Les plans de l'agent superviseur sont divisés en quatre ensembles de plans :

- La mise en œuvre de la supervision, qui comprend notamment la découverte de fonctionnalités dans l'environnement.
- La collecte d'information sur les contraintes locales.
- La résolution des contraintes locales, qui permet de déterminer un ensemble de valeurs possibles pour la variable supervisée.
- L'expression de la description contextualisée de la fonctionnalité choisie.

Mise en œuvre de la supervision Cet ensemble de plans permet à un agent superviseur de commencer la supervision d'une variable. En particulier, les plans permettent d'obtenir des descriptions de fonctionnalités, grâce au mécanisme de découverte fourni par FCAP. Deux plans sont utilisés :

Initialisation de la supervision Le plan `BeginSupervise` (figure 7.8) permet de répondre à une demande de supervision provenant d'un agent composeur. Au cours de ce plan, les informations concernant la variable sont tout d'abord enregistrées dans la base de croyance (1). Ensuite, le superviseur inscrit une requête correspondant à la fonctionnalité recherchée auprès du système de découverte fournit par FCAP (2).

Réception de descriptions de fonctionnalités découvertes Le plan `CreateValue` (figure 7.8) permet de prendre en compte la découverte d'une description de fonctionnalité. Ce plan consiste principalement à adopter l'intention d'évaluer l'instanciation de la variable supervisée par la valeur qui vient d'être découverte (2). Le plan définit de plus l'intention future de proposer à l'agent composeur les valeurs trouvées (4). La date à laquelle cette intention sera adoptée est calculée par une fonction `scheduleNewProposal`(). À chaque réception d'une nouvelle description, l'agent redéfinit son intention de proposer les valeurs au composeur en repoussant la date de proposition. Cela lui permet de collecter préalablement des informations sur les nouvelles fonctionnalités qui apparaissent.

Collecte d'information sur les contraintes locales Pour chaque valeur potentielle de la variable supervisée, cet ensemble de plans permet au superviseur d'obtenir des informations sur les contraintes locales portant sur l'instanciation de la variable par cette valeur. Pour cela, l'agent interroge les agents d'interprétation présents dans l'environnement. Les plans de collecte d'interprétation sont les suivants. :

Mise en place de la collecte d'utilités d'interprétation Le plan `CollectInterpretations` (figure 7.9) permet d'initialiser la collecte d'informations sur les interprétations relative à l'instanciation de la variable par une valeur donnée. Ce plan consiste à ouvrir une discussion dans laquelle des agents d'interprétation peuvent proposer leur interprétation pour cette instanciation (1).

Réception des interprétations Le plan `HandleInterpretation` (figure 7.9) permet de prendre en charge une interprétation d'une instanciation. Pour chaque interprétation reçue (0), ce plan effectue une mise à jour de la base de croyances,

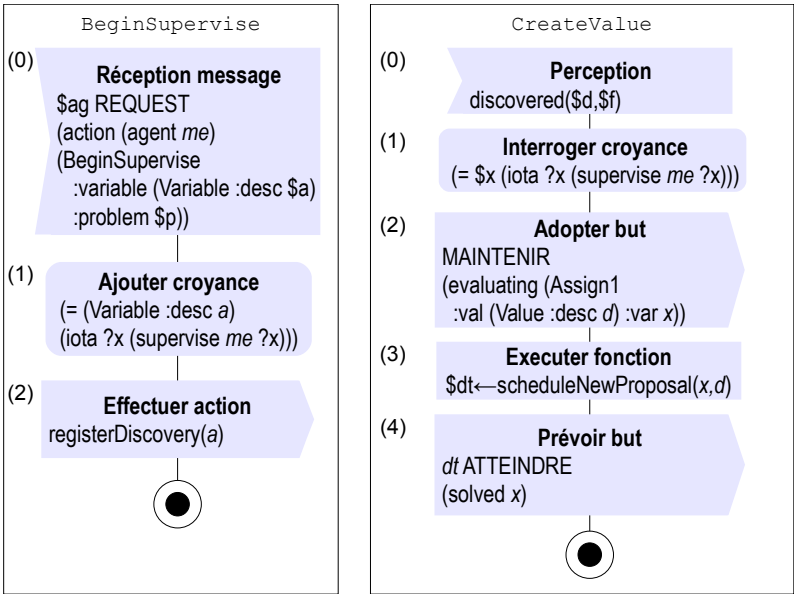

FIGURE 7.8 – Les plans `BeginSupervise` et `CreateValue`

en stockant non seulement d'interprétation fournie, mais aussi quel agent d'interprétation la propose (1). Le plan définit ensuite l'intention future d'analyser l'instanciation concernée par cette interprétation (3). Une fonction `scheduleAnalysis`() calcule la date à laquelle cette intention sera activée (2). L'utilisation d'une intention future permet d'éviter d'effectuer l'analyse juste après chaque réception d'une interprétation, car de nouvelles interprétations peuvent arriver à tout moment (et l'agent ne connait jamais le nombre d'interprétation qu'il doit recevoir).

Résolution des contraintes locales Cet ensemble de plans permet à l'agent de résoudre les contraintes locales, c'est-à-dire de discriminer les valeurs qui pourront faire partie de la solution et les valeurs qui ne peuvent pas être choisies pour cette variable. L'agent utilise les coûts et les coûts limites pour déterminer si la valeur est autorisée. Deux plans sont utilisés :

Détermination des valeurs acceptables Le plan `AnalyzeValue` (figure 7.10) permet de déterminer si une valeur est acceptable pour la variable supervisée. Pour cela, le plan collecte tout d'abord toutes les interprétations connues pour cette valeur (1). La fonction `selectInterpretations1`() lui permet de sélectionner les interprétations utilisées, en particulier lorsque plusieurs interprétations de même type sont disponibles (2). La fonction `isAllowed1`() détermine ensuite si l'instanciation considérée est permise ou non (3). Cette fonction s'appuie sur les coûts et les coûts limites pour effectuer ce choix. Le plan détermine ensuite si l'autorisation pour cette instanciation a été modifiée (5). Si elle a été modifié, il met à jour les connaissances (7), puis définit l'intention future de résoudre le CSP en tenant compte de cette modification (9). La date à laquelle le CSP doit être résolu est calculé par une fonction `scheduleUpdatedProposal`(). Comme précédemment, l'utilisation d'une intention future évite à l'agent d'effectuer la proposition trop tôt, alors que de nouvelles informations pourraient lui parvenir.

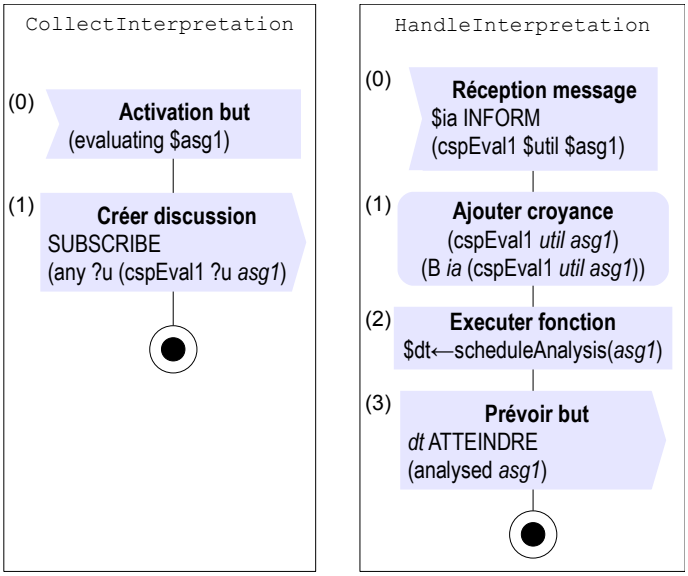

FIGURE 7.9 – Les plans `CollectInterpretations` et `HandleInterpretation`

Proposition des valeurs au composeur Le plan `SetAllValues` (figure 7.10) permet de finaliser la recherche et de proposer au composeur les valeurs pour la variable supervisée. Ce plan vérifie tout d'abord qu'il n'existe aucune analyse d'interprétation en cours (1). Si c'est le cas, ce plan se termine directement. Sinon, le plan obtient l'ensemble des instanciations considérées comme valides pour la variable supervisée (2), et stocke la liste des valeurs possibles dans sa base de croyance (3). Le plan n'envoie pas directement cette liste au composeur, mais un plan standard l'informe automatiquement le composeur, à condition qu'il ait demandé à être tenu informé de la liste des valeurs de la variable.

Expression de la description contextualisée de la fonctionnalité choisie Afin de construire une description de la configuration adaptée, l'agent composeur doit obtenir auprès des agents superviseurs une description contextualisée de la fonctionnalité choisie. Le superviseur utilise des informations fournies par les agents d'interprétation pour construire cette description. Un plan est utilisé pour cela :

Expression de description de fonctionnalité Le plan `DescribeValue` (figure 7.11) construit la description contextualisée associée à une valeur. Il obtient tout d'abord la liste des agents d'interprétation dont les interprétations ont été retenues (1). Il demande ensuite à chacun de ces agents de lui fournir une description de l'interprétation effectuée (2-3). Le plan utilise ensuite une fonction `aggregateComposableDescription`() pour construire une description contextualisée de la fonctionnalité, selon le principe décrit dans la section 6.2.4 (6). Enfin, il stocke la description obtenue dans sa base de croyance (7), et l'envoie à l'agent composeur (8).

7.2.3 Agent composeur d'application

L'agent composeur d'application est directement chargé de la mise en place d'une configuration correspondant aux besoins définis pour l'application. Pour cela, il se base

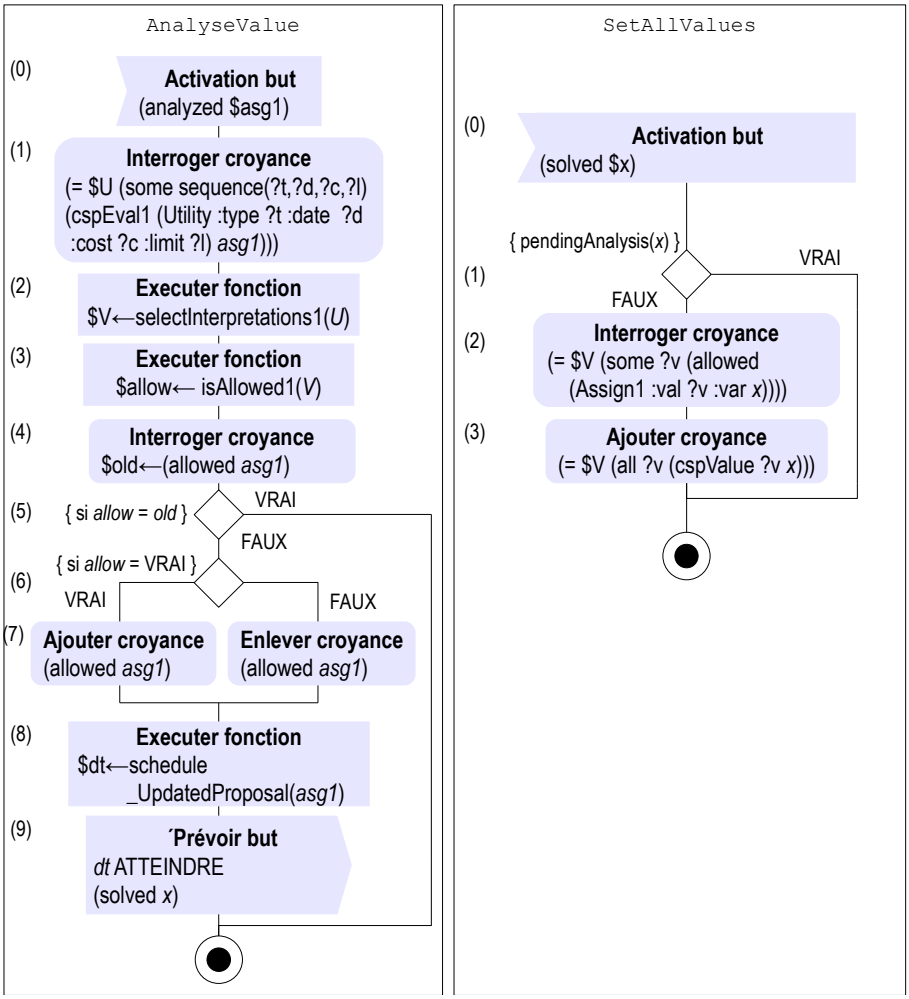

FIGURE 7.10 – Les plans `AnalyzeValue` et `SetAllValues`

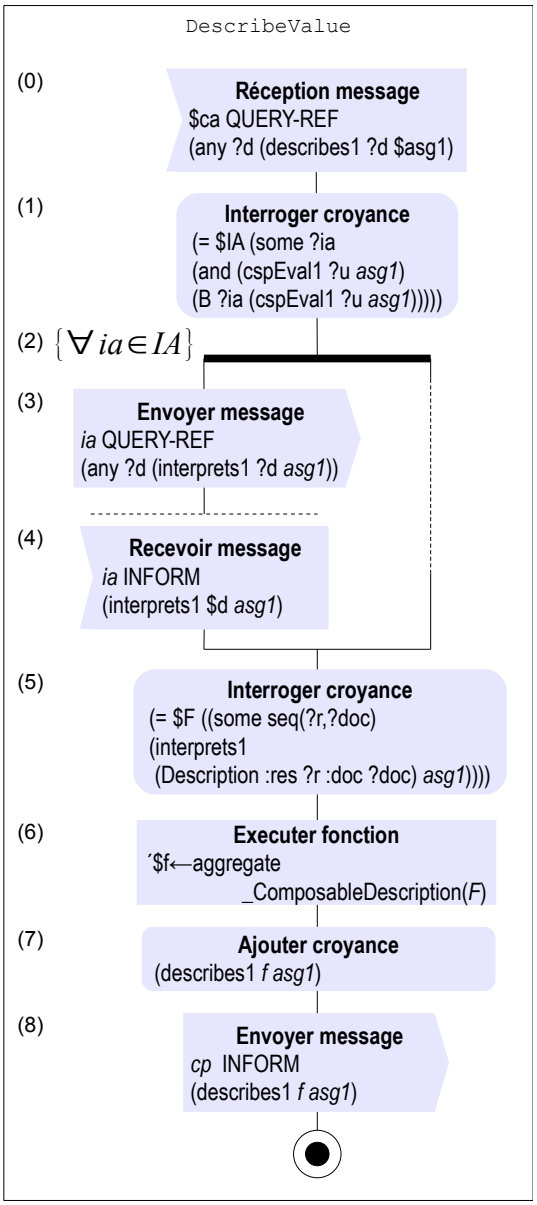

FIGURE 7.11 – Le plan `DescribeValue`

sur la description abstraite de l'application, qui lui fournit les indications essentielles sur les propriétés de l'application à réaliser. À partir de cette description, l'agent composeur définit les variables du réseau de contraintes associé à l'application. Il cherche ensuite à obtenir des valeurs possibles pour chacune des variables, puis à résoudre le problème de satisfaction de contraintes associé à l'application. Enfin, il met en place la configuration adaptée obtenue à partir des solutions du problème de contraintes.

a) Perceptions et actions d'un agent composeur

Les capacités d'actions et de perceptions d'un agent composeur sont liées au support pour la mise en place de configuration fourni par FCAP. L'agent possède ainsi les actions et perceptions suivantes :

Action assemble([app:Description],[F:Description*],[C:Description*]) met en place une configuration pour l'application app. La configuration est décrite par l'ensemble F de descriptions de fonctionnalités et l'ensemble C de descriptions de connecteurs.

Perception assembled([app:Description],[F:Description*],[C:Description*]) informe que la configuration courante pour l'application app est décrite par l'ensemble F de descriptions de fonctionnalités et l'ensemble C de descriptions de connecteurs.

b) Croyances et intentions internes d'un agent composeur

En plus du modèle de connaissances communes décrit dans la section 7.2.1, un agent composeur utilise les prédicats suivants :

(cspNeighbor [x:Variable] [y:Variable]) signifie que les variables x et y sont voisines dans le CSP (il existe une contraintes partagées entre elles).

(available [sp:Agent]) signifie que l'agent (superviseur) sp est disponible.

(allowed [asg2:Assign2]) signifie que l'instanciation de deux variables asg2 est permise.

(describeComposable [f:Description] [a:Description]) signifie que la description f décrit une fonctionnalité de l'application a.

(describeConnector [c:Description] [a:Description]) signifie que la description c décrit un connecteur de l'application a.

L'agent composeur peut définir les intentions suivantes :

(supervised [x:Variable] [p:Problem]) désigne l'intention que la variable x du problème p soit supervisée (par un agent superviseur).

(analysingValue [v:Value] [x:Variable]) désigne l'intention d'analyse la valeur v proposée pour la variable x.

(evaluating [asg2:Assign2]) désigne l'intention d'effectuer l'évaluation de l'instanciation de deux variables asg2, c'est-à-dire de chercher à obtenir des interprétations pour cette instanciation.

(analysed [asg2:Assign2]) désigne l'intention d'analyser l'instanciation de deux variables asg2, c'est-à-dire de déterminer si cette instanciation est permise en fonction des interprétations connues.

(solved [p:CSP]) désigne l'intention de résoudre le problème de contrainte p.

(described [a:Description]) désigne l'intention d'obtenir une description de la configuration adaptée pour l'application a.

(described1 [asg1:Assign1]) désigne l'intention d'obtenir une description de l'instanciation d'un variable asg1.

(described2 [asg2:Assign2]) désigne l'intention d'obtenir une description de l'instanciation de deux variables asg2.

c) Plans d'un agent composeur

Les plans d'un agent composeur sont organisés en cinq ensembles de plans :
- L'organisation de la composition, qui consiste à analyser la description abstraite de l'application et à organiser l'équipe d'agents chargés de gérer la composition.
- La collecte d'information sur les contraintes partagées entre plusieurs fonctionnalités.
- La résolution des contraintes partagées, qui permet d'élaborer une solution au problème de satisfaction de contraintes associé à l'application.
- La construction d'une description pour la configuration choisie.
- La mise en place de la configuration choisie.

Organisation de la composition Cet ensemble de plans permet à un agent composeur d'initialiser la composition d'une application en recrutant des agents superviseurs pour prendre en charge les différentes variables. Ils permettent ainsi à l'agent composeur d'obtenir des ensembles de valeurs possibles pour les différentes variables. Ces ensembles de valeurs sont mis à jour dynamiquement par les superviseurs en fonction des modifications de la situation.

Initialisation de composition Le plan `BeginCompose` (figure 7.12) permet à l'agent de répondre à une demande de composition (0). Au cours de l'exécution de ce plan, les informations contenues dans la description de fonctionnalités abstraites sont lues et permettent d'ébaucher un réseau de contraintes associé à l'application. Pour cela, il utilise une fonction `decompose`(), qui lit la description pour déterminer l'ensemble A des fonctionnalités abstraites et l'ensemble N des fonctionnalités qui interagissent (variables voisines dans le réseau de contraintes) (2). Les différentes informations sur les variables et les voisinages entre variables sont stockées dans la base de croyance de l'agent (4 et 7). Enfin, des intentions concernant la mise en place d'une supervision pour chacune des variables sont définies (5).

Mise en place de supervision de variable Le plan `SuperviseVariable` (figure 7.13) permet à l'agent de satisfaire une intention de mise en place d'une supervision d'une variable. Au cours de ce plan, l'agent commence par chercher un agent superviseur disponible (1), puis lui envoie une requête de supervision (2). Selon la réponse reçue (3 et 4), les connaissances sont mises à jour. Si la requête a échoué (6), le plan se termine sur un échec et le but poursuivi reste actif : le composeur utilise de nouveau ce plan avec d'autres superviseurs. Si la requête a réussi, le composeur s'inscrit pour recevoir les propositions de valeurs possibles pour la variable considérée (5).

Réception des valeurs possibles Le plan `HandleValues` (figure 7.13) permet à l'agent de prendre en charge un ensemble de valeurs proposées par un superviseur. Son déroulement consiste à mettre à jour les connaissances et à définir des intentions concernant l'analyse de ces valeurs possibles. Le plan compare ainsi l'ensemble V des valeurs proposées et l'ensemble U des valeurs précédemment considérées. Pour les nouvelles valeurs (2), il ajoute une intention d'analyse (4). Pour les valeurs qui ont disparu (5), il abandonne le but d'analyse précédemment adopté (7).

Dans ces différents plans, la manière dont on obtient les noms des agents superviseurs disponibles n'est pas précisée : les mécanismes standards de gestion d'agents définis par la FIPA sont utilisés pour découvrir les agents présents dans le système.

Collecte d'information sur les contraintes partagées Lorsque des valeurs pour les variables sont connues, cet ensemble de plans permet au composeur d'obtenir

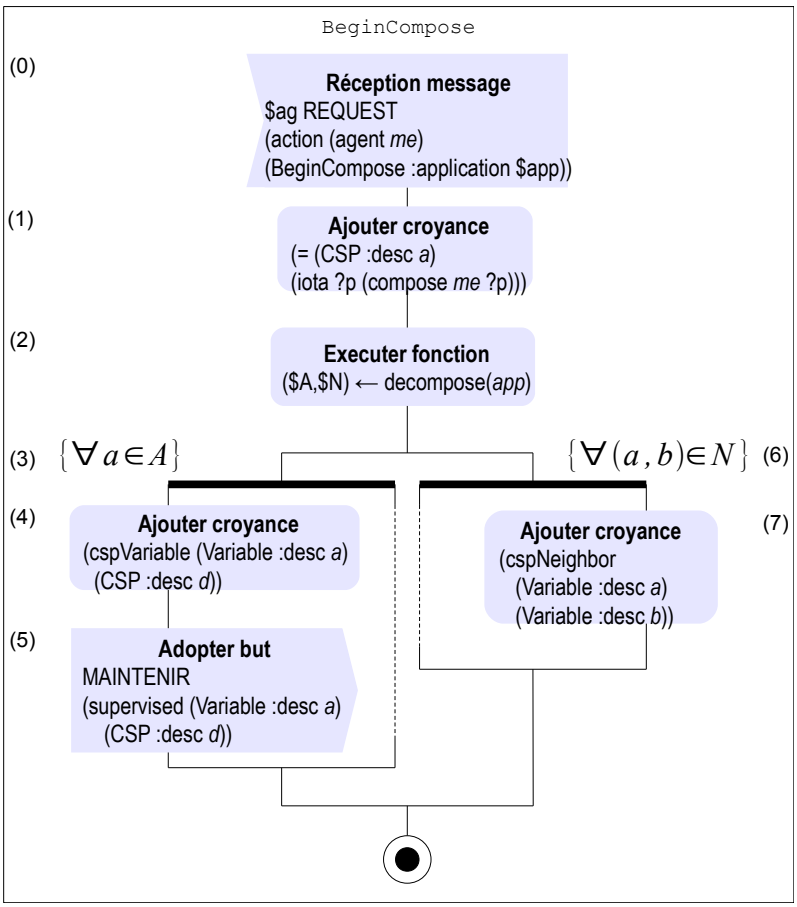

FIGURE 7.12 – Le plan `BeginCompose`

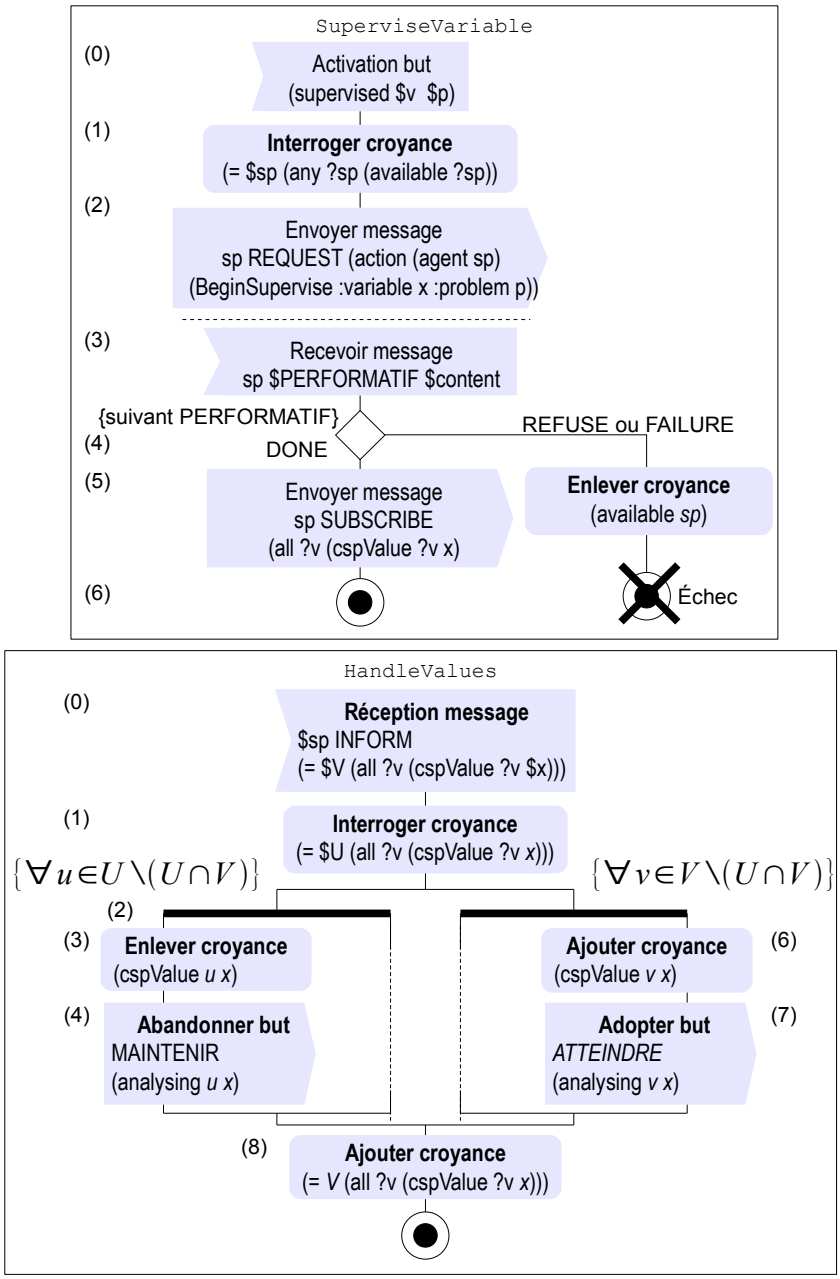

FIGURE 7.13 – Les plan `SuperviseVariable` et `HandleValues`

des informations sur les contraintes partagées portant sur ces variables. Les plans de collecte d'interprétation sont très similaires à ceux de l'agent superviseur :

Détermination des instanciations à interpréter Le plan `AnalyzeValue` (figure 7.14) permet à l'agent de satisfaire l'intention d'analyse de l'instanciation d'une variable x par une valeur v (0). Au cours de son déroulement, ce plan examine les différentes variables voisines de x et leurs valeurs possibles (1-4). Il détermine ainsi les différentes instanciations possible pour les couples de variables, et définit l'intention de maintenir une évaluation pour ces instanciations (5). À la fin du plan, l'agent définit une intention future concernant la résolution du problème de contraintes (8). Si aucune interprétation n'a été reçue au moment où cette intention sera activée, l'agent détectera un problème dans le fonctionnement du système. La date à laquelle cette intention doit être adoptée est calculée par une fonction `scheduleNewResolution`(). Cette date est réévaluée à chaque ajout d'une intention d'évaluer une instanciation (6), et la date finalement retenue est la dernière date calculée.

Mise en place de la collecte d'information sur les interprétations Le plan `CollectInterpretation` (figure 7.14) permet d'initialiser la collecte d'informations sur les interprétations relative à une instanciation de deux variables. Pour chaque instanciation possible, le plan met en place une discussion dans laquelle des interpréteurs peuvent soumettre leurs interprétations (1).

Réception des interprétations Le plan `HandleInterpretation` (figure 7.14) permet à l'agent de prendre en charge les différentes interprétations soumises par les agents d'interprétation. Pour chaque interprétation reçue (0), ce plan effectue une mise à jour de la base de croyances, en stockant non seulement l'interprétation fournie, mais aussi quel agent d'interprétation la propose (1). Le plan définit ensuite l'intention future d'analyser l'instanciation concernée par cette interprétation (3). Une fonction `scheduleAnalysis`() calcule la date à laquelle cette intention sera activée (2). L'utilisation d'une intention future permet d'éviter d'effectuer l'analyse juste après chaque réception d'une interprétation, car de nouvelles interprétations peuvent arriver à tout moment (et l'agent ne connait jamais le nombre d'interprétation qu'il doit recevoir).

Résolution des contraintes partagées L'agent composeur est chargé de résoudre globalement le problème de satisfaction de contraintes associé à une application. Pour cela, il considère uniquement les contraintes partagées : les contraintes locales ont déjà été prises en compte par les agents superviseurs. Les mécanismes de résolution des contraintes partagées sont composés de deux plans :

Analyse d'une instanciation Le plan `AnalyseAssignment` (figure 7.15) permet à l'agent d'analyser une instanciation particulière de deux variables, pour déterminer si cette instanciation est autorisée. Pour cela, le plan collecte tout d'abord toutes les interprétations connues de cette instanciation (1). La fonction `selectInterpretations2`() lui permet de sélectionner les interprétations utilisées, en particulier lorsque plusieurs interprétations de même type sont disponibles (2). La fonction `isAllowed2`() détermine ensuite si l'instanciation considérée est permise ou non (3). Cette fonction s'appuie sur les coûts et les coûts limites pour effectuer ce choix. Le plan détermine ensuite si l'autorisation pour cette instanciation a été modifiée (5). Si elle a été modifié, il met à jour les connaissances (7), puis définit l'intention future de résoudre le problème de contraintes en tenant compte de cette modification (9). La date à laquelle le problème de contraintes doit être résolu est calculé par une fonction `scheduleUpdatedResolution`(). Comme précédemment, l'utilisation d'une intention future évite à l'agent d'effectuer la résolution trop tôt, alors que de nouvelles informations pourraient lui parvenir.

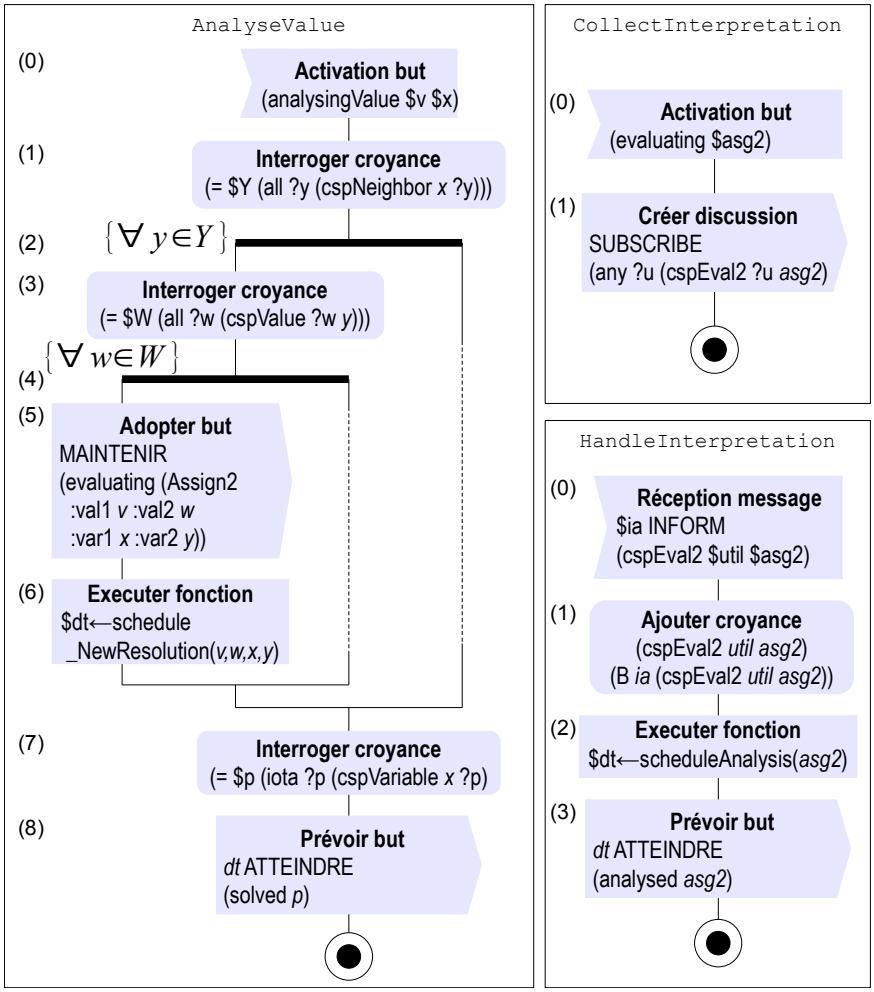

FIGURE 7.14 – Les plan `AnalyzeValue`, `CollectInterpretation` et `HandleInterpretation`

Résolution du problème de contraintes Le plan `SolveCSP` (figure 7.15) permet à l'agent d'effectuer la résolution finale du problème de satisfaction de contraintes. Tout d'abord, il vérifie qu'il n'existe aucune analyse d'interprétation en cours (1). Si c'est le cas, ce plan se termine directement. Sinon, le plan récupère l'ensemble des variables (2), l'ensemble des contraintes (3), et utilise une fonction `solveCSP`() pour résoudre le problème de contraintes (4). En fonction de la solution G du problème, le plan met à jour ses croyances (5-6). Enfin, l'agent adopte l'intention d'obtenir une description de la configuration correspondant à la solution trouvée (8).

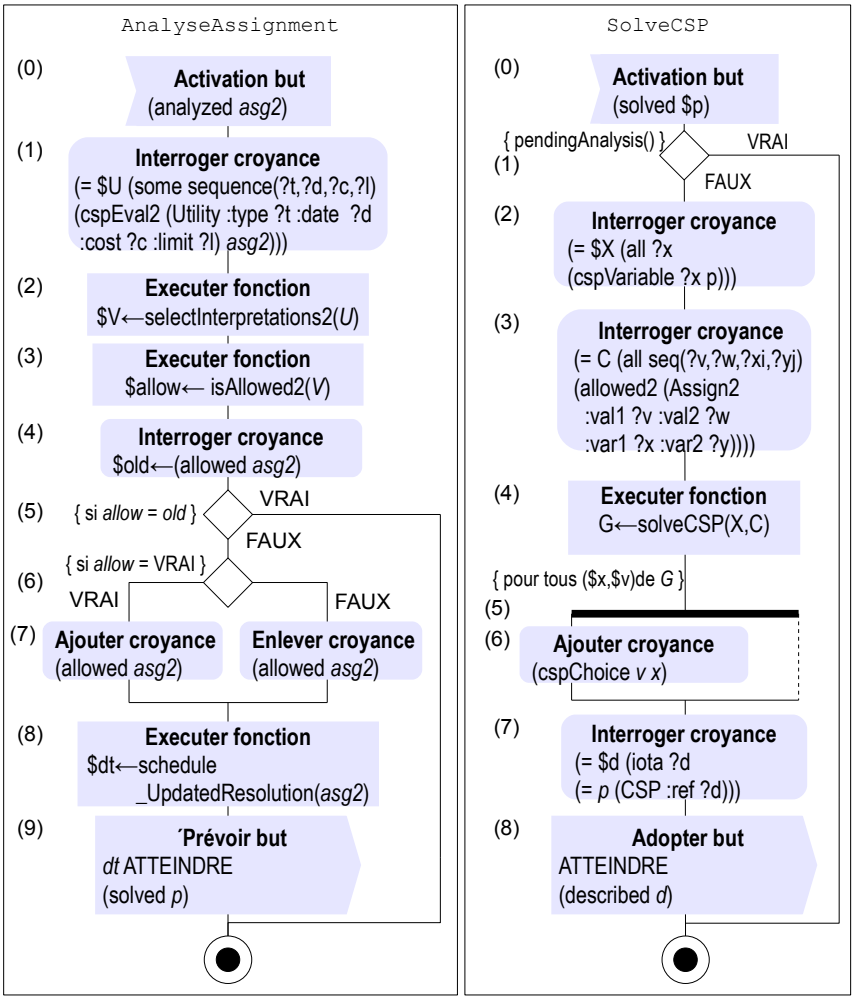

FIGURE 7.15 – Les plans `AnalyseAssignment` et `SolveCSP`

Construction d'une description de la configuration choisie Lorsqu'une nouvelle solution du problème de contraintes est connue, cet ensemble de plans permet à l'agent de construire la description de la configuration correspondante. Pour cela, l'agent doit obtenir les descriptions contextualisées des fonctionnalités et les descriptions des connecteurs, puis les agréger.

Construction de la description d'une configuration Le plan `DescribeConfiguration` (figure 7.16) permet d'obtenir une description de la configuration adaptée. Tout d'abord, l'agent cherche à obtenir des descriptions des fonctionnalités (2-5) ainsi que des descriptions des connecteurs (7-11). Ensuite, le plan utilise la fonction `aggregate`() pour agréger ces descriptions en une description complète de l'application. Enfin, l'agent adopte l'intention permanente de maintenir la configuration souhaitée en fonctionnement.

Obtention de la description d'une fonctionnalité Le plan `GetComposableDescription` (figure 7.17) permet d'obtenir une description contextualisée pour chaque fonctionnalité sélectionnée, en interrogeant le superviseur concerné.

Construction de la description d'un connecteur Le plan `GetConnectorDescription` (figure 7.17) permet au composer d'obtenir des descriptions des interprétations portant sur plusieurs variables, à partir desquels il peut construire une description du connecteur correspondant.

Mise en place de la configuration choisie Cet ensemble de plans permet à l'agent de mettre en place la configuration choisie. Il utilise pour cela le support pour la mise en place de configuration fourni par FCAP. Comme on l'a indiqué dans la description du système d'assemblage de configuration (section 5.3.3), il est possible que la configuration courante soit modifiée par des évènements extérieurs, notamment en cas de panne d'un élément. L'agent dispose d'un plan pour surveiller ces modifications extérieures et réagir.

Invocation du module d'assemblage Le plan `Assemble` (figure 7.18) permet au composeur de mettre en place une configuration en invoquant le module d'assemblage de configuration fourni par FCAP.

Perception de la configuration courante Le plan `Monitor` (figure 7.18) permet à l'agent de recevoir des perceptions provenant du module d'assemblage de configuration indiquant les modifications éventuelles de la configuration. Il peut ainsi chercher à remettre en place la configuration souhaitée.

7.3 Agents d'interprétation

Comme indiqué précédemment, un agent d'interprétation est un agent spécifique, dédié à fournir des informations portant sur un aspect particulier des fonctionnalités et de l'environnement. En conséquence, FCAP ne propose pas d'architecture générique pour les agents d'interprétation : l'architecture de chaque agent dépend de l'aspect pour lequel il fournit une interprétation. Cependant, il est possible de souligner quelques éléments communs du fonctionnement d'un agent d'interprétation ainsi que quelques caractéristiques intéressantes présentées par les agents d'interprétation.

La section 7.3.1 décrit tout d'abord le fonctionnement de référence d'un agent d'interprétation. Ce fonctionnement de référence peut servir de base pour concevoir de nouveaux agents d'interprétation. La section 7.3.2 détaille trois exemples d'agents d'interprétation, dédiés à l'interprétation des types de fonctionnalité, à l'interprétation de conditions de contexte et à l'interprétation de correspondance des services. Enfin, la section 7.3.3 souligne les aspects caractéristiques du comportement des agents de résolution, ainsi que leur impact sur les mécanismes de gestion de composition.

7.3.1 Fonctionnement de référence d'un agent d'interprétation

Bien que chaque agent d'interprétation possède un fonctionnement spécifique, la majorité des agents d'interprétation ont un fonctionnement similaire, que l'on ap-

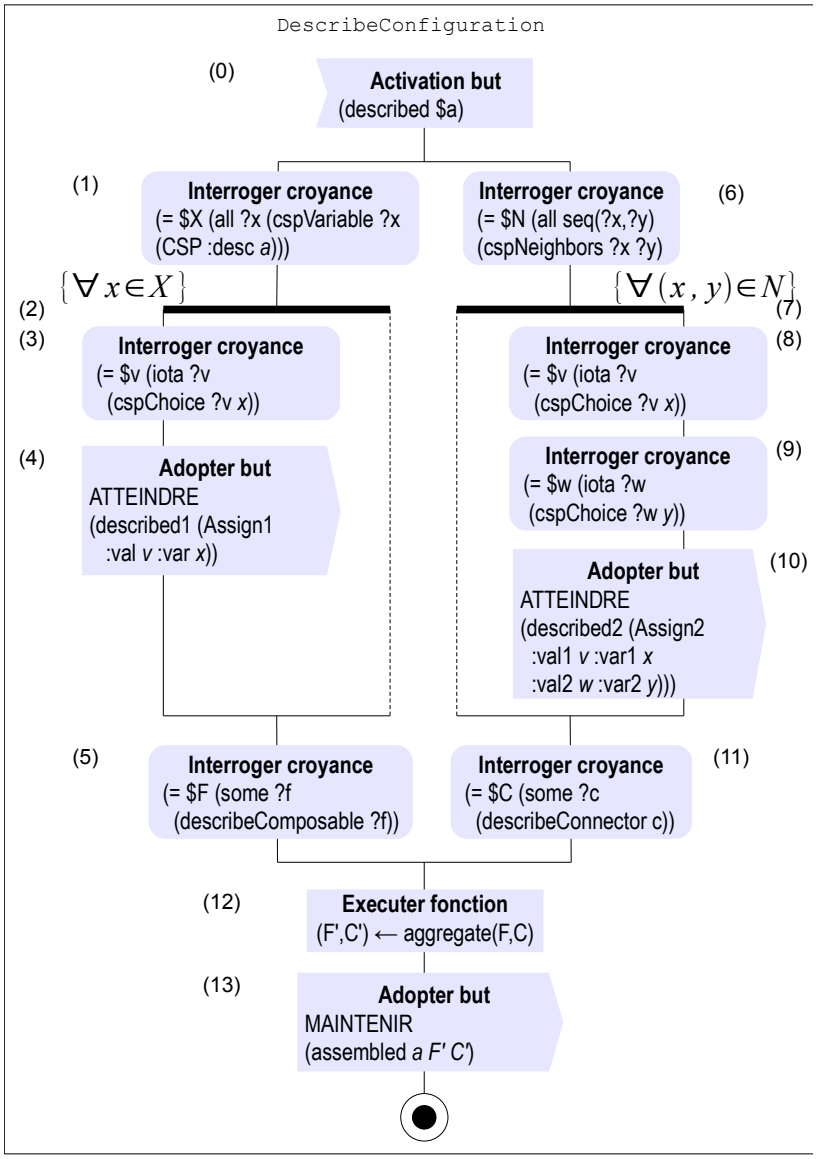

FIGURE 7.16 – Le plan `DescribeConfiguration`

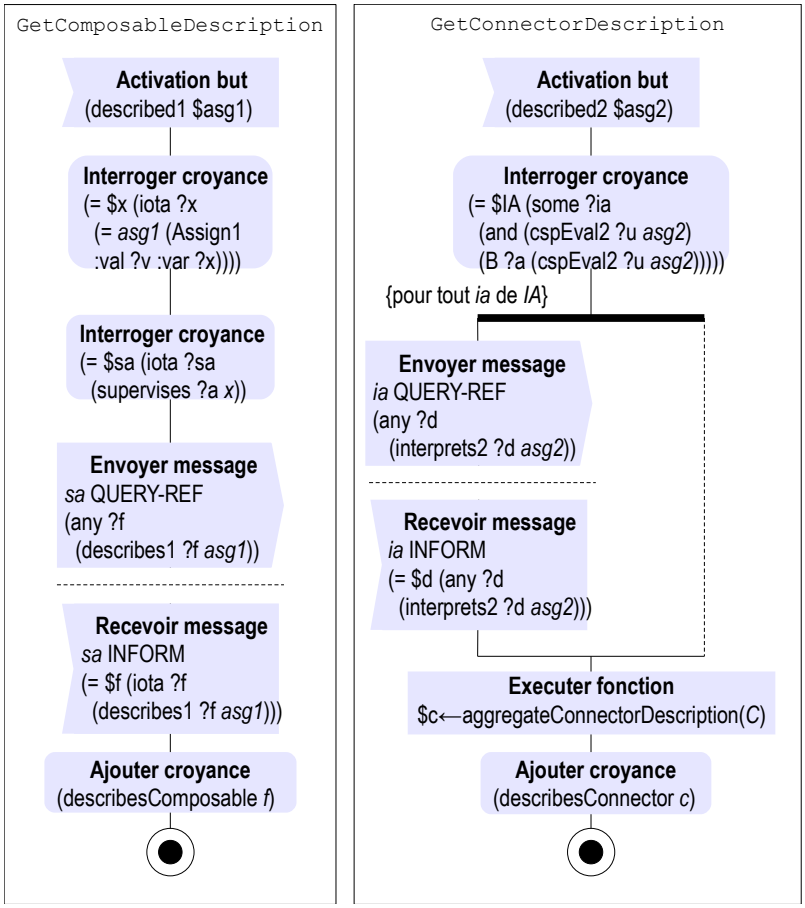

FIGURE 7.17 – Les plans `GetComposableDescription` et `GetComposableDescription`

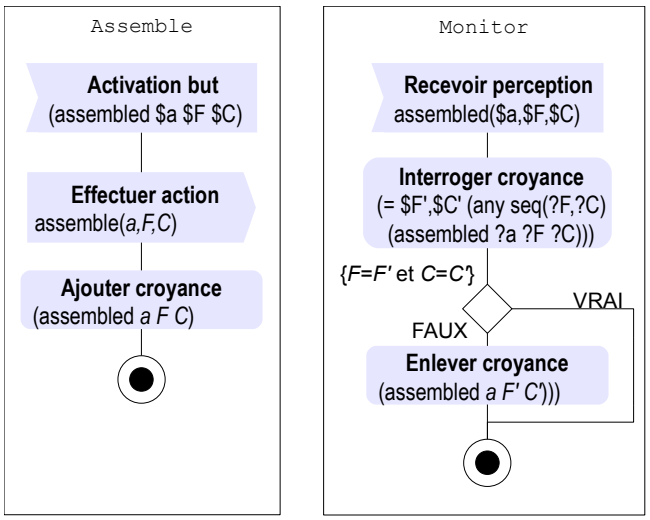

FIGURE 7.18 – Les plans `Assemble` et `Monitor`

pelle ici le fonctionnement de référence. Les fonctionnements réels des différents agents d'interprétation sont généralement proches du fonctionnement de référence, mais présentent des différences liées à leur activité spécifique. Le fonctionnement de référence est en particulier guidé par les protocoles d'interprétation de description (section 7.1.2, figure 7.4) et de description de solution (section 7.1.2, figure 7.5) proposés par FCAP. Le fonctionnement de référence d'un agent d'interprétation est composé de cinq étapes, décrites par les paragraphes suivants.

Intégration dans un système de gestion de composition Lors de son arrivée dans l'environnement, l'agent s'inscrit pour recevoir des demandes d'interprétation. En effet, la présence d'un agent d'interprétation particulier n'est jamais requise pour le fonctionnement du système de gestion de composition : les agents d'interprétation peuvent intégrer ou quitter le système au cours de son fonctionnement. Cette étape d'inscription leur permet d'intégrer dynamiquement un système de gestion de composition. Lors de l'inscription, l'agent peut aussi établir un filtre sur les demandes d'interprétation qui lui seront transmises. Un agent d'interprétation se restreint généralement à traiter uniquement des interprétations portant sur une contrainte locale ou uniquement des interprétations portant sur une contrainte partagée.

Construction d'une interprétation Lorsqu'une demande d'interprétation lui parvient, un agent d'interprétation doit effectuer trois taches principales :

- décider s'il souhaite fournir une interprétation pour les descriptions considérées.
- estimer l'utilité de l'interprétation qu'il peut fournir.
- exprimer son interprétation dans un document de l'espace de description.

Selon les agents, ces trois taches sont réalisées successivement ou conjointement. Souvent, l'agent commence par consulter les descriptions à interpréter pour déterminer s'il peut fournir une interprétation. S'il n'y parvient pas, en particulier si les descriptions ne contiennent pas les informations dont il a besoin, il décide de ne pas donner suite et ne fournit pas d'interprétation. Dans le cas contraire, il construit son interprétation à partir des informations de la description, et détermine l'utilité de l'interprétation obtenue.

Expression d'une utilité d'interprétation Conformément au protocole `InterpretDescription`, l'agent ne fournit dans un premier temps que l'estimation de l'utilité de l'interprétation. L'utilité est exprimé par sous la forme d'un terme (Utility :type [type:STRING] :date [date:DATE] :cost [cost:INT] :limit [limit:INT]) qui comprend les éléments suivants :

type indique le type d'interprétation considéré. Chaque agent d'interprétation s'intéresse à une interprétation particulière, tel que l'interprétation des types de fonctionnalité (section 6.1.1) ou l'interprétation de conditions de contexte (section 6.1.2).

date indique la date à laquelle l'interprétation a été effectuée. Dans le cas d'une interprétation dynamique, l'agent d'interprétation met à jour son interprétation en fonction de la situation, et indique la date à laquelle il considère son interprétation comme valide. Dans le cas d'une interprétation statique, la date est laissée vide : l'interprétation est valide à tout moment.

cost indique le coût Δ_τ associé à l'interprétation effectuée.

limite indique un coût limite λ_τ associé à l'interprétation effectuée.

Expression de l'interprétation dans un document Si l'interprétation proposée par cet agent est retenue par les agents de résolution, l'agent d'interprétation fournit la description complète de l'interprétation qu'il a effectué. Cette description prend la forme d'une description contextualisée de la fonctionnalité réelle, comme défini dans la section 5.2.3. L'agent d'interprétation effectue les opérations de manipulation de description nécessaires pour exprimer l'interprétation dans un document et la mettre à disposition des agents de résolution dans l'espace de description. Il fournit ensuite la référence de cette description aux agents de résolution.

Mise à jour de l'interprétation À tout moment, l'agent peut modifier l'interprétation qu'il propose. En particulier, un agent dont l'interprétation dépend de la situation de l'environnement doit pouvoir la réviser en fonction de l'évolution de l'environnement. Lorsqu'il modifie son interprétation, l'agent informe les agents de supervision en leur fournissant une estimation de l'utilité de cette nouvelle interprétation.

7.3.2 Exemples d'agents d'interprétation

Les agents d'interprétation peuvent couvrir toutes sortes de caractéristiques des applications composites flexibles. Cette section présente trois exemples d'agents d'interprétation fréquemment utilisés pour la réalisation d'applications attentives. Ces agents sont basés sur les algorithmes définis dans la section 6.1.

a) Agent d'interprétation de types de fonctionnalité

Cet agent fournit une interprétation en analysant les types de fonctionnalités à l'aide d'ontologies de domaine. Il utilise pour cela le mécanisme d'interprétation définis dans la section 6.1.1.

Les capacités de cet agent sont principalement fournies par un plan `InterpretTypes`. Ce plan prend en charge une demande d'interprétation et met en œuvre l'algorithme d'interprétation `InterpretTypes` de la section 6.1.1. Les résultats de l'algorithme sont transmis à l'agent de résolution qui demandait l'interprétation.

Intégration avec le module de découverte Cet agent d'interprétation peut être utilisé pour optimiser le processus de découverte. En effet, dans l'architecture présentée, le découvreur de descriptions de fonctionnalités n'est pas capable de déterminer si les fonctionnalités découvertes sont appropriées : il fournit donc de nombreuses descriptions de fonctionnalités non pertinentes au superviseur.

L'interprétation des types de fonctionnalités possède quant à elle un fort pouvoir discriminant et permet d'éliminer simplement de nombreuses possibilités. Une optimisation consiste donc à associer un agent d'interprétation des types de fonctionnalités à un adaptateur de découverte. Au cours de la découverte, l'agent interprète les descriptions de manière pro-active et élimine les fonctionnalités non pertinentes. Ensuite, lorsqu'une interprétation est demandée, il fournit immédiatement les informations obtenues lors de la découverte.

Dans ce mécanisme, les résultats de l'interprétation sont ainsi fournis en trois étapes. Lors de la première étape, seule une information grossière est prise en compte, sous la forme d'une priorité associée à une description découverte. Lors de la seconde étape, une information plus fine est prise en compte, sous la forme d'une indication de l'utilité de l'interprétation de fonctionnalité. Enfin, lors de la dernière étape, une information complète est fournie avec la description de l'interprétation effectuée.

b) Agent d'interprétation de conditions de contexte

Cet agent fournit une interprétation basée sur le mécanisme d'interprétation des conditions de contexte décrit dans la section 6.1.2. En particulier, l'agent prend en charge les interactions avec le module de gestion de contexte de la plate-forme (section 5.3.2), de manière à adapter l'interprétation au contexte courant.

Initialisation de l'interprétation Un plan `InitInterpretation` permet de prendre en charge une demande d'interprétation. Afin d'utiliser le module de gestion de contexte, la première étape consiste à définir les intérêts sur le contexte sous la forme d'une expression SPARQL. Cette étape est basée sur la première partie de l'algorithme `InterpretContextCondition` décrit dans la section 6.1.2. Deux cas se présentent ensuite. Dans le cas où aucune expression d'intérêt sur le contexte ne peut être extraite de la description, l'agent n'a pas d'interprétation à proposer et reste donc silencieux (il ne répond pas à la requête). Dans le cas où une expression d'intérêt sur le contexte peut être définie, l'agent adopte le but permanent de savoir si l'expression est vérifiée dans la situation courante.

Enregistrement d'un intérêt sur le contexte Un plan `RegisterCondition` permet à l'agent d'enregistrer un intérêt sur le contexte auprès du module de gestion de contexte de la plate-forme. L'utilisation d'un but permanent et d'un plan séparé permet à l'agent de renouveler cet enregistrement en cas de problème.

Mise à jour de l'interprétation Le plan `HandleResult` permet à l'agent de mettre à jour son interprétation en fonction des informations sur le contexte. Pour cela, l'agent utilise la section partie de l'algorithme de la section 6.1.2, qui construit une description de l'interprétation à partir du résultat fourni en réponse à la requête. Il stocke ensuite les informations sur l'interprétation dans sa base de croyance. Ces informations sont transmises à l'agent superviseur par les mécanismes standards.

c) Agent d'interprétation de correspondance des services

Cet agent fournit une interprétation basée sur le mécanisme d'interprétation de correspondance de services décrit dans la section 6.1.3.

Les capacités de cet agent sont principalement fournies par un plan `InterpretServiceMatching`. Ce plan prend en charge une demande d'interprétation et met en œuvre l'algorithme d'interprétation `InterpretServiceMatching` de la section 6.1.3.

Les résultats de l'algorithme sont transmis à l'agent de résolution qui demandait l'interprétation.

7.3.3 Propriétés caractéristiques d'un agent d'interprétation

Il est possible de dégager des propriétés caractéristiques fréquemment présentées par les agents d'interprétation. Cette section décrit ces propriétés et leur rôle dans le fonctionnement d'un système de gestion de composition.

Ouverture Un agent d'interprétation peut intégrer ou quitter un système de gestion de composition à tout moment. Il peut de plus faire partie de plusieurs systèmes de gestion de composition. Un système de gestion de composition est ainsi un système ouvert. Cette propriété est essentielle dans le cadre d'applications attentives fonctionnant dans un environnement qui peut évoluer au cours de son fonctionnement.

Hétérogénéité Chaque agent d'interprétation présente un fonctionnement spécifique et il est généralement conçu indépendamment des autres agents d'interprétation. Les agents d'interprétation sont ainsi relativement hétérogènes. FCAP n'impose pas de contrainte sur le fonctionnement interne des agents d'interprétation, et se limite à définir un protocole d'interaction simple pour leur permettre d'interagir avec des agents de résolution. Cette hétérogénéité permet de traduire l'hétérogénéité présente dans un environnement attentif, aussi bien au niveau des fonctionnalités que des stratégies d'adaptation.

Autonomie Les agents d'interprétation sont autonomes vis à vis du reste du système de gestion de composition, car ils sont libres de participer ou non à la composition. La prise en compte de cette autonomie est nécessaire à la flexibilité du système, car la participation d'un agent d'interprétation ne peut jamais être garantie. Un agent de résolution peut ainsi quitter le système librement et choisir de fournir une interprétation selon des critères qui lui sont propres.

Dynamicité Le comportement d'un agent d'interprétation est dynamique : au cours du fonctionnement d'une application, il peut modifier l'interprétation qu'il fournit. Cette propriété est essentielle pour refléter la dynamicité d'un environnement attentif, dans lequel la situation évolue constamment.

Proactivité Un agent d'interprétation peut présenter un comportement pro-actif : il cherche alors à modifier l'environnement en vue d'améliorer l'utilité de l'interprétation qu'il peut fournir. Un agent d'interprétation pro-actif va ainsi non seulement informer un système de gestion de composition d'une contrainte liée à l'utilisation d'une fonctionnalité, mais il va aussi chercher à relâcher cette contrainte pour permettre l'utilisation de cette fonctionnalité. En règle générale, il est intéressant qu'un agent d'interprétation prenne ce genre d'initiative lorsqu'il est impossible de trouver une solution satisfaisante dans la situation courante.

Interactivité Le fonctionnement d'un agent d'interprétation permet d'impliquer l'utilisateur dans la gestion de composition. En particulier, il est possible de concevoir des agents d'interprétation dont les choix sont guidés par un utilisateur. Dans ce type de mécanisme, l'autonomie permise aux agents d'interprétation favorise la flexibilité du système : il n'est pas nécessaire d'imposer à l'utilisateur d'interagir avec le système de gestion de composition. En l'absence d'une participation de l'utilisateur, le système peut tout de même fonctionner, même s'il dispose alors d'informations moins précises.

De même que précédemment, il est particulièrement intéressant qu'un agent d'interprétation fasse appel à un utilisateur lorsqu'aucune solution ne peut être déterminée automatiquement.

7.4 Synthèse

Ce chapitre a présenté l'architecture d'un système de gestion de composition. Dans cette architecture, la flexibilité et l'indépendance des différents mécanismes intervenant dans la gestion de composition est garantie par l'utilisation d'une approche multi-agents. L'architecture décrite dans ce chapitre comprend ainsi plusieurs types d'agents, dont chacun est responsable de la prise en charge d'un aspect de la composition. L'architecture du système multi-agents est principalement définie par les types d'agent et les protocoles d'interaction utilisés par les agents pour interagir. Ce chapitre détaille de plus l'architecture interne des agents de résolutions, qui sont des agents génériques fournis par FCAP. Ces agents ont la capacité de résoudre collectivement le problème de satisfaction de contrainte associé à une application et d'effectuer ainsi la gestion flexible de composition. Leur tâche repose cependant sur la participation d'agents d'interprétation, spécifiques à un environnement, qui sont chargés d'interpréter des descriptions de fonctionnalités et de fournir des informations sur celles-ci. L'architecture des agents de résolution est conçue de manière à gérer les interactions avec les agents d'interprétation de manière ouverte et dynamique.

L'architecture présentée dans ce chapitre présente deux caractéristiques fondamentales pour répondre aux problématiques des applications attentives : la flexibilité d'organisation et l'ouverture.

La flexibilité d'organisation se traduit par l'absence d'une organisation prédéfinie d'un système de gestion de composition, qui serait valable pour toutes les applications. À l'inverse, l'organisation d'un système de gestion de composition dépend de l'application considérée : le nombre d'agents impliqués, leurs tâches et leurs interactions sont guidés par la description de l'application et par les caractéristiques propres de l'environnement. De plus, cette organisation peut varier au cours du fonctionnement de l'application. Ceci permet de refléter au mieux les besoins des utilisateurs et leur évolution au cours du fonctionnement.

L'ouverture se traduit par la possibilité d'intégrer dynamiquement des agents d'interprétation variés et hétérogènes. Ces agents exercent une influence importante sur la gestion de composition en fournissant des interprétations spécifiques des descriptions d'application et de fonctionnalités. En fonction des agents d'interprétation présents dans un système de gestion de composition, le comportement obtenu pour l'application peut être très variable. Ainsi, la manière dont l'application s'adapte la situation peut varier au cours du fonctionnement, afin de refléter l'évolution de l'environnement et des besoins de l'utilisateur. L'ouverture du système permet de plus d'introduire dynamiquement des comportements non prévus lors de la conception de l'application, en introduisant de nouveaux agents d'interprétation.

Les trois chapitres précédents ont détaillé l'approche FCAP pour l'adaptation générique d'applications dans les environnements attentifs. Les différents aspects de FCAP permettent de fournir une réponse adaptée aux problématiques de découplage des fonctionnalités, d'évolution de l'environnement et d'imprécision des besoins. Le chapitre suivant décrit des applications réalisées à l'aide de FCAP, qui illustrent la contribution de FCAP pour faire face à ces problématiques. Les différentes applications sont en particulier présentées graduellement, en insistant sur l'augmentation progressive des capacités de l'application et sur l'évolution de son fonctionnement.

Troisième partie

Mise en oeuvre et expérimentation

Chapitre 8

Exemples d'applications réalisées avec FCAP

Les chapitres précédents ont présenté les solutions proposées par FCAP pour répondre aux problématiques posées par les applications attentives. FCAP définit une approche générique fondée sur la composition flexible des applications.

Ce chapitre présente les expérimentations réalisées sur la base de l'intergiciel FCAP. Ces expérimentations consistent à développer et à faire évoluer des applications attentives en mettant l'accent sur la réutilisation de fonctionnalités existantes, sur la généricité des stratégies d'adaptation et sur la possibilité de modifier simplement le comportement d'une application. Ainsi, plusieurs applications sont présentées, et leur évolution en fonction des besoins est détaillée. Chaque cas comprend une présentation de l'application et une description de sa mise en œuvre dans le cadre de FCAP. La présentation de l'application s'articule en trois points :

1. La *définition des besoins* donne une description textuelle du fonctionnement souhaité de l'application. Cette description ne fait généralement pas intervenir d'informations spécifiques au contexte d'un environnement particulier.
2. La présentation des *caractéristiques de l'environnement* indique la configuration spatiale de l'environnement, les dispositifs présents dans l'environnement et les fonctionnalités proposées par ces dispositifs.
3. La présentation du *support fourni par FCAP* précise comment FCAP facilite la création de l'application souhaitée dans le cadre de l'environnement considéré.

La mise en œuvre de l'application est décrite en présentant successivement les descriptions sémantiques des fonctionnalités, la description sémantique de l'application abstraite, le problème de satisfaction de contraintes associé à l'application et le fonctionnement du système de gestion de composition.

La section 8.1 précise tout d'abord quelques éléments de l'implémentation de FCAP réalisée pour la mise en œuvre des applications. La section 8.2 présente une première application simple appelée le « diaporama ambiant ». Cette application attentive consiste à afficher un diaporama de photographies sur différents écrans d'un environnement domestique. L'évolution de l'application pour un cas mono-utilisateur, multi-utilisateur puis pour la gestion de conflits dans le cas multi-utilisateur est présentée. La section 8.3 présente une seconde application plus complexe, qui correspond au scénario détaillé dans la section 2.1.2. Il s'agit de la notification adaptée de propositions de co-voiturage. Dans cette application, des dispositifs hétérogènes de l'environnement sont utilisés pour informer un utilisateur de propositions de co-voiturage. L'évolution de l'application pour intégrer de nouveaux dispositifs et pour permettre une gestion fine du choix des dispositifs de notification est présentée.

8.1 Implémentation de l'intergiciel FCAP

Dans le cadre de la mise en œuvre des applications attentives décrites dans ce chapitre, les solutions proposées par FCAP ont été implémentées et intégrées aux éléments d'infrastructure existants. Dans cette section, nous décrivons l'architecture logicielle mise en place dans le cadre de nos expérimentations, et nous précisons l'implémentation des différents composants de l'intergiciel FCAP. La section 8.1.1 donne une présentation générale de l'architecture, la section 8.1.2 détaille l'implémentation de la gestion de descriptions et la section 8.1.3 détaille l'implémentation de la gestion de composition flexible d'application.

8.1.1 Présentation générale de l'architecture logicielle

La figure 8.1 représente globalement l'architecture logicielle utilisée pour les expérimentations. Elle est composée de trois couches :

Infrastructure de services À la base de l'architecture se trouve les infrastructures de services qui permettent d'accéder aux divers services proposés par les objets communicants de l'environnement. Dans cette architecture, plusieurs infrastructures de services existantes sont considérées. En particulier, les applications réalisées utilisent d'une part une infrastructure de services appelée Amigo, qui a été développée dans le cadre du projet Européen Amigo [1], et d'autre part une infrastructure de services appelée ZigBeeS, qui permet d'accéder à des dispositifs communicants à l'aide du protocole de communication sans fil ZigBee. Chacune de ces infrastructures de services fournit une implémentation de mécanismes de découverte et une implémentation de mécanismes de communication.

Gestion de descriptions La couche de gestion de description se trouve au milieu de l'architecture et s'appuie sur la couche d'infrastructure de services. Les composants de cette couche gèrent un espace de descriptions sémantiques, comme décrit dans la section 5.2.2. Ils implémentent en particulier les mécanismes de gestion de description décrits dans la section 5.3 : la gestion des documents contenant les descriptions et l'intégration avec les différentes infrastructures de services. Ces mécanismes permettent aux systèmes de gestion de composition de manipuler les descriptions des fonctionnalités et des applications dans un cadre distribué et dynamique.

Gestion de composition La couche de gestion de composition se situe au sommet de l'architecture et s'appuie sur la couche de gestion de descriptions. Elle est constituée par un ensemble d'agents qui forment un système multi-agents. Comme décrit dans le chapitre 7, chaque application composite flexible comprend un système de gestion de composition formé par une équipe d'agents. Au sein de cette équipe, on distingue les agents de résolution (composeurs et superviseurs), en charge d'une application donnée, et les agents d'interprétation, qui peuvent être partagés entre les applications.

Concernant les infrastructures de services, FCAP s'appuie sur des solutions existantes, dont l'implémentation n'est donc pas précisée ici. Les deux sections suivantes précisent uniquement l'implémentation des couches de gestion de descriptions et de gestion de composition.

8.1.2 Implémentation de la gestion de descriptions

Cette section décrit les deux types de composants de la couche de gestion de description : la gestion des documents contenant les descriptions et l'intégration avec les infrastructures de services.

1. http ://www.hitech-projects.com/euprojects/amigo/

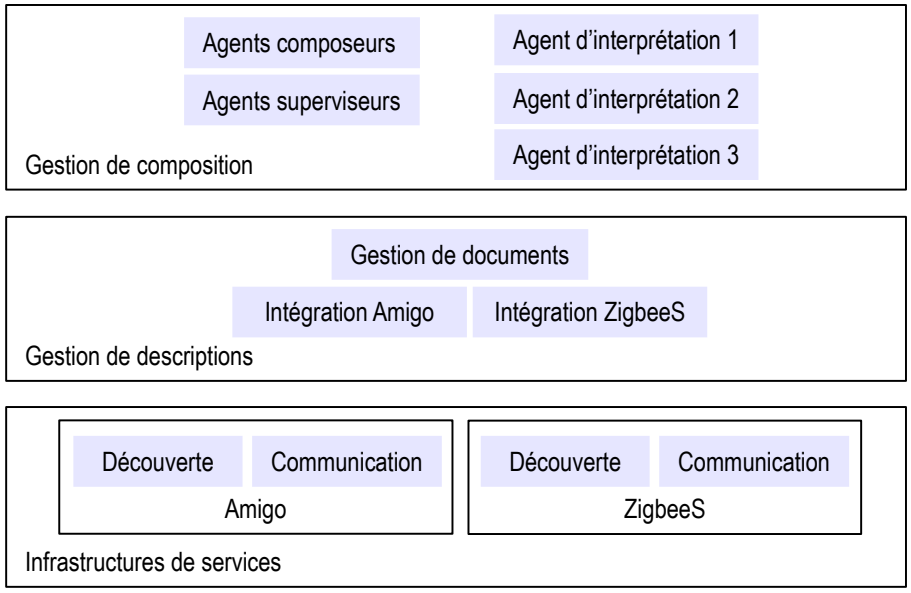

FIGURE 8.1 – Présentation de l'implémentation

a) Gestion des documents

FCAP fonctionne sur la base d'un espace de description (section 5.2.2), constitué par un ensemble de documents contenant des descriptions (de fonctionnalités, d'application, d'interprétation ...). Un module de gestion de documents (section 5.3.1) est chargé de gérer ces documents pour permettre aux agents du système de gestion de composition de manipuler les descriptions.

L'implémentation repose sur la librairie Jena [2]. Jena fournit un ensemble d'outils pour le Web sémantique, et permet en particulier de manipuler des descriptions exprimés dans les langages RDF et OWL. Jena fournit en particulier le `Model`, un objet grâce auquel il est possible de manipuler une description sémantique. Jena fournit le support pour construire un `Model` à partir d'une description exprimée en RDF.

Le `Model` sert de base à l'implémentation du concept de document de descriptions dans la section 5.2.2. Le module de gestion de documents maintient ainsi un ensemble d'objets `Model`, correspondant à un ensemble des documents de l'espace de descriptions. Il doit de plus maintenir une table des noms permettant d'associer le nom d'un document au `Model` correspondant. Enfin, le module de gestion de documents implémentent des mécanismes permettant d'accéder directement aux ressources contenues dans un document : à partir de l'identification d'une description $(\underline{r}, \underline{D})$, il retourne un pointeur vers un objet `Resource` identifié par $\underline{r}$ et contenue dans un objet `Model` identifié par $\underline{D}$.

La distribution est gérée en deux temps. D'une part, la construction d'un `Model` à partir d'une description distante est gérée directement : Jena effectue le téléchargement des données de manière transparente, lorsqu'un document n'est pas présent localement. Ainsi, les documents statiques sont hébergés sur un serveur, et accessible à tous les éléments du système (utilisent une copie exacte). Pour un document généré au cours de la composition, le module de gestion de documents fournit un mécanisme simple de publication, qui permet de rendre un document local accessible sur le réseau. L'adresse

2. http ://jena.sourceforge.net

(URL) à laquelle il est disponible est utilisée pour l'identifier. Grâce à ces mécanismes de distribution, il peut exister plusieurs modules de gestion de documents, qui gèrent chacun une partie de l'espace de description et qui peuvent s'échanger des documents.

b) Intégration d'infrastructures de services

L'intégration avec les infrastructures de services consiste à réaliser une passerelle entre une infrastructure de services et l'espace de descriptions sémantiques. Cette intégration concerne d'une part l'acquisition de descriptions sémantiques (voir section 5.3.2) et d'autre part la mise en place de configurations (voir section 5.3.3). Pour chacun de ces aspects, un adaptateur spécifique à l'infrastructure de services considérée a été développé. Ainsi, l'intégration des infrastructures de services Amigo et ZigbeeS a donné lieu à l'implémentation de 5 adaptateurs : un découvreur de services Amigo, un contrôleur de fonctionnalités Amigo, un contrôleur de connecteurs Amigo, un découvreur de services ZibgeeS et un contrôleur de connecteur Amigo-ZigbeeS.

Une plateforme OSGi [3] (*Open Service Gateway Initiative*) est utilisée pour faciliter l'intégration de différents adaptateurs pour les différentes infrastructures de services. Chaque adaptateur est réalisé sous la forme d'un *bundle* OSGi. Il est ainsi possible de gérer différentes configurations en fonction des infrastructures de services considérées, et d'ajouter des adaptateurs sans avoir à redémarrer le système. Par ailleurs, des librairies permettant d'utiliser certaines infrastructures de services (Amigo, UPnP) sont aussi disponibles sous la forme de *bundles* OSGi : nos adaptateurs s'appuient sur ces librairies pour accéder aux mécanismes de découverte et de communication.

8.1.3 Implémentation de la gestion de composition flexible

Les mécanismes de gestion de composition flexible sont présentés dans les chapitres 6 et 7. Concernant l'implémentation de ces mécanismes, on peut distinguer d'une part l'implémentation des algorithmes de gestion de composition et d'autre part l'implémentation du système multi-agents mettant en œuvre ces algorithmes. L'implémentation des algorithmes de gestion de composition découle de manière relativement directe du pseudo-code détaillé dans le chapitre 6. Cette section précise uniquement l'implémentation du système multi-agents présenté dans le chapitre 7, et indique en particulier l'infrastructure multi-agents utilisée, l'implémentation des agents de résolution (agents composeurs et superviseurs) et l'implémentation des agents d'interprétation.

a) Infrastructure du système multi-agents

L'implémentation des agents pour la gestion de composition se base sur la plateforme multi-agents Jade [4]. Pour une grande partie, Jade implémente les standards définis par la FIPA [5], en particulier le langage de communication FIPA-ACL et les services de gestion des agents. Jade fournit un environnement complet de déploiement d'un système multi-agents, et gère notamment la distribution physique des agents, les mécanismes de communication par messages, ainsi que le cycle de vie des agents.

Jade définit les primitives à partir desquelles un agent spécifique peut être implémenté. Jade fournit ainsi une classe `Agent`, qui peut être étendue pour réaliser un agent particulier. Pour les expérimentations, les deux types d'agents de résolution (composeur et superviseurs) ainsi que plusieurs types d'agents d'interprétation (en particulier les agents définis dans la section 7.3.2) ont été implémentés de cette manière.

Une particularité de notre implémentation est l'utilisation conjointe de la plateforme multi-agents Jade avec une plateforme OSGi, utilisée pour l'intégration des

3. Open Service Gateway Initiative, http://www.osgi.org
4. Java Agent Development Environment, `http://jade.tilab.org/`
5. Foundation for Intelligent Physical Agents, `http://www.fipa.org/`

différentes infrastructures de services. Pour cela, l'implémentation des différents agents est fournie sous la forme de *bundles* OSGi.

b) Implémentation des agents de résolution

Les agents de résolution possèdent une architecture d'agents rationnels, et leur comportement est défini par un ensemble de plans permettant d'atteindre leurs buts. Les plans utilisés pour les agents de résolution sont détaillés dans la section 7.2.2 (pour l'agent superviseur) et dans la section 7.2.3 (pour l'agent composeur).

Dans le cadre de la plateforme Jade, l'implémentation des agents de résolution repose sur une extension de Jade appelée JSA (*Jade Semantic Add-on*). Le JSA fournit les éléments de base pour l'implémentation d'agents rationnels capables d'interpréter des actes de communications exprimés à l'aide des langages FIPA-ACL (FIPA, 2002a) et FIPA-SL (FIPA, 2002c). FIPA-SL est un langage de contenu standardisé par la FIPA, grâce auquel il est possible d'exprimer formellement les croyances et intentions des différents agents. Le JSA permet en particulier d'implémenter un agent autour d'une base de croyances contenant des formules FIPA-SL, en définissant un ensemble de mécanismes d'interprétation pour ces formules (SIP, *Semantic Interpretation Principle*).

Chacun des plans des agents composeurs et superviseurs est implémenté sous la forme d'un SIP. Pour cela, la condition d'activation du plan est exprimée sous la forme d'une expression FIPA-SL, afin que le plan soit exécuté lorsqu'une formule FIPA-SL correspondante apparait. Pour chacun des types de condition d'activation définis dans la section 7.2.1, une expression FIPA-SL est obtenue assez simplement :

- une perception est traduite par l'ajout direct d'une croyance.
- la réception d'un message est traduite par l'ajout d'une formule FIPA-SL correspondant à la sémantique du message, obtenue par le JSA selon la spécification établie par la FIPA (FIPA, 2002b).
- l'activation d'un but est traduite par l'ajout d'une intention de l'agent.

La plupart des primitives utilisées pour former les corps de plans ont un équivalent dans le JSA (manipulation des croyances, interaction avec d'autres agents), ou sont implémentées directement par des méthodes Java (interaction avec l'environnement). Seule la manipulation des intentions donne lieu à une implémentation particulière, en particulier pour gérer les buts persistants et les buts futurs. Ces cas sont gérés par un module particulier, qui se charge d'ajouter à la base de croyance des formules décrivant les intentions lorsqu'elles sont nécessaires. Dans le cas d'un but persistant, la formule correspondante est ajoutée à la base de croyances à chaque fois que le but n'est plus satisfait. Dans le cas d'un but futur, la formule correspondante n'est ajoutée à la base de croyances qu'après écoulement du temps prévu.

c) Implémentation des agents d'interprétation

Pour les agents d'interprétation, l'implémentation est relativement simple et repose sur l'implémentation de quelques comportements (`Behaviour`) d'un agent Jade. Cela consiste généralement à réaliser les trois points suivants :

1. effectuer les échanges de messages conformément aux protocoles de communication `InterpretDescription` et `DescribeSolution` (section 7.1.2).
2. obtenir les descriptions appropriées en utilisant les mécanismes de gestion de description. Pour cela, chaque message de demande d'interprétation est lu pour extraire les couples $(\underline{r}, \underline{D})$ identifiants les descriptions considérées.
3. invoquer une fonction qui réalise l'algorithme d'interprétation considéré, en fournissant les identifiants de description en paramètre. Cette fonction utilise les mécanismes de gestion de descriptions sont utilisés pour accéder aux descriptions, puis effectue les traitements d'un algorithme de la section 6.1 .

Dans certains cas (par exemple, l'agent d'interprétation de conditions de contexte), il peut aussi être nécessaire d'implémenter l'intégration avec un système externe (par exemple, un système de gestion de contexte), ainsi que de gérer des évènements asynchrones.

8.2 Application 1 : le diaporama ambiant

Cette section présente la réalisation d'une première application simple. Cette application a été mise en place dans les laboratoires de France Telecom R&D à Grenoble.

8.2.1 Diaporama ambiant pour un utilisateur unique

a) Présentation de l'application

Définition des besoins Le diaporama ambiant est une application qui se répartit dans l'environnement et utilise les dispositifs d'interfaces à sa disposition. Il est actif en continu et présente des images sur certains écrans de l'environnement. En particulier, le diaporama suit l'utilisateur lorsqu'il se déplace, et présente les images sur l'écran le plus approprié. Au cours de son fonctionnement, l'utilisateur peut contrôler le diaporama, et ainsi arrêter le défilement des images, le démarrer ou changer la bibliothèque d'images utilisées.

Caractéristiques de l'environnement La figure 8.2 illustre l'environnement considéré pour l'application de diaporama. Cet environnement comporte trois pièces : le salon, la cuisine et le couloir. Chaque pièce est équipée d'un cadre photo numérique fixé au mur, sur lequel des images peuvent être affichées. L'utilisateur dispose aussi d'un PDA, qu'il peut utiliser pour contrôler le diaporama.

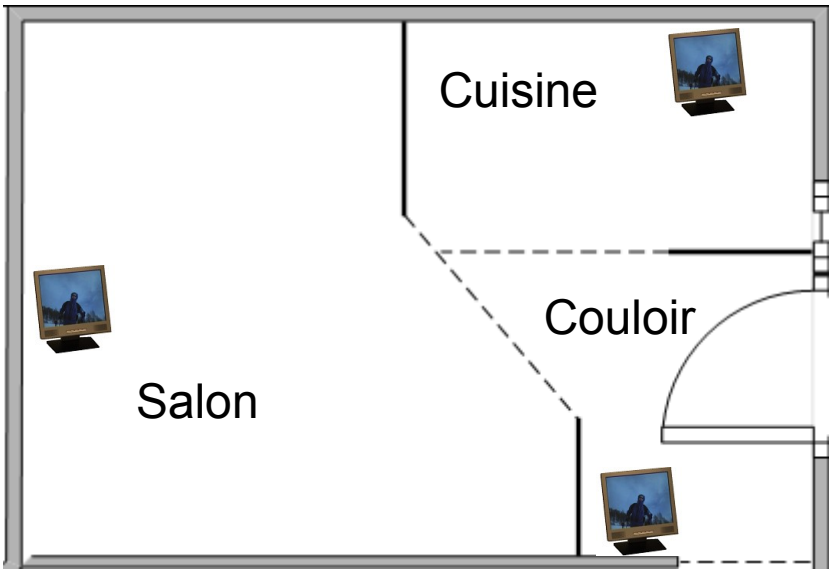

FIGURE 8.2 – Plan de l'environnement pour l'application de diaporama ambiant

L'environnement comporte ainsi trois types de fonctionnalités :

La fonctionnalité d'affichage d'images est fournie par les différents écrans de l'environnement. Elle consiste à recevoir des images numériques et à les afficher sur l'écran.

La fonctionnalité de fournisseur de photos est fournie par un serveur multi-média contenant une librairie de photos. Elle consiste à envoyer périodiquement des photos à afficher.

La fonctionnalité de commande de diaporama est fournie par le PDA de l'utilisateur. Elle consiste à fournir à l'utilisateur une interface graphique qui lui permet de commander le diaporama. Ces commandes sont par exemple démarre/arrêter le diaporama ou modifier la librairie de photos affichée.

Support fourni par FCAP Dans ce premier cas d'application simple, le support fourni par FCAP a pour principal intérêt de permettre la définition des besoins indépendamment de l'environnement particulier dans lequel l'application est déployée. Pour cela, trois aspects de FCAP entrent en jeu.

Tout d'abord, FCAP permet de *définir l'application de manière abstraite*, sans faire référence aux dispositifs particuliers de l'environnement. Il est ainsi possible de concentrer la définition sur les besoins de l'utilisateur plutôt que sur la mise en œuvre de l'application. Dans le cas présenté ici, on peut ainsi exprimer les besoins sous la forme « afficher un diaporama sur un écran proche de l'utilisateur », plutôt que « envoyer successivement la photo A puis la photo B sur le cadre photo du salon ». Cet aspect du support fourni par FCAP provient notamment de l'utilisation de descriptions sémantiques.

De plus, FCAP permet de *réutiliser des mécanismes d'interprétation génériques*, qui facilitent l'adaptation dans un contexte particulier. En particulier, on a défini dans la section 6.1 plusieurs mécanismes d'interprétation génériques, qui s'appliquent à de nombreux contextes et qui sont utilisés dans le cas présent. Ces mécanismes permettent d'éviter la définition de règles ad-hoc du type « si l'utilisateur est dans le salon, alors utiliser le cadre photo du salon ». La généricité des mécanismes d'interprétation favorise ainsi l'évolution de l'environnement : le diaporama peut continuer à fonctionner si on supprime le cadre photo du salon, ou si on le remplace par un autre.

Enfin, FCAP permet d'*utiliser directement des fonctionnalités existantes* sans qu'il soit nécessaire de développer des adaptateurs spécifiques pour les intégrer dans l'application. L'intégration est réalisée directement au niveau de l'infrastructure de services, grâce aux mécanismes de contrôleurs de fonctionnalités et de connecteurs définis dans la section 5.3.3. Le système de gestion de composition manipule ainsi une fonctionnalité de l'environnement en considérant sa description sémantique, et peut ignorer les détails de sa mise en œuvre.

b) Mise en œuvre

Descriptions sémantiques des fonctionnalités La figure 8.3 présente les documents dans lesquels sont décrites les fonctionnalités considérées :

PF-LIVING contient la description de la fonctionnalité d'affichage d'images fournie par l'écran du salon. Cette description indique que la fonctionnalité picture_displayer fournit un service display_service et précise le point d'accès Amigo pour ce service. La description fait aussi apparaître une condition sur le contexte, qui indique que l'écran concerné doit être visible par l'utilisateur. Les descriptions des fonctionnalités proposées par les cadres photo de la cuisine ou du couloir sont similaires, mais la condition de contexte porte sur les écrans de la cuisine ou du couloir.

PIC-SERVER contient la description de la fonctionnalité de fournisseur de photos. Cette description indique que la fonctionnalité picture_publisher attend le service display_service et fournit le service command_service. Elle précise un point d'accès

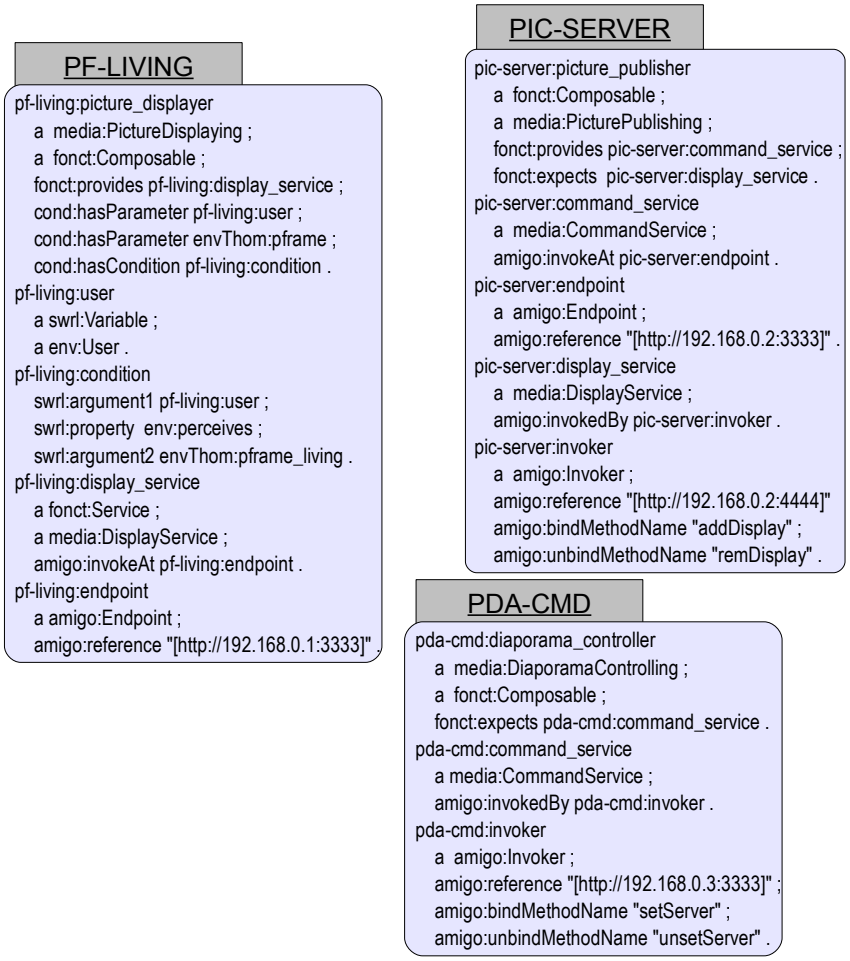

FIGURE 8.3 – Documents décrivants la fonctionnalité d'affichage d'images sur le cadre photo du salon (PF-LIVING), la fonctionnalité de fournisseur de photos (PIC-SERVER) et la fonctionnalité de commande de diaporama (PDA-CMD)

Amigo pour le service fourni et un point de connexion Amigo pour le service attendu.

PDA-CMD contient la description de la fonctionnalité de commande de diaporama. Cette description indique que la fonctionnalité pda-cmd:diaporama_controller attend le service pda-cmd:command_service et précise un point de connexion Amigo pour ce service attendu.

Dans la suite, on utilise les notations suivantes pour désigner les descriptions des fonctionnalités réelles :

- lpf = (pf-living:picture_displayer, PF-LIVING) désigne la description de la fonctionnalité d'affichage d'images réalisée par le cadre photo du salon.
- kpf = (pf-kitchen:picture_displayer, PF-KITCHEN) désigne la description de la fonctionnalité d'affichage d'images réalisée par le cadre photo de la cuisine.
- cpf = (pf-corridor:picture_displayer, PF-CORRIDOR) désigne la description de la fonctionnalité d'affichage d'images réalisée par le cadre photo du couloir.
- psv = (pic-server:picture_publisher, PIC-SERVER) désigne la description de la fonctionnalité de fournisseur de photos réalisée par le serveur multi-média.
- cmd = (pda-cmd:diaporama_controller, PDA-CMD) désigne la description de la fonctionnalité de commande de diaporama réalisée par le PDA.

DIAPO

```
diapo:ambient_diaporama
   a app:Application ;
   app:hasFunctionality diapo:displaying ;
   app:hasFunctionality diapo:publishing ;
   app:hasFunctionality diapo:controlling ;
   app:hasInteraction diapo:display_interaction ;
   app:hasInteraction diapo:control_interaction .
diapo:displaying
   a app:AbsFunctionality ;
   a media:PictureDisplaying ;
   cond:hasParameter envThom:Thom .
diapo:publishing
   a app:AbsFunctionality ;
   a media:PicturePublishing ;
   cond:hasParameter envThom:Thom .
diapo:controlling
   a app:AbsFunctionality ;
   a media:DiaporamaControlling ;
   cond:hasParameter envThom:Thom .
diapo:display_interaction
   a app:Interaction
   app:involves diapo:displaying ;
   app:involves diapo:publishing .
diapo:control_interaction
   a app:Interaction
   app:involves diapo:controlling ;
   app:involves diapo:publishing .
```

FIGURE 8.4 – Document décrivant l'application de diaporama ambiant pour l'utilisateur Thom

Description sémantique de l'application abstraite La figure 8.4 présente la description abstraite de l'application de diaporama ambiant. Cette description indique que l'application diapo:ambient_diaporama comporte trois fonctionnalités abstraites diapo:displaying, diapo:publishing et diapo:controlling. Elle indique de plus qu'il existe une interaction diapo:display_interaction entre diapo:displaying et diapo:publishing, et une interaction diapo:control_interaction entre diapo:publishing et diapo:controlling.

Dans la suite, on utilise les notations suivantes pour désigner les descriptions de fonctionnalités abstraites [6] :

- dp = (diapo:displaying, DIAPO) désigne la description de la fonctionnalité abstraite d'affichage d'images.

6. Dans tout ce chapitre, on emploie deux lettres pour noter les descriptions de fonctionnalités abstraites et trois lettres pour noter les descriptions de fonctionnalités réelles.

- pr = (diapo:providing, DIAPO) désigne la description de la fonctionnalité abstraite de fournisseur de photos.
- ct = (diapo:controlling, DIAPO) désigne la description de la fonctionnalité abstraite commande de diaporama.

Problème de satisfaction de contraintes associé à l'application FCAP associe le CSP suivant à l'application de diaporama ambiant :

- l'ensemble des variables est $X = \{dp, pr, ct\}$.
- l'ensemble des valeurs est $V = \{lpf, kpf, cpf, psv, cmd\}$.
- l'ensemble des contraintes est

$$\begin{aligned} C = \{ \quad & (\{x\}, \Delta_{TYPE})_{x \in X}, \\ & (\{x\}, \Delta_{CONTEXT})_{x \in X}, \\ & (\{dp, pr\}, \Delta_{CONNECT})\} \\ & (\{pr, ct\}, \Delta_{CONNECT})\} \end{aligned}$$

Comme la description de l'application ne mentionne pas d'interaction entre les fonctionnalités abstraites ct et dp, il n'existe pas de contraintes partagées entre ct et dp.

Dans ce cas simple, on peut détailler les coûts associés aux interprétations des différentes instanciations. La table 8.1 donne les coûts Δ_{TYPE} pour les différentes interprétations de types de fonctionnalités. La table 8.2 donne les coûts $\Delta_{CONTEXT}$ selon les situations. Ici, toutes les fonctionnalités abstraites ont le même paramètre de contexte, donc les conditions de contexte qui s'appliquent aux fonctionnalités réelles sont les mêmes pour toutes les instanciations. En revanche, l'interprétation des conditions de contexte est différente selon la localisation de Thom : dans le salon, la cuisine ou le couloir.

	dp	pr	ct	Explication
lpf kpf cpf	0	1000	1000	Pour toutes les fonctionnalités d'affichage, l'interprétation de type est la même. Toutes ces fonctionnalités ont le même type que la fonctionnalité abstraite d'affichage, mais ont des types différents des autres fonctionnalités abstraites.
psv	1000	0	1000	Le serveur photo a le même type que la fonctionnalité abstraite de publication.
cmd	1000	1000	0	Les fonctionnalité de contrôle ont le même type que la fonctionnalité abstraite de contrôle.

TABLE 8.1 – Coûts associés aux interprétations de types de fonctionnalités pour l'application de diaporama ambiant

Fonctionnement du système de gestion de composition Le système de gestion de composition associé à l'application de diaporama ambiant est décrit par la figure 8.5.

Dans la situation où Thom est dans le salon, le fonctionnement du système de gestion de composition est le suivant :

1. Le composeur reçoit la demande de composition de l'application décrite dans la figure 8.4. Pour cette application, il demande à trois superviseurs sa-dp, sa-pr et sa-ct de s'intéresser respectivement aux fonctionnalités abstraites dp, pr et ct.
2. Chaque superviseur utilise les mécanismes de découverte (section 5.3.2) pour obtenir les descriptions de fonctionnalités de l'environnement qui correspondent à la fonctionnalité qu'il supervise :

Loc.	salon	cuisine	couloir	Explication
lpf	0	1000	1000	Pour les fonctionnalités d'affichage, les conditions de contexte sont vérifiées si Thom est dans la même pièce que l'écran.
kpf	1000	0	1000	
cpf	1000	1000	0	
psv	0	0	0	La fonctionnalité de serveur photo n'a pas de condition de contexte associée.
cmd	0	0	0	La fonctionnalité de contrôle sur le PDA n'a pas de condition de contexte associée.

TABLE 8.2 – Coûts associés aux interprétations de conditions de contexte pour l'application de diaporama ambiant

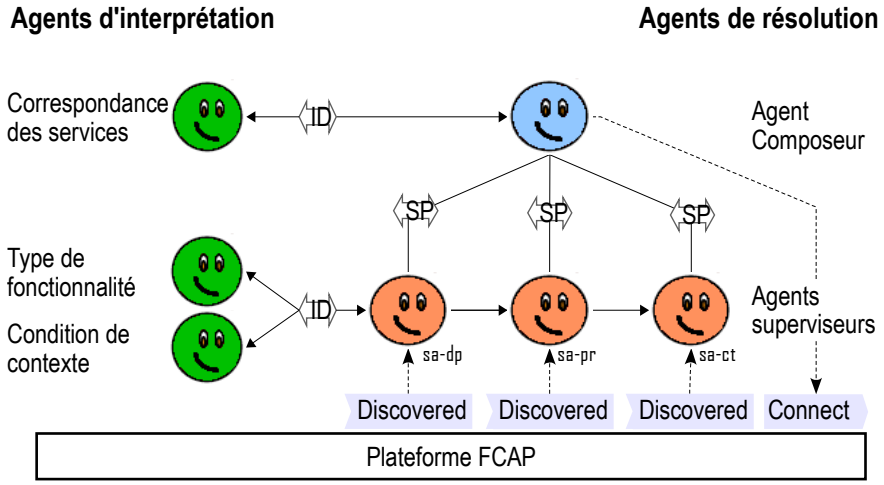

FIGURE 8.5 – Organisation du système de gestion pour l'application de diaporama ambiant

- sa-dp reçoit les descriptions kpf, lpf et cpf.
- sa-pr reçoit la description psv.
- sa-ct reçoit la description cmd.

1. Chaque superviseur collecte les interprétations pour les fonctionnalités considérées, et décide de proposer au composeur les fonctionnalités les plus appropriées :

- sa-dp propose lpf pour réaliser dp. Le cadre photo du salon est la seule fonctionnalité d'affichage possible dans le cas où Thom est dans le salon.
- sa-pr propose psv pour réaliser pr.
- sa-ct propose cmd pour réaliser ct.

1. Le composeur collecte les interprétations pour les contraintes partagées pour les instanciations $\{(lpf, dp), (psv, pr)\}$, et $\{(psv, pr), (cmd, ct)\}$.
2. Le composeur établit que la meilleure solution au problème de contrainte est l'instanciation $\{(lpf, dp), (psv, pr), (cmd, ct)\}$.
3. Le composeur obtient les descriptions contextualisées des fonctionnalités et les descriptions des connecteurs pour la configuration adaptée correspondant à la solution. Il met enfin en place la configuration, grâce aux mécanismes d'assemblage fournis par FCAP.

La configuration choisie par le système de gestion de composition évolue ensuite en fonction de la situation de l'environnement. Par exemple, si Thom se déplace dans la cuisine, les interprétations des conditions de contexte pour les fonctionnalités d'affichage du salon et de la cuisine sont modifiées. La fonctionnalité d'affichage dans la cuisine devient alors plus appropriée que celle du salon, et la configuration choisie est modifiée en conséquence.

8.2.2 Diaporama ambiant multi-utilisateur

De manière générale, un environnement attentif est destiné à accueillir plusieurs applications attentives fonctionnant en parallèle. C'est en particulier le cas lorsque plusieurs utilisateurs sont présents dans l'environnement. Dans cette section, on considère l'utilisation en parallèle de deux applications de diaporama ambiant par deux utilisateurs.

a) Présentation de l'application

Définition des besoins L'application de diaporama ambiant présentée précédemment ne prend en compte qu'un seul utilisateur. Dans le cas considéré à présent, on souhaite étendre l'utilisation à plusieurs utilisateurs. Chaque utilisateur dispose de sa propre version de l'application et affiche une bibliothèque d'images personnalisée.

Caractéristiques de l'environnement L'environnement est le même que précédemment, mais un second utilisateur, nommé Jerry, est présent. Il dispose lui aussi d'un PDA qu'il peut utiliser pour contrôler son diaporama. Pour chacun des deux utilisateurs, il doit exister une application de diaporama ambiant spécifique et personnalisée. Deux applications de diaporama ambiant fonctionnent donc simultanément dans l'environnement. Ces applications exploitent les mêmes fonctionnalités que précédemment, et doivent partager leur utilisation. Dans cet environnement, trois cas se présentent :

La fonctionnalité de commande de diaporama est fournie par les PDA de chacun des utilisateurs. Dans ce cas, chacun utilise son propre PDA et il n'y est pas nécessaire de partager les fonctionnalités.

La fonctionnalité de fournisseur de photos n'est fournie que par un unique serveur photo. Cependant, ce serveur photo peut être partagé entre plusieurs utilisateurs à l'aide d'un système de profil. Chaque utilisateur dispose d'un profil personnalisé, qui contient des librairies d'images et des préférences différentes. Ainsi, lorsque les deux diaporamas utilisent le même serveur d'images, celui-ci gère en réalité deux sessions d'utilisation distinctes qui permettent à Thom et à Jerry de contrôler leurs diaporama indépendamment.

La fonctionnalité d'affichage d'images est fournie par plusieurs écrans de l'environnement, et deux cas peuvent se présenter. Si Thom et Jerry sont dans deux pièces différentes, chacune de leur application utilise un écran différent et aucun partage n'est nécessaire. En revanche, si Thom et Jerry sont simultanément dans la même pièce, les deux applications doivent utiliser le même écran. Dans un premier temps, on ne considère pas de mécanisme destiné à gérer ce partage. Ainsi, un écran utilisé par le diaporama de Thom peut être attribué au diaporama de Jerry, ce qui a pour résultat de mélanger les photos de Thom et de Jerry.

Support fourni par FCAP Ce cas multi-utilisateur illustre comment le support fourni par FCAP permet de prendre en charge plusieurs applications simultanément. En particulier, chaque nouvelle application introduite dans le système donne lieu à la formation d'un système de gestion de composition qui lui est propre. Chaque système de gestion de composition s'organise ensuite dynamiquement. En particulier, les agents d'interprétation présents dans l'environnement peuvent participer à plusieurs systèmes de gestion de composition, en fonction de leur capacité à fournir des interprétations pour les fonctionnalités impliqués.

b) Mise en œuvre

Descriptions sémantiques des fonctionnalités Pour les fonctionnalités d'affichage d'images et de contrôle de diaporama, on considère les mêmes descriptions que dans le cas précédent (figure 8.3). Le document PF-LIVING de la figure 8.3 contient ainsi la description de la fonctionnalité d'affichage d'images fournie par l'écran du salon, et les descriptions des fonctionnalités fournies par les autres cadres photo sont similaires. Le document PDA-CMD de la figure 8.3 contient la description de la fonctionnalité de commande de diaporama pour le PDA de Thom, et celle fournie par le PDA de Jerry est similaire.

Pour la fonctionnalité de fournisseur de photos, on doit à présent considérer le service permettant l'ouverture d'une session spécifique pour chaque utilisateur. Ce service n'est pas un service fourni (au sens du modèle des fonctionnalités défini par FCAP), mais un service d'initialisation de la fonctionnalité, destiné à être utilisé directement pas le système de gestion de composition. L'accès à ce service d'initialisation est pris en charge par le contrôleur de fonctionnalité pour l'infrastructure Amigo, défini dans l'annexe B.1. Pour prendre en charge ce service, on considère à présent la description complète de la fonctionnalité de fournisseur de photos (dont certains éléments n'étaient précédemment pas utilisés par le système de gestion de composition).

La figure 8.6 donne la description complète de la fonctionnalité de fournisseur d'images. La partie gauche de la description était déjà présentée dans la figure 8.3. La partie droite décrit le service d'initialisation de session pic-server:init_service. Elle indique ainsi que ce service peut être utilisé pour initialiser une session avec le paramètre pic-server:user. Lors de l'initialisation, ce service redéfinit dynamiquement la référence du point d'accès pic-server:endpoint.

Description sémantique de l'application abstraite Une description personnalisée de l'application de diaporama ambiant est définie pour chacun des utili-

PIC-SERVER

```
pic-server:picture_publisher
   a  fonct:Composable ;
   a  media:PicturePublishing ;
   fonct:provides pic-server:command_service ;
   fonct:expects  pic-server:display_service .
pic-server:command_service
   a  media:CommandService ;
   amigo:invokeAt pic-server:endpoint .
pic-server:endpoint
   a  amigo:Endpoint ;
   amigo:reference "[http://192.168.0.2:3333]" .
pic-server:display_service
   a  media:DisplayService ;
   amigo:invokedBy pic-server:invoker .
pic-server:invoker
   a  amigo:Invoker ;
   amigo:reference "[http://192.168.0.2:4444]"
   amigo:bindMethodName "addDisplay" ;
   amigo:unbindMethodName "remDisplay" .
pic-server:init_service
   a  amigo:SessionInit ;
   amigo:inits pic-server:endpoint ;
   amigo:methodName "useUserProfile" ;
   amigo:paramName "user" ;
   amigo:paramValue pic-server:User ;
   amigo:invokeAt pic-server:initEndpoint .
pic-server:initEndpoint
   a  amigo:Endpoint ;
   amigo:reference "[http://192.168.0.2:5555]" .
pic-server:endpoint
   a  amigo:SessionEndpoint .
pic-server:picture_publisher
   cond:hasParameter pic-server:user ;
pic-server:user
   a swrl:Variable ;
   a env:User .
```

FIGURE 8.6 – Document décrivant la fonctionnalité de fournisseur de photos pour plusieurs utilisateurs

sateurs. Les deux descriptions sont similaires, mais le nom de l'utilisateur diffère. La figure 8.7 illustre ainsi la description de l'application pour Jerry, tandis que la description de l'application pour Thom reste la même que précédemment (figure 8.4).

Dans la suite, on utilise les notations suivantes pour désigner les fonctionnalités abstraites de l'application de Jerry :

- $dp' =$ (diapo-j:displaying, DIAPO-J) désigne la description de la fonctionnalité abstraite d'affichage d'images.
- $pr' =$ (diapo-j:providing, DIAPO-J) désigne la description de la fonctionnalité abstraite de fournisseur de photos.
- $ct' =$ (diapo-j:controlling, DIAPO-J) désigne la description de la fonctionnalité abstraite commande de diaporama.

Problèmes de satisfaction de contrainte associés aux applications Dans ce cas multi-utilisateurs, les applications de Thom et de Jerry sont associées à deux CSP différents et indépendants. Le CSP pour l'application de Thom est le même que précédemment. Le CSP pour l'application de Jerry est similaire :

- l'ensemble des variables est $X' = \{dp', pr', ct'\}$.
- l'ensemble des valeurs est $V' = \{lpf, kpf, cpf, psv, cmd, cmd'\}$.
- l'ensemble des contraintes est

$$\begin{aligned} C' = \{ \quad & (\{x\}, \Delta_{TYPE})_{x \in X'}, \\ & (\{x\}, \Delta_{CONTEXT})_{x \in X'}, \\ & (\{dp', pr'\}, \Delta_{CONNECT})\} \\ & (\{pr', ct'\}, \Delta_{CONNECT})\} \end{aligned}$$

Malgré leurs structures similaires, ces deux CSP ont généralement des solutions différentes car les descriptions de fonctionnalités abstraites associés aux variables sont différentes (le paramètre désignant l'utilisateur est différent) . Les interprétations de fonctionnalités ne sont pas les mêmes et l'évaluation des contraintes produit donc des solutions différentes. En particulier, les contraintes sur le contexte sont influencées par

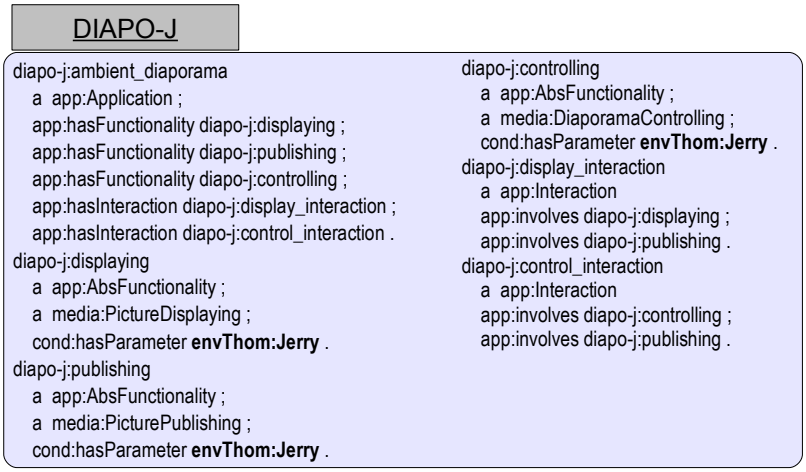

FIGURE 8.7 – Document décrivant l'application de diaporama ambiant pour l'utilisateur Jerry

l'identité de l'utilisateur.

Étant donné les similarités entre les deux CSP, on peut facilement déduire les solutions à partir du cas précédent. Par exemple, si Thom est dans le salon et Jerry est dans la cuisine, la solution du CSP pour l'application de Thom est $(lpf, dp), (psv, pr), (cmd, ct)$. La solution du CSP pour l'application de Jerry est $(kpf, dp'), (psv, pr'), (cmd', ct')$. (en notant cmd' la description de la fonctionnalité de commande de diaporama du PDA de Jerry). Si Jerry se déplace dans le salon, la solution du CSP pour l'application de Jerry est $(lpf, dp'), (psv, pr'), (cmd', ct')$, ce qui signifie que Thom et Jerry utilisent en même temps le cadre photo du salon.

Fonctionnement du système de gestion composition La figure 8.8 présente les systèmes de gestion de composition créés pour les deux applications. Chaque système possède ses propres agents de résolution, mais les deux systèmes partagent les mêmes agents d'interprétation. Ces agents effectuent cependant un traitement totalement indépendant des interprétations qu'ils fournissent, et les deux systèmes fonctionnent donc en parallèle avec un comportement et des résultats exactement identiques au cas précédent.

8.2.3 Diaporama ambiant avec gestion du partage des dispositifs

Comme on l'a vu dans la section précédente, l'utilisation de l'application de diaporama ambiant par plusieurs utilisateurs nécessite un mécanisme de gestion du partage des dispositifs. Cette section illustre comment FCAP permet d'ajouter un tel mécanisme sous la forme d'un nouvel agent d'interprétation.

a) Présentation de l'application

Définition des besoins On considère à présent une évolution de l'application de diaporama ambiant qui doit permettre une meilleure gestion du partage des dispositifs. Diverses stratégies sont envisageables pour gérer le partage de dispositifs. Dans la suite, on considère ainsi trois stratégies :

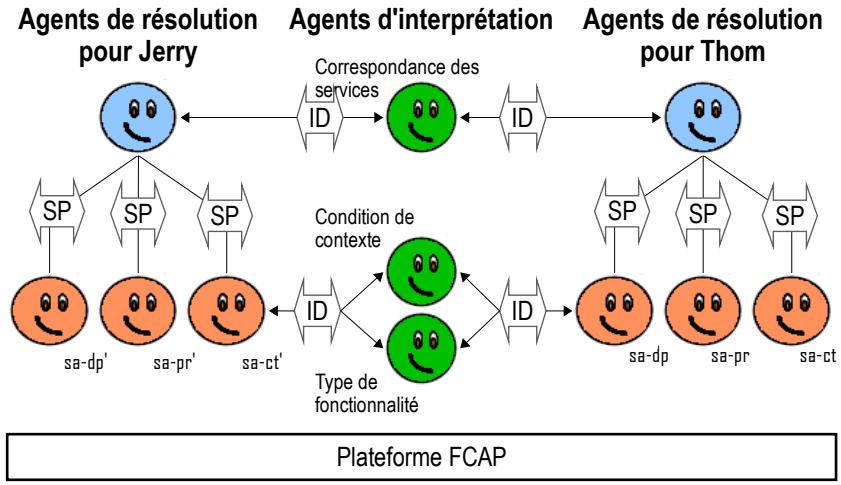

FIGURE 8.8 – Organisation des systèmes de gestion pour deux applications de diaporama ambiant

- la simple *détection d'utilisation*, qui donne la priorité au premier utilisateur d'une fonctionnalité en empêchant son utilisation pour une autre application.
- la *négociation d'utilisation*, qui permet aux systèmes de gestion de composition de définir automatiquement l'utilisation des fonctionnalités.
- la *résolution interactive*, qui rend le contrôle aux utilisateurs pour leur permettre de gérer les conflits concernant les dispositifs partagées.

Caractéristiques de l'environnement L'environnement considéré est le même que dans le cas multi-utilisateur précédent. En particulier, aucune modification des dispositifs n'est nécessaire pour permettre la gestion du partage des dispositifs. Ce nouvel aspect est entièrement géré par les systèmes de gestion de composition pour les applications de Thom et de Jerry.

Support fourni par FCAP La prise en charge du partage des dispositifs est réalisée par l'introduction d'un agent d'interprétation qui fournit des informations sur la disponibilité des dispositifs. FCAP est prévu pour permettre l'intégration de nouveaux agents d'interprétations au fur et à mesure de l'évolution de l'environnement et des besoins. Chaque nouvel agent d'interprétation influence les décisions du système de gestion de composition en proposant de nouvelles interprétations. Il peut ainsi modifier les configurations de l'application choisies en fonction des situations.

b) Mise en œuvre

Descriptions sémantiques des fonctionnalités Les descriptions sémantiques des fonctionnalités sont les mêmes que précédemment. Le document PF-LIVING de la figure 8.3 contient ainsi la description de la fonctionnalité d'affichage d'images fournie par l'écran du salon, et les descriptions des fonctionnalités fournies par les autres cadres photo sont similaires. Le document PDA-CMD de la figure 8.3 contient la description de la fonctionnalité de commande de diaporama pour le PDA de Thom, et celle fournie par le PDA de Jerry est similaire. Le document PIC-SERVER de la figure 8.6 contient

la description de la fonctionnalité de fournisseur de photos (avec prise en charge d'une session pour chaque utilisateur).

Description sémantique de l'application abstraite Les descriptions sémantiques des applications abstraites pour Thom et pour Jerry sont les mêmes que précédemment. Le document DIAPO de la figure 8.4 présente ainsi la description abstraite de l'application de diaporama ambiant pour Thom. Le document DIAPO-J de la figure 8.7 présente la description abstraite de l'application de diaporama ambiant pour Jerry.

Problèmes de satisfaction de contrainte associés aux applications Les CSP des applications sont légèrement modifiés pour introduire un nouveau type de contrainte, qui concerne la disponibilité des dispositifs. Ce nouveau type de contrainte est noté $DISPO$. Le CSP pour l'application de Thom est défini par :

- l'ensemble des variables $X = \{dp, pr, ct\}$.
- l'ensemble des valeurs $V = \{lpf, kpf, cpf, psv, cmd, cmd'\}$.
- l'ensemble des contraintes

$$\begin{aligned} C \quad = \{ \quad & (\{x\}, \Delta_{TYPE})_{x \in X}, \\ & (\{x\}, \Delta_{CONTEXT})_{x \in X}, \\ & (\{x\}, \Delta_{DISPO})_{x \in X}, \\ & (\{dp, pr\}, \Delta_{CONNECT})\} \\ & (\{pr, ct\}, \Delta_{CONNECT})\} \end{aligned}$$

Le CSP pour l'application de Jerry est défini par :

- l'ensemble des variables $X' = \{dp', pr', ct'\}$.
- l'ensemble des valeurs $V' = \{lpf, kpf, cpf, psv, cmd, cmd'\}$.
- l'ensemble des contraintes

$$\begin{aligned} C' \quad = \{ \quad & (\{x\}, \Delta_{TYPE})_{x \in X}, \\ & (\{x\}, \Delta_{CONTEXT})_{x \in X}, \\ & (\{x\}, \Delta_{DISPO})_{x \in X}, \\ & (\{dp', pr'\}, \Delta_{CONNECT})\} \\ & (\{pr', ct'\}, \Delta_{CONNECT})\} \end{aligned}$$

La contrainte $DISPO$ présente la particularité d'être évaluée dynamiquement, en fonction des résultats de la résolution des CSP. Ainsi pour un couple fonctionnalité réelle/fonctionnalité abstraite (f, a), le coût $\Delta_{DISPO}[(f, a)]$ dépend de l'emploi de la fonctionnalité réelle f dans une autre application. Si f est utilisée par une autre application, $\Delta_{DISPO}[(f, a)] = +\infty$. Sinon $\Delta_{DISPO}[(f, a)] = 0$. Cette contrainte crée ainsi une dépendance indirecte entre les deux CSP qui étaient auparavant indépendants.

Fonctionnement du système de gestion de composition La figure 8.9 présente les systèmes de gestion de composition créés pour les deux applications. Comme précédemment, les deux systèmes partagent les mêmes agents d'interprétation, mais un nouvel agent d'interprétation est introduit pour interpréter la disponibilité des dispositifs. Contrairement aux autres agents, les interprétations qu'il fournit ne sont pas indépendantes : l'interprétation qu'il fournit aux agents de résolution d'une application est corrélée à l'interprétation qu'il fournit aux agents de résolution de l'autre application.

L'agent d'interprétation de la disponibilité des dispositifs possède trois ensembles de plans :

Détection d'utilisation Cet ensemble de plans permet à l'agent de détecter qu'une fonctionnalité est déjà utilisée, et qu'elle n'est donc plus disponible. Pour obtenir des informations sur l'utilisation des fonctionnalités, l'agent obtient les informations sur les configurations mise en place auprès du système d'assemblage de FCAP. Un plan `DetectUsage` permet de déterminer quelles fonctionnalités sont

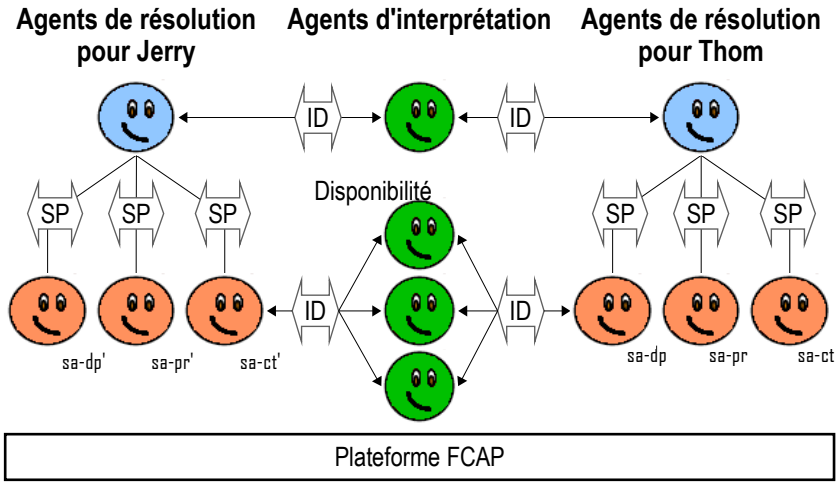

FIGURE 8.9 – Organisation des systèmes de gestion pour deux applications de diaporama ambiant, avec gestion du partage des dispositifs

en cours d'utilisation : lorsque le système d'assemble indique la mise en place d'une nouvelle configuration, ce plan met à jour les croyances de l'agent concernant les fonctionnalités en cours d'utilisation. Lorsqu'un agent de résolution demande l'interprétation d'une fonctionnalité, le plan `InterpretUsage` utilise les croyances de l'agent et fournit une interprétation de disponibilité pour cette fonctionnalité. Enfin, un plan `UpdateUsage` permet de maintenir les interprétations de disponibilité à jour : les identifiants des fonctionnalités pour lesquelles une interprétation a été demandée sont stockés, et les agents de résolution sont avertis en cas de modification de la disponibilité.

Négociation d'utilisation La négociation d'utilisation intervient lorsqu'aucune solution n'a été trouvée pour une application. Tout d'abord, un plan `CheckCompletion` permet à l'agent de détecter si une application n'a pas pu trouver de solution en raison de l'indisponibilité d'une fonctionnalité. Pour cela, le plan exploite les croyances sur les configurations mises en place, et peut vérifier si une fonctionnalité abstraite a pu être réalisée par une fonctionnalité de l'environnement. Si ce n'est pas le cas, l'agent décide de négocier l'utilisation des fonctionnalités réelles qui pourraient être utilisées mais qui ne sont pas disponibles. Un plan `NegociateUsage` permet d'effectuer cette négociation, en interrogeant les agents composeurs sur les autres possibilités dont ils disposent. Si un agent composeur accepte de cesser d'utiliser une fonctionnalité, les interprétations de disponibilité sont mises à jour et transmises aux agents superviseurs concernés.

Interaction avec l'utilisateur Dans le cas où la négociation n'est pas possible, l'agent interagit directement avec un des utilisateurs pour le laisser choisir quelle application peut utiliser la fonctionnalité. Dans le cas du diaporama, cette interaction se fait à l'aide du PDA de Thom. Si l'application de Jerry cherche à utiliser un écran, Thom reçoit une notification sur son PDA et peut décider de laisser Jerry utiliser l'écran. Les interprétations de disponibilité sont mises à jour en fonction de la décision de l'utilisateur. Ainsi, lorsque Thom décide de laisser Jerry utiliser l'écran, l'agent met à jour son interprétation, en effectuant deux étapes. D'une part, il indique aux agents de résolution de l'application de Thom que la fonctionnalité d'affichage sur l'écran n'est plus disponible. Une fois

que les agents de résolution ont pris en compte cette modification et ont cessé d'utiliser l'écran pour le diaporama de Thom, l'agent indique aux agents de résolution de l'application de Jerry que la fonctionnalité d'affichage sur l'écran est maintenant disponible. Ces agents de résolution mettent alors en place une configuration dans laquelle le diaporama de Jerry utilise l'écran.

8.3 Application 2 : notification de propositions de co-voiturage

La section 2.1.2 a présenté un scénario de notification de propositions de co-voiturage qui fait intervenir une application attentive. Grâce à cette application, nommée iRiderNotification, un utilisateur peut utiliser un système de co-voiturage et être tenu informé des voyages qui lui sont proposés à l'aide de dispositifs variés de son environnement.

Cette section détaille le fonctionnement de l'application iRiderNotification, en insistant sur l'intégration de fonctionnalités de notification variées et hétérogènes, réparties dans l'environnement. La section 8.3.2 décrit tout d'abord l'intégration de fonctionnalités basées sur des infrastructures de service différentes. La section 8.3.1 décrit la gestion de l'interopérabilité de fonctionnalités qui n'ont pas été conçues pour fonctionner ensemble. La section 8.3.3 décrit un mode de fonctionnement plus élaboré, dans lequel les fonctionnalités de notification sont choisies en fonction de la pertinence des propositions.

8.3.1 Notification nécessitant une adaptation de services

a) Présentation de l'application

Définition des besoins L'application iRiderNotification permet à un utilisateur d'utiliser un système de co-voiturage en étant tenu informé des voyages qui lui sont proposés à l'aide de dispositifs variés de son environnement. Pour cela, l'application doit donc exploiter les fonctionnalités fournies par les dispositifs de l'environnement particulier dans lequel elle s'exécute.

Caractéristiques de l'environnement Pour cette application, l'environnement est constitué d'un espace unique. Cet espace contient divers dispositifs de notification. Dans un premier temps, on considère les deux dispositifs suivants :

L'écran mural est un cadre photo accroché au mur de la pièce. En temps normal, il affiche des images, mais il dispose d'une fonctionnalité permettant d'afficher des messages destinés à l'utilisateur.

Le PDA de l'utilisateur peut être utilisé pour alerter son l'utilisateur, au moyen d'un texte qui s'affiche sur l'écran et qui est accompagné d'une sonnerie.

Ces deux dispositifs pourraient être utilisés pour notifier des propositions de co-voiturage. La fonctionnalité de notification proposée par l'écran mural, qui a été présenté dans les exemples des chapitres précédents, fournit un service directement utilisable par iRider. En revanche, la fonctionnalité d'alerte proposée par le PDA n'a pas été prévue pour fonctionner avec iRider, et ne fournit pas de service de notification directement utilisable par iRider. Le service fourni par la fonctionnalité d'alerte possède ainsi une interface légèrement différente de celle attendue par iRider.

Support fourni par FCAP Dans cette application, le support fourni par FCAP permet d'utiliser des fonctionnalités qui n'ont pas été prévues pour fonctionner ensemble. En effet, il n'est pas garanti que les fonctionnalités intéressantes pour réaliser

une application puissent interagir directement. Si elles n'ont pas été spécifiquement prévues pour fonctionner ensemble, une adaptation entre les services fournis et attendus par les fonctionnalités est souvent nécessaire.

La résolution de ce type de problème d'interopérabilité repose sur l'utilisation des descriptions de services et sur la génération de connecteurs qui permettent une adaptation. Une réalisation simple basée sur des correspondances prédéfinies entre les services est décrite dans l'annexe B.1. D'autres solutions plus élaborées proposées dans la littérature pourraient être utilisées (par exemple (Ben Mokhtar, 2007)).

b) Mise en œuvre

Descriptions sémantiques des fonctionnalités Les descriptions des fonctionnalités de iRider et de la notification sur le cadre photo ont été présentées dans les exemples des chapitres précédents. Ces fonctionnalités sont basées sur l'infrastructure Amigo, et utilisent donc le modèle de description spécifique défini dans l'annexe B (figure B.1). La figure 8.10 reproduit les descriptions complètes de ces deux fonctionnalités. Dans ces descriptions, on voit que la fonctionnalité de iRider, nommée irider:irider, attend un service irider:notif_service de type notification:MessageNotify. La fonctionnalité de notification de l'écran, nommée pframe:msg_displayer, fournit quant à elle un service pframe:display_service de type notification:MessageNotify. Pour simplifier l'exemple, on a supprimé de la description les paramètres et les conditions de contexte qui apparaissaient dans les exemples des chapitres précédents.

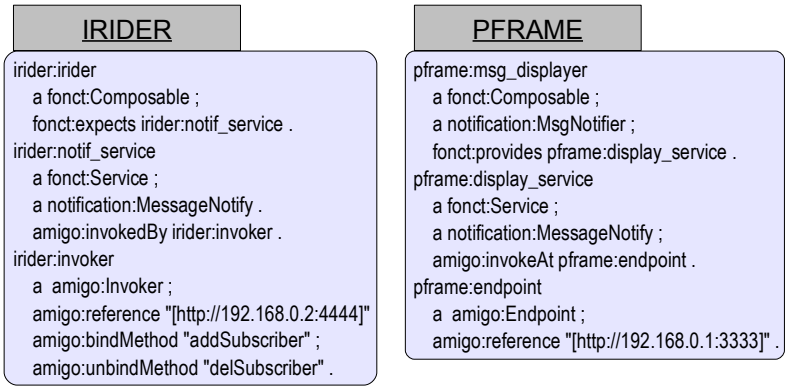

FIGURE 8.10 – Descriptions des fonctionnalités de iRider et du cadre photo

La figure 8.11 illustre la description de la fonctionnalité de notification fournie par le PDA. On voit que la fonctionnalité fournit un service pda:alert_service dont le type est notification:MessageAlert. Ce type est différent du type notification:MessageNotify attendu par iRider.

Dans la suite, on utilise les notations suivantes pour désigner les descriptions des fonctionnalités réelles :

- ird = (irider:irider, IRIDER) désigne la description de la fonctionnalité iRider.
- pfr = (pframe:msg_displayer, PFRAME) désigne la description de la fonctionnalité de notification sur le cadre photo.
- pda = (pda:alerter, PDA) désigne la description de fonctionnalité d'alerte fournie par le PDA.

Description sémantique de l'application abstraite La figure 8.12 présente la description abstraite de l'application iRiderNotification. Ici, l'application est très

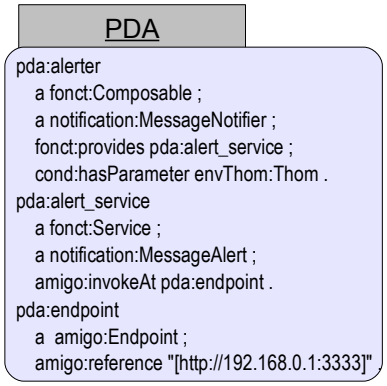

FIGURE 8.11 – Description de la fonctionnalité d'alerte du PDA

simple et se compose seulement de deux fonctionnalités qui interagissent. De plus, la fonctionnalité réelle irider:irider est une fonctionnalité spécifique à l'application, et la description indique donc directement que irider:irider réalise la fonctionnalité abstraite irnotif:sender. Ainsi, la seule inconnue est le choix d'une fonctionnalité de notification correspondant à la fonctionnalité abstraite irnotif:notifier.

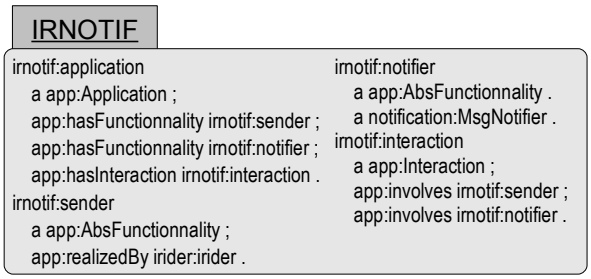

FIGURE 8.12 – Description abstraite de l'application de notification de propositions de co-voiturage (IRiderNotification)

Dans la suite, on utilise les notations suivantes pour désigner les descriptions des fonctionnalités abstraites :

- sr = (irnotif:sender, IRNOTIF) désigne la description de la fonctionnalité abstraite qui fournit des propositions.
- nt = (irnotif:notifier, IRNOTIF) désigne la description de la fonctionnalité abstraite de notification.

Problème de satisfaction de contraintes associé à l'application FCAP associe le CSP suivant à l'application iRiderNotification :

- l'ensemble des variables est $X = \{sr, nt\}$.
- l'ensemble des valeurs est $V = \{ird, pfr, pda\}$.
- l'ensemble des contraintes est

$$\begin{aligned} C \quad = \{ \quad & (\{nt\}, \Delta_{TYPE}), \\ & (\{nt\}, \Delta_{CONTEXT}), \\ & (\{sr, nt\}, \Delta_{CONNECT})\} \end{aligned}$$

On peut noter qu'il n'existe pas de contrainte locale sur la fonctionnalité abstraite qui fournit des propositions sr, car la description de l'application indique directement qu'elle est réalisée par la fonctionnalité iRider. Par ailleurs, pour la fonctionnalité abstraite de notification la contrainte sur le contexte $(nt, \Delta_{CONTEXT})$ est toujours vérifiée, car les descriptions de fonctionnalités ne précisent pas de conditions de contexte. On a donc $\forall v \in V, \Delta_{CONTEXT}[(v, nt)]) = 0$.

Selon les cas, la contrainte sur la capacité d'interaction $(\{sr, nt\}, \Delta_{CONNECT})$ est déterminée par différents agents d'interprétation. Dans le cas où on considère l'utilisation de l'écran mural pour réaliser la notification, l'interprétation est fournie par l'agent d'interprétation standard de correspondance de service (section 7.3.2). Le coût associé à cette interprétation est alors $\Delta_{CONNECT}[(ird, sr)(pfr, nt)] = 0$. Dans le cas où on considère le PDA, l'interprétation est fournie par un agent d'interprétation d'interopérabilité de services. S'il est possible de définir un adaptateur pour permettre à iRider d'utiliser le PDA, cet agent fournit une interprétation dont le coût est $\Delta_{CONNECT}[(ird, sr)(pda, nt)] = 50$. Dans le cas contraire, l'agent ne fournit pas d'interprétation : le système de gestion de composition ne dispose donc d'aucune solution pour connecter iRider et le PDA.

Fonctionnement du système de gestion composition La figure 8.13 décrit l'organisation du système de gestion de composition associé à l'application iRiderNotification.

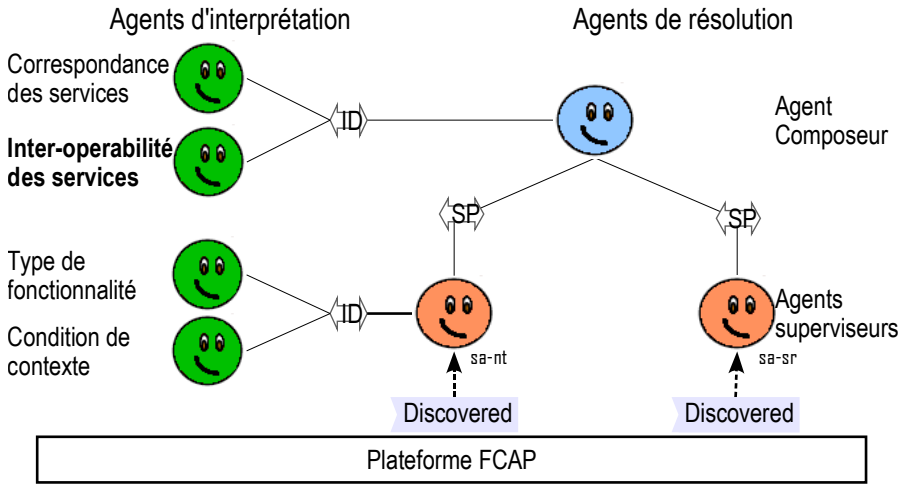

FIGURE 8.13 – Organisation du système de gestion de composition pour iRiderNotification nécessitant une adaptation de services

Un nouvel agent d'interprétation intervient pour traiter l'interopérabilité entre les services. Cet agent met en œuvre un mécanisme d'interprétation destiné à produire une description de connecteur adapté. Ce mécanisme d'interprétation effectue un traitement semblable au mécanisme d'interprétation de correspondance de service (section 6.1.3). Lorsque les services ne sont pas directement compatibles, ce mécanisme d'interprétation détermine s'il est possible de mettre en place un adaptateur permettant néanmoins aux fonctionnalités d'interagir. Pour cela, il utilise simplement un ensemble de descriptions prédéfinies des adaptations possibles pour certains services connus. Ces descriptions détaillent comment mettre en œuvre un adaptateur au moyen d'un service

d'adaptation présent dans l'environnement. Le modèle utilisé pour ces descriptions est défini en annexe B.1 (figure B.3).

La figure 8.14 décrit un connecteur résultant de l'interprétation d'interopérabilité de services. Par rapport à un connecteur standard, ce connecteur indique une correspondance entre les opérations du service fourni et celles du service attendu, et précise l'adresse d'un adaptateur capable de réaliser cette adaptation.

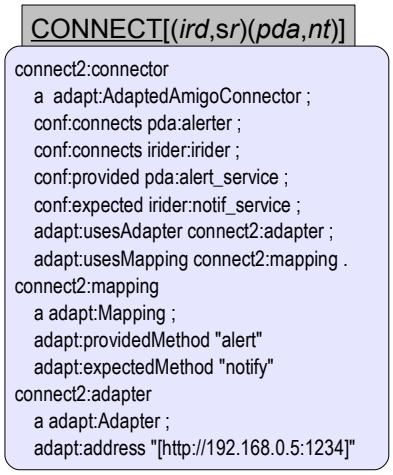

FIGURE 8.14 – Description d'un connecteur entre iRider et le PDA

Le fonctionnement du système de gestion de composition est le suivant :

1. Le composeur reçoit la demande de composition de l'application décrite dans la figure 8.12. Pour cette application, il demande à deux superviseurs sa-sr et sa-nt de s'intéresser respectivement aux fonctionnalités abstraites sr et nt.
2. Le superviseur sa-sr recherche uniquement la description de la fonctionnalité réelle fournie par iRider, car la description de l'application indique que cette fonctionnalité réalise sr. Cette description est ird.
3. Le superviseur sa-nt découvre les fonctionnalités de l'environnement qui correspondent à la fonctionnalité nt. Il obtient donc les descriptions pfr et pda.
4. Le superviseur sa-nt collecte les interprétations pour les fonctionnalités considérées. Ici, les interprétations sur les types de fonctionnalités et sur le contexte ont un coût nul, car les types de fonctionnalité sont exactement les mêmes et il n'existe pas de condition de contexte. L'utilisation des deux fonctionnalités est donc possible.
5. Le composeur collecte les interprétations sur les contraintes partagées pour les instanciations $\{(ird, sr), (pfr, nt)\}$, et $\{(ird, sr), (pda, nt)\}$. Pour l'interprétation de l'instanciation $\{(ird, sr), (pfr, nt)\}$, seul l'agent de correspondance de service fournit une interprétation $\underline{\text{CONNECT}}[(ird, sr), (pfr, nt)]$. Pour l'instanciation $\{(ird, sr), (pda, nt)\}$, seul l'agent d'interopérabilité de services fournit une interprétation $\underline{\text{CONNECT}}[(ird, sr), (pda, nt)]$.
6. Le composeur établit que la meilleure solution du problème de contrainte est l'instanciation $\{(ird, sr), (pfr, nt)\}$. Cependant, dans le cas où le cadre mural n'est pas présent, il est possible d'utiliser le PDA et la solution est alors $\{(ird, sr), (pda, nt)\}$.

8.3.2 Notification utilisant plusieurs infrastructures de services

a) Présentation de l'application

Définition des besoins La définition des besoins pour l'application iRiderNotification est la même que précédemment. L'application permet à un utilisateur d'utiliser un système de co-voiturage en étant tenu informé des voyages qui lui sont proposés à l'aide de dispositifs variés de son environnement.

Caractéristiques de l'environnement L'environnement considéré est le même que précédemment, mais un nouveau dispositif peut être utilisé pour la notification :

La lampe ambiante est une lampe colorée dont l'intensité et la teinte sont variables. Elle permet de notifier l'arrivée de messages par modulation de son intensité et de sa couleur.

Contrairement aux cas précédents, la fonctionnalité de notification fournie par la lampe ambiante ne repose pas sur l'infrastructure de services Amigo. La lampe ambiante appartient à un réseau d'objet communicants basé sur le protocole de communication sans fil Zigbee. Au sein de ce réseau, les dispositifs peuvent utiliser une infrastructure de services différente (qu'on nommera ZigbeeS), pour fournir des services donnant accès à leurs fonctionnalités.

Support fourni par FCAP Dans cette application, le support fourni par FCAP permet d'utiliser simplement des fonctionnalités qui reposent sur des infrastructures de services différentes. Dans les autres cas, toutes les fonctionnalités reposaient sur l'infrastructure de services Amigo. La lampe ambiante utilise ici une autre infrastructure de services.

L'intégration de plusieurs infrastructures de services repose notamment sur l'inclusion de plusieurs niveaux d'abstraction dans les descriptions de fonctionnalités. Ainsi, les agents de résolution prennent en compte les éléments de description générique fondés sur le modèle de description de fonctionnalité défini dans la section 5.2.1. Les agents d'interprétation prennent en compte d'autres éléments de description qui sont utiles pour leur interprétation. Enfin, le système d'assemblage prend en compte des éléments de descriptions spécifiques à une infrastructure de services particulière. Il est ainsi possible d'utiliser conjointement plusieurs infrastructures de services.

b) Mise en œuvre

Descriptions sémantiques des fonctionnalités La figure 8.15 présente la description de la fonctionnalité de notification proposée par la lampe ambiante. Cette description indique que la fonctionnalité ambiant_notifier fournit un service notif_service. Pour simplifier, on considère ici que le service fourni correspond exactement au service attendu par iRider. En revanche, la description de l'accès à ce service utilise un protocole différent de celui de l'infrastructure Amigo.

Dans la suite, on utilise les mêmes notations que dans la section précédente (8.3.1) pour désigner les descriptions des fonctionnalités réelles et on ajoute une notation pour la lampe ambiante . Pour simplifier, on ne considère que plus la fonctionnalité d'alerte fournie par le PDA :

- ird = (irider:irider, IRIDER) désigne la description de la fonctionnalité iRider.
- pfr = (pframe:msg_displayer, PFRAME) désigne la description de la fonctionnalité de notification sur le cadre photo.
- lmp = (lamp:lamp_notifier, LAMP) désigne la description de la fonctionnalité de notification de la lampe ambiante.

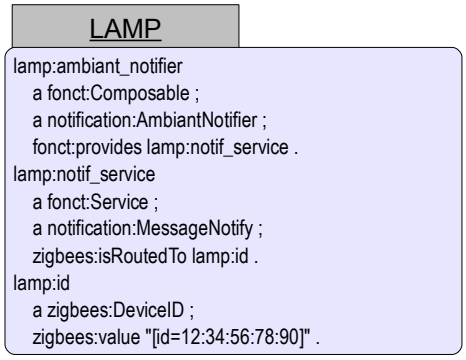

FIGURE 8.15 – Description de la fonctionnalité de notification de la lampe ambiante

Description sémantique de l'application abstraite La description de l'application est la même que précédemment (figure 8.12), et on considère les mêmes notations pour désigner les descriptions de fonctionnalités abstraites :

- sr = (irnotif:sender, IRNOTIF) désigne la description de la fonctionnalité abstraite qui fournit des propositions.
- nt = (irnotif:notifier, IRNOTIF) désigne la description de la fonctionnalité abstraite de notification.

Problème de satisfaction de contrainte associé à l'application Le CSP associé à l'application est le suivant (pour simplifier, on ne considère plus le PDA de la section précédente) :

- l'ensemble des variables est $X = \{sr, nt\}$.
- l'ensemble des valeurs est $V = \{ird, pfr, lmp\}$.
- l'ensemble des contraintes est

$$\begin{aligned} C \quad = \{ \quad & (\{nt\}, \Delta_{TYPE}), \\ & (\{nt\}, \Delta_{CONTEXT}), \\ & (\{sr, nt\}, \Delta_{CONNECT})\} \end{aligned}$$

Ce problème présente des caractéristiques similaires au cas précédent. En particulier, la contrainte sur la capacité d'interaction $(\{sr, nt\}, \Delta_{CONNECT})$ est déterminée par différents agents d'interprétation. Dans le cas où on considère l'utilisation de l'écran mural pour réaliser la notification, l'interprétation est fournie par l'agent d'interprétation standard de correspondance de service (section 7.3.2). Le coût associé à cette interprétation est alors $\Delta_{CONNECT}[(ird, sr)(pfr, nt)] = 0$. Dans le cas où on considère la lampe ambiante, l'interprétation est fournie par un agent d'interprétation pour la communication entre infrastructures de services. S'il est possible d'utiliser une passerelle qui permet à iRider d'utiliser la lampe ambiante, cet agent fournit une interprétation dont le coût est $\Delta_{CONNECT}[(ird, sr)(lmp, nt)] = 10$. Dans le cas contraire, l'agent ne fournit pas d'interprétation : le système de gestion de composition ne dispose donc d'aucune solution pour connecter iRider et la lampe.

Fonctionnement du système de gestion composition La figure 8.16 décrit l'organisation du système de gestion de composition associé à l'application iRiderNotification.

Un nouvel agent d'interprétation intervient pour permettre la connexion entre des services fournis dans des infrastructures de services différentes. En particulier, lorsqu'un agent de résolution demande une interprétation concernant des fonctionnalités

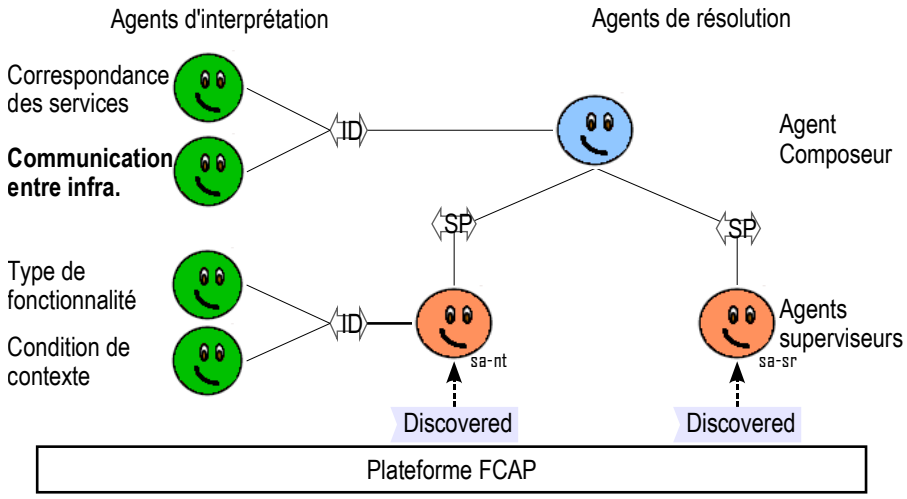

FIGURE 8.16 – Organisation du système de gestion de composition pour iRider-Notification utilisant plusieurs infrastructures de services

dont l'une fournit un service `ZigbeeS` et l'autre attend un service Amigo, cet agent d'interprétation détermine s'il est possible de faire interagir les fonctionnalités en utilisant une passerelle ZigbeeS-Amigo présente dans l'environnement. Si une telle passerelle existe, l'agent fournit une interprétation décrivant un connecteur pour utiliser cette passerelle. Le modèle utilisé pour décrire un tel connecteur est défini dans l'annexe B.2 (figure B.5). La figure 8.17 décrit un connecteur résultant de cette interprétation. Cette description précise l'adresse de la passerelle connect2:gateway qui doit être utilisée.

CONNECT[(*ird*,*sr*)(*lmp*,*nt*)]

```
connect3:connector
   a  am-zb:AmigoZigbeeConnector ;
   conf:connects lamp:ambiant_notifier ;
   conf:connects irider:irider ;
   conf:provided lamp:notif_service ;
   conf:expected irider:notif_service ;
   am-zb:usesGateway connect3:gateway .
connect3:gateway
   a am-zb:Gateway ;
   am-zb:address "[http://192.168.0.6:1234]" .
```

FIGURE 8.17 – Description d'un connecteur entre la lampe ambiante et iRider

Le fonctionnement du système de gestion de composition est tout à fait semblable au cas précédent. Lors de l'étape de collecte des interprétations sur les contraintes partagées, seul l'agent d'interprétation pour la communication entre infrastructure de services fournit une interprétation CONNECT$[(ird, sr), (lmp, nt)]$. Ainsi, dans le cas où l'écran mural n'est pas présent, il est possible d'utiliser la lampe et la solution est alors $\{(ird, sr), (lmp, nt)\}$.

8.3.3 Choix du dispositif d'interface en fonction de l'intérêt de la notification

a) Présentation de l'application

Définition des besoins Dans les cas précédents, un unique dispositif est utilisé pour la notification. L'écran mural est toujours préféré, car la fonctionnalité qu'il propose correspond exactement à la fonctionnalité recherché. S'il n'est pas disponible le PDA ou la lampe ambiante peuvent être utilisés, grâce aux mécanismes d'adaptation fournis par FCAP.

Dans cette section, on s'intéresse à un mécanisme plus élaboré, dans lequel le dispositif à utiliser dépend de l'intérêt de la proposition à notifier : si la proposition est intéressante, l'utilisateur reçoit une notification directe, en utilisant par exemple son PDA. Si elle est peu intéressante, l'utilisateur reçoit une notification périphérique, qu'il ne remarquera peut-être pas et qui ne l'interrompt pas. Une telle notification peut être produite grâce à l'écran mural ou à la lampe.

Caractéristiques de l'environnement L'environnement considéré est le même que précédemment. Il contient les trois fonctionnalités de notification proposées par les dispositifs précédemment définis : l'écran mural, la lampe ambiante et le PDA.

On considère de plus une nouvelle fonctionnalité qui permet d'organiser les propositions selon leur intérêt. Cette fonctionnalité est capable d'analyser une proposition et de déterminer la priorité avec laquelle elle doit être transmise à l'utilisateur. Elle fournit ainsi un service de notification générique qui peut recevoir des propositions venant de iRider, et elle attend deux types de services de notification : un pour les propositions très intéressantes et un pour les propositions moins intéressantes.

Support fourni par FCAP Dans cette évolution de l'application, le support fourni par FCAP permet la définition de mécanismes d'interprétation spécifiques à l'application, qui permettent une adaptation précise aux besoins de l'utilisateur. De tels mécanismes sont mis en œuvre par des agents d'interprétation spécifiques à l'application. Ces agents d'interprétation s'intègrent au système de gestion de composition de l'application et ainsi complètent ainsi les agents d'interprétation génériques précédemment définis.

Ce fonctionnement est rendu possible par la flexibilité d'organisation du système de gestion de composition. En particulier, un système de gestion de composition est un système ouvert, auquel de nouveau agents d'interprétation peuvent se joindre dynamiquement. De plus, la représentation des interprétations de fonctionnalités et les protocoles d'interaction avec les agents de résolution sont peu contraignant et permettent l'intégration d'agents d'interprétation relativement hétérogènes. Le fonctionnement d'un agent d'interprétation peut ainsi être défini de manière très spécifique à l'application.

Par ailleurs, le support fourni par FCAP prend en charge l'évolution des besoins de l'application au cours de son fonctionnement. Ainsi, lorsque les besoins sont modifiés, la description sémantique de l'application abstraite peut être redéfinie. Les modifications sont prises en compte dynamiquement par le système de gestion de composition. Celui-ci se réorganise pour satisfaire la nouvelle définition de l'application.

b) Mise en œuvre

Descriptions sémantiques des fonctionnalités La figure 8.18 présente la description de la nouvelle fonctionnalité.

Cette fonctionnalité attend deux services de notification de propositions. Pour les distinguer, la description utilise les concepts de priority:HighPriorityNotify et prior-

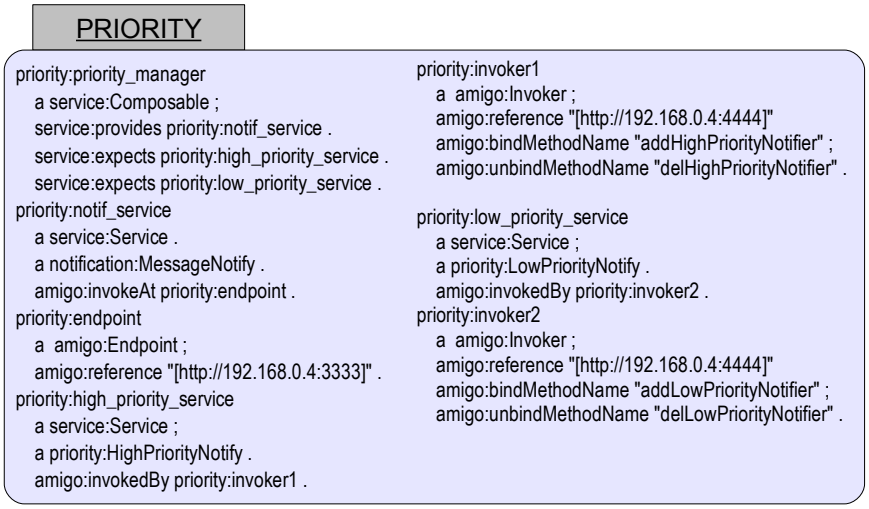

FIGURE 8.18 – Description de la fonctionnalité de notification graduelle

ity:LowPriorityNotify. Il faut noter que ces concepts sont locaux à cette fonctionnalité, et ne sont *a priori* pas utilisés dans les descriptions des fonctionnalités de notification.

Dans la suite, on utilise les notations des sections précédentes et on ajoute une notation pour la fonctionnalité de notification graduelle :

- ird = (irider:irider, IRIDER) désigne la description de la fonctionnalité iRider.
- pfr = (pframe:msg_displayer, PFRAME) désigne la description de la fonctionnalité de notification sur le cadre photo.
- pda = (pda:alerter, PDA) désigne la description de fonctionnalité d'alerte fournie par le PDA.
- lmp = (lamp:lamp_notifier, LAMP) désigne la description de la fonctionnalité de notification de la lampe ambiante.
- grd = (priority:priority_manager, PRIORITY) désigne la description de fonctionnalité d'alerte fournie par le PDA.

Description sémantique de l'application abstraite La figure 8.19 présente la description abstraite de l'application iRiderNotification avec une fonctionnalité de notification graduelle, qui gère les priorités des propositions. Cette description est une extension de la description précédente (figure 8.12) et le document GRDNOTIF importe le document IRNOTIF. Par rapport aux cas précédents, l'application comporte ainsi deux nouvelles fonctionnalités de notification irnotif:direct_notifier et irnotif:peripheral_notifier. La description de l'application indique directement que la fonctionnalité abstraite irnotif:notifier est réalisée par la nouvelle fonctionnalité priority:priority_manager.

Dans la suite, on utilise les notations suivantes pour désigner les descriptions des fonctionnalités abstraites :

- sr' = (irnotif:sender, GRDNOTIF) désigne la description de la fonctionnalité abstraite qui fournit des propositions.
- nt' = (irnotif:notifier, GRDNOTIF) désigne la description de la fonctionnalité abstraite de notification.
- dn = (grdnotif:direct_notifier, GRDNOTIF) désigne la description de la fonctionnalité abstraite qui effectue une notification directe.
- pn = (grdnotif:peripheral_notifier, GRDNOTIF) désigne la description de la fonc-

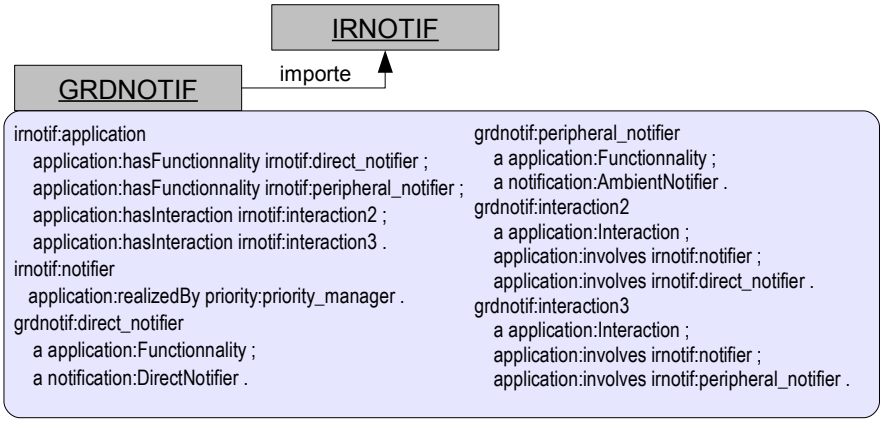

FIGURE 8.19 – Description abstraite de l'application IRiderNotification avec notification graduelle de propositions

tionnalité abstraite qui effectue une notification périphérique.

Problème de satisfaction de contraintes associé à l'application Le CSP utilisé est modifié par l'apparition de nouvelles variables et de contraintes sur ces variables :

- l'ensemble des variables est $X = \{sr', nt', dn, pn\}$.
- l'ensemble des valeurs est $V = \{ird, pfr, pda, lmp, grd\}$.
- l'ensemble des contraintes est

$$\begin{aligned} C = \{ \quad & (\{dn\}, \Delta_{TYPE}), (\{dn\}, \Delta_{CONTEXT}), \\ & (\{pn\}, \Delta_{TYPE}), (\{pn\}, \Delta_{CONTEXT}), \\ & (\{nt, dn\}, \Delta_{CONNECT}), (\{nt, dn\}, \Delta_{PRIORITY}), \\ & (\{nt, pn\}, \Delta_{CONNECT}), (\{nt, pn\}, \Delta_{PRIORITY}), \\ & (\{sr, nt\}, \Delta_{CONNECT})\} \end{aligned}$$

PRIORITY est un type de contraintes spécifique à l'application. Il permet de déterminer quels dispositifs doivent être utilisés pour effectuer une notification directe ou une notification périphérique. Ici, on considère que le PDA permet d'effectuer une notification directe, tandis que l'écran mural et la lampe ambiante permettent d'effectuer une notification périphérique. La table 8.3 donne les coûts $\Delta_{PRIORITY}$ pour les différentes fonctionnalités de notification.

Fonctionnement du système de gestion composition La figure 8.20 décrit l'organisation du système de gestion de composition associé à l'application iRiderNotification.

Par rapport aux cas précédents, le système de gestion de composition se réorganise pour répondre à la modification des besoins de l'application. Cette réorganisation se déroule selon les étapes suivantes :

1. Le composeur reçoit la nouvelle description de l'application.
2. Le composeur informe les superviseurs existants de la modification éventuelle de leur tâche. En particulier, la description précise que la fonctionnalité de notification est réalisée par la nouvelle fonctionnalité priority:priority_manager. Le superviseur sa-nt se charge donc uniquement d'obtenir la description de cette fonctionnalité.

	dn	pn	Explication
pda	0	100	le PDA permet d'effectuer une notification directe. Il peut éventuellement être utilisé pour une notification périphérique, si aucun autre dispositif n'est disponible.
pfr	100	0	L'écran mural permet d'effectuer une notification périphérique. Il peut éventuellement être utilisé pour une notification directe, si le PDA n'est pas disponible.
lmp	200	0	La lampe ambiante permet d'effectuer une notification périphérique. Elle peut éventuellement être utilisée pour une notification directe, si aucun autre dispositif n'est disponible.

TABLE 8.3 – Coûts associés aux interprétations de priorité de notification

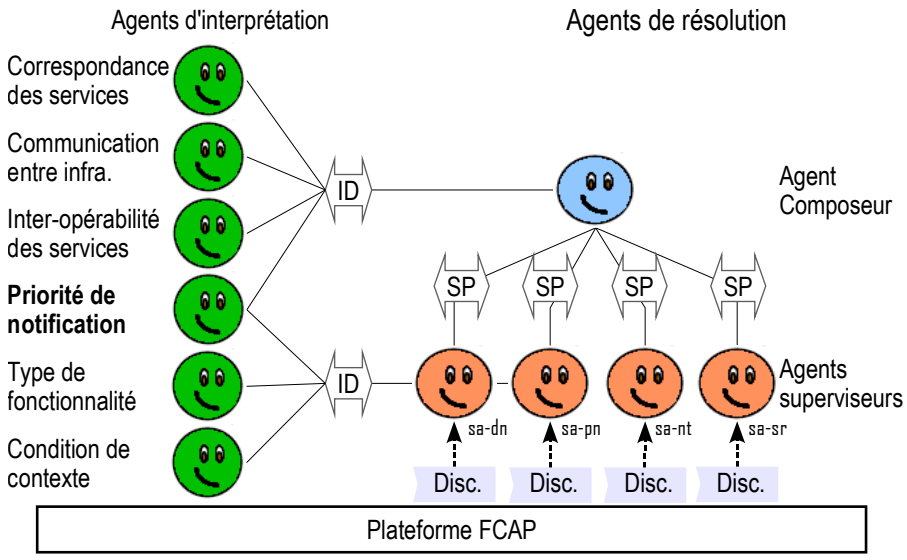

FIGURE 8.20 – Organisation du système de gestion pour iRiderNotification avec gestion des priorités de notification

3. Pour les nouvelles fonctionnalités abstraites *dn* et *pn*, le composeur recrute de nouveaux agents superviseurs sa-dn et sa-pn. Chacun de ces superviseurs lui fournit alors des propositions pour les fonctionnalités.
4. Le composeur collecte les interprétations pour les contraintes partagées pour les instanciations $\{(grd, nt), (x, dn)\}$ et $\{(grd, nt), (x, pn)\}$, avec $x \in \{pfr, pda, lmp\}$ est l'une des fonctionnalités de notification
5. L'agent d'interprétation de priorité de notification (spécifique à l'application) joint le système de gestion de composition lorsqu'il détecte que le composeur demande des interprétations concernant les instanciations $\{(grd, nt), (x, dn)\}$ et $\{(grd, nt), (x, pn)\}$, où x est l'une des fonctionnalités de notification. Il fournit alors une interprétation indiquant si la fonctionnalité de notification proposée est autorisée ou non.
6. Le composeur détermine une solution du CSP en tenant compte des interprétations fournies par les différents agents d'interprétation. En particulier, l'agent d'interprétation de priorité de notification permet de déterminer quelle fonctionnalité utiliser pour la notification directe et pour la notification périphérique.

8.4 Intérêt du support fournit par FCAP

Les expérimentations décrites dans ce chapitre permettent de mettre en évidence l'intérêt du support fourni par FCAP pour faire face aux problèmes de conception des applications attentives. Cette section détaille les solutions fournies par FCAP pour faire face aux différents problèmes mis en évidence dans la section 2.2 .

8.4.1 Intérêt concernant le découplage des fonctionnalités

La problématique de découplage des fonctionnalités mise en évidence dans la section 2.2.1 apparait dans l'ensemble des applications réalisées. FCAP permet de traiter ces problèmes, notamment grâce à l'utilisation d'une approche-orientée service et de descriptions sémantiques de fonctionnalités :

Distribution La distribution apparaît dans toutes les applications. Les problèmes associés sont en grande partie pris en charge au niveau des infrastructures de services. L'intérêt de FCAP dans ce domaine est donc la possibilité de s'appuyer sur ces infrastructures existantes et de bénéficier ainsi des solutions qu'elles apportent pour gérer la décentralisation, la sécurité et la fiabilité des communications.

Réutilisabilité Le problème de la réutilisabilité apparaît aussi dans toutes les applications. D'une part, les applications ont été développées principalement à partir de fonctionnalités existantes. D'autre part, certaines fonctionnalités telles que celles du cadre photo peuvent être utilisées dans plusieurs applications. Ce problème est en partie traité par l'utilisation d'une approche-orientée service, mais FCAP augmente la réutilisabilité en intégrant aux descriptions des fonctionnalités une représentation explicite du contexte.

Hétérogénéité Deux types d'hétérogénéité apparaissent dans les applications présentées : l'hétérogénéité des fonctionnalités (section 8.3.1) et l'hétérogénéité des infrastructures de services (section 8.3.2). FCAP traite l'hétérogénéité des fonctionnalités par l'utilisation de descriptions sémantiques, grâce auxquelles il est possible d'utiliser des fonctionnalités qui n'ont pas été prévues pour une application précise et qui n'ont pas été prévues pour fonctionner ensemble. FCAP traite de plus l'hétérogénéité des infrastructures de services en rendant possible l'intégration de différentes infrastructures de services.

8.4.2 Intérêt concernant la robustesse des applications

La problématique de robustesse des applications mise en évidence dans la section 2.2.2 est généralement présente dans les applications réalisées.

Déploiement Toutes les applications présentées sont définies de manière abstraite et déployées dynamiquement dans un environnement donné. FCAP exploite notamment les mécanismes de découverte et d'assemblage de fonctionnalités proposés par les infrastructures de services existantes.

Dynamicité Le problème de la dynamicité est principalement illustré dans le cas du diaporama ambiant (section 8.2). En particulier, l'application est reconfigurée dynamiquement pour utiliser l'écran le plus approprié en fonction de la localisation de l'utilisateur. La reconfiguration est prise en charge par le système de gestion de composition de l'application. Ces mécanismes s'appuient en particulier sur des plans qui permettent aux agents de réagir aux modifications de l'environnement.

Pannes La tolérance aux pannes n'est pas spécifiquement détaillée mais elle apparaît cependant dans le cas de l'application iRiderNotification. En particulier, nous avons mentionné que lorsque la fonctionnalité de notification la plus appropriée disparaît ou tombe en panne, il est possible d'utiliser une fonctionnalité alternative (le PDA ou la lampe ambiante). De manière générale, le fonctionnement du système de gestion de composition est conçu de manière à rechercher dynamiquement de nouvelles solutions en cas de panne.

Évolution L'évolution apparaît dans le cas de l'application iRiderNotification, dans laquelle de nouvelles fonctionnalités de notification apparaissent progressivement. Ces fonctionnalités sont prises en charge par FCAP et intégrées sans modification de la description sémantique de l'application abstraite. FCAP peut ainsi prendre en charge l'évolution d'un environnement attentif sans nécessiter de modification interne. Cette capacité repose sur la présence d'agents d'interprétation capables d'interpréter les descriptions des nouvelles fonctionnalités. En général, les agents d'interprétation génériques sont suffisants. Dans certains cas, il est nécessaire d'introduire de nouveaux agents d'interprétation capables de prendre en charge des fonctionnalités particulières.

8.4.3 Intérêt concernant l'adaptabilité à des besoins imprévisibles

La problématique d'adaptabilité à des besoins imprévisibles mise en évidence dans la section 2.2.3 apparaît dans certains cas, même s'ils sont relativement restreints dans le cas de ces applications simples.

Personnalisation Le problème de la personnalisation apparaît dans tous les cas présentés. En particulier, une application doit toujours être définie de manière personnalisée pour un utilisateur. La section 8.2.3 illustre de plus la possibilité de modifier dynamiquement les besoins.

Délégation Le problème de la délégation est présent dans tous les cas présentés, puisque FCAP permet au système de prendre certaines décisions concernant la composition de l'application de manière autonome.

Contrôle L'équilibre autonomie/contrôle apparaît principalement dans le cas de la gestion du partage des dispositifs (section 8.2.3). Dans ce cas, le système peut rendre le contrôle à l'utilisateur lorsqu'il n'est pas capable de prendre une décision de manière autonome. L'équilibre autonomie/contrôle est principalement favorisé par l'utilisation d'une architecture multi-agents et par la séparation entre les agents de résolution et les agents d'interprétation. Cette distinction

donne les moyens aux utilisateurs, représentés par des agents d'interprétation, d'influencer le fonctionnement des mécanismes de gestion de composition.

Interférences La gestion des interférences entre les applications apparaît dans le cas du diaporama multi-utilisateur (sections 8.2.2 et 8.2.3). Lorsque deux applications sont présentes dans l'environnement, des conflits apparaissent notamment concernant l'utilisation des dispositifs physiques. FCAP permet de traiter ces interférences en les intégrant au mécanismes de gestion de composition à l'aide d'interprétations mises à jour dynamiquement.

8.5 Synthèse

Ce chapitre a présenté plusieurs exemples d'applications qui ont pu être mises en œuvre en utilisant le support fourni par FCAP. Les applications considérées permettent d'illustrer les différentes problématiques de conception des applications attentives. En particulier, l'application de diaporama ambiant illustre la dynamicité de l'environnement et la gestion des interférences. L'application de notification de propositions de co-voiturage illustre l'intégration de fonctionnalités hétérogènes et l'évolution des besoins de l'utilisateur.

Ces expérimentations ont démontré l'utilité des solutions fournies par FCAP pour faire face à la plupart des problématiques de conception dégagées dans le chapitre 2. Concernant le découplage des fonctionnalités, on a en particulier montré que FCAP facilite l'intégration de fonctionnalité hétérogènes. Concernant la robustesse des applications, on a montré que FCAP permet d'adapter dynamiquement une application non seulement aux changements prévisibles (dynamicité due au déplacement d'un utilisateur pour le diaporama ambiant) mais aussi à l'évolution imprévue de l'environnement (évolution des dispositifs de notification disponibles). Concernant l'adaptabilité à des besoins imprévisibles, on a vu que FCAP permet notamment l'implication des utilisateurs dans la résolution des interférences ainsi que la définition de mécanismes personnalisés spécifiques (cas du choix de dispositif en fonction de l'intérêt de la notification).

Quatrième partie

Conclusion et perspectives

Chapitre 9

Conclusion et perspectives

Nos travaux se sont intéressés aux problèmes liés à la conception d'applications dans les environnements attentifs. En raison de leur intégration dans la vie quotidienne, les environnements attentifs présentent de nombreux défis, à la fois technologiques, économiques et sociologiques. Nous nous sommes principalement intéressés aux défis technologiques, en gardant à l'esprit les autres dimensions.

Nous avons mis en évidence trois grandes problématiques de conception d'applications attentives :

- La problématique de *découplage des fonctionnalités* est une conséquence de la diversité des fonctionnalités mise en œuvre dans le cadre d'un environnement attentif. Cette diversité nécessite l'implication de différents types d'acteurs dans la conception d'un environnement attentif. Ainsi, il n'est pas possible de considérer que les différentes fonctionnalités nécessaires pour le fonctionnement d'une application attentive ont été spécifiquement conçues pour fonctionner ensemble, ni qu'elles seront disponibles à tout moment lors du fonctionnement de l'application.
- La problématique de *robustesse des applications* est essentielle dans un domaine où les applications fonctionnent sur de longues périodes et interagissent avec l'environnement physique. En particulier, on doit considérer que les applications attentives doivent s'adapter dynamiquement à l'évolution continuelle de l'environnement.
- La problématique d'*adaptabilité à des besoins imprévisibles* est une caractéristique très particulière des applications attentives. En raison du fonctionnement des applications sur de longues durées et de leur utilisation dans le cadre d'activités de la vie quotidienne dont les objectifs ne sont pas formellement définis, il est impossible de définir de manière définitive les besoins des utilisateurs. Les applications doivent ainsi pouvoir prendre en compte l'évolution de ces besoins.

Pour répondre aux problèmes posés par la conception des applications attentives, il semble intéressant de considérer une approche de composition dynamique d'application, dans laquelle une application est assemblée automatiquement en fonction des besoins. Dans les travaux présentés ici, nous nous sommes plus particulièrement intéressés à une approche de composition flexible d'application, dans laquelle une application n'est pas considérée comment une entité monolithique prédéfinie, mais comme un ensemble de fonctionnalités dont les interactions sont définies en fonction des besoins des utilisateurs et de la situation de l'environnement.

Nous avons ainsi défini un intergiciel pour la gestion de composition flexible d'applications. Cet intergiciel est destiné à faciliter la conception d'applications attentives en prenant en charge une partie des problématiques dégagées précédemment. Son fonc-

tionnement repose sur la manipulation de descriptions sémantiques des fonctionnalités et des applications. Deux types de mécanismes sont impliqués dans cette manipulation de descriptions. Les mécanismes d'interprétation sont des mécanismes spécialisés qui effectuent un raisonnement sur les descriptions sémantiques et fournissent des informations spécifiques à un domaine particulier. Le mécanisme de résolution est un mécanisme générique qui agrège les contributions des mécanismes d'interprétation pour déterminer comment composer une application en fonction de la situation. Pour chaque application attentive, ces mécanismes sont mis en œuvre au sein d'un système multi-agents appelé système de gestion de composition.

Enfin, nous avons validé l'intergiciel proposé en le mettant en œuvre pour concevoir plusieurs applications attentives. Les applications considérées illustrent les principales problématiques de conception des applications attentives, en particulier l'intégration de fonctionnalités hétérogènes, l'évolution de l'environnement et l'imprécision des besoins des utilisateurs. Les expériences menées indiquent comment l'intergiciel proposé facilite la prise en charge générique de ces problèmes.

Contribution Dans le cadre de nos travaux, nous avons ainsi pu montrer qu'un *intergiciel pour la composition flexible d'application* permet de prendre en charge de manière générique un grand nombre de problèmes liés à la conception d'applications attentives. Nos travaux constituent ainsi une avancée vers la définition d'une infrastructure générique pour les environnements attentifs, qui doit permettre de faire face aux défis technologiques, économiques et sociologiques présentées par les évolutions des systèmes informatiques vers des systèmes d'intelligence ambiante.

Notre contribution porte en particulier sur trois points :

1. Nous avons mis en évidence et organisé un ensemble de problèmes auxquels les concepteurs d'applications attentives doivent répondre. Nous avons ainsi dégagé les trois grandes problématiques de conception des applications attentives.
2. Nous avons défini les principes et les mécanismes fondamentaux d'un intergiciel pour la composition flexible d'applications. En particulier, nous avons proposé des mécanismes de gestion de composition flexible d'application autorisant la prise en charge de l'évolution de l'environnement et des besoins.
3. Nous avons validé l'intergiciel proposé en effectuant des expérimentations pratiques de conception d'applications attentives dans un environnement physique (non simulé). Ces expérimentations illustrent comment notre intergiciel permet de prendre en charge une grande partie des problèmes de conception des applications attentives.

Limitations Malgré les résultats obtenus, nous pouvons souligner quelques limitations à nos travaux. Une première limitation est l'absence d'une méthode exacte et de critères bien définis permettant de garantir les propriétés des applications conçues selon l'approche de composition flexible d'applications. Nous n'avons pas cherché à lever cette limitation dans le cadre de nos travaux, car nous avons considéré qu'il ne s'agissait pas d'une limitation liée à notre solution, mais d'une limitation intrinsèque au domaine auquel nous nous intéressons. En particulier, nous considérons qu'il n'est pas possible de définir *a priori* un critère global qui permettrait d'évaluer la pertinence des choix effectués par un système tel que FCAP pour la composition d'une application (par exemple le choix d'un dispositif pour notifier un utilisateur). La pertinence des choix de composition dans un environnement attentif est en effet un critère subjectif, qui dépend de l'application, de l'utilisateur et de son contexte, pour lesquels il n'est pas possible de définir un modèle exact. Ce domaine est en particulier très différent du domaine de composition des services Web, dans lequel il est généralement possible de considérer des critères objectifs (coût, temps d'attente ...) qui sont satisfaisant dans le cadre d'application aux besoins bien définis (par exemple trouver un vol au meilleur

prix). Dans le cadre des environnements attentifs, nous avons ainsi préféré la mise en place d'un système flexible qui permet à l'utilisateur d'influencer et de corriger les erreurs du système. La conception d'une application devient ainsi un processus interactif qui implique l'utilisateur, plutôt qu'un processus complètement automatique.

Une seconde limitation est le manque d'évaluation auprès d'un nombre suffisant d'utilisateurs potentiels (concepteurs d'applications attentives). La conception d'applications dans le cadre de nos expérimentations n'a en effet impliqué que quelques membres de nos équipes de recherche, et n'a donc pas permis d'évaluer précisément l'utilisabilité de nos solutions par des développeurs non experts. Afin de compenser ces lacunes, nous avons cependant veillé à exploiter autant que possible des concepts et des solutions issus de l'état de l'art, en particulier dans le cadre des architectures orientées service. Cette approche favorise l'adoption de nos solutions dans un cadre déjà bien établi. Cette approche s'est révélée essentielle dans la réalisation de nos expérimentations, puisqu'il a été possible d'utiliser des fonctionnalités développées indépendamment de nos applications, pour lesquelles aucune connaissance spécifique à notre système n'est requise.

Perspectives Afin de répondre aux limitations exposées précédemment, une perspective essentielle est la mise à disposition de l'intergiciel proposé pour permettre son utilisation par des concepteurs d'application non experts. Cette mise à disposition permettrait d'évaluer plus précisément l'intérêt de l'intergiciel proposé pour la conception d'applications variées. Parallèlement à la mise à disposition, l'adoption de ces travaux pourrait donner lieu à l'intégration d'un certain nombre d'autres contributions, en particulier :

- L'intégration de différentes infrastructures de services, ainsi que de modèles de descriptions sémantiques associés. Dans le cadre de nos expérimentations, nous nous sommes principalement basés sur deux infrastructures de services qui étaient à notre disposition.
- L'intégration de nouveaux agents d'interprétation permettant de nouvelles stratégies d'adaptation. En particulier, la littérature propose de nombreuses solutions concernant l'interopérabilité de services et la génération d'adaptateurs sur la base sur de descriptions sémantiques. De telles solutions pourraient facilement être intégrées à FCAP.
- Dans le cadre des systèmes multi-agents qui constituent le cœur de FCAP, de nombreuses contributions existantes pourraient être intégrées afin d'améliorer la flexibilité et l'ouverture du système. En particulier, on a ici considéré des mécanismes de décision relativement simples, basés sur la résolution répétée d'un problème de satisfaction de contraintes. Cette solution s'est révélée suffisante dans le cadre de nos expériences, mais il serait possible d'envisager l'utilisation de mécanismes de décisions plus élaborés. Par ailleurs, dans le cadre d'un système multi-agents ouvert, dans lequel de nouveaux agents d'interprétation peuvent apparaitre et fournir des informations, un apport intéressant serait l'intégration de mécanismes de gestion de confiance, permettant au système de mieux gérer le choix des informations à prendre en compte.

Cinquième partie

Annexes

Annexe A

API du système de gestion de descriptions distribuées 3DB

Le système de gestion de descriptions est décrit dans la section 5.3.1. Il constitue l'un des éléments du support pour la manipulation de descriptions. Le système de gestion de descriptions est utilisé par la plupart des autres éléments de FCAP. Cette annexe décrit l'API du système de gestion de description. Les fonctions définies ici sont notamment utilisées dans les algorithmes des mécanismes d'interprétation de fonctionnalités (section 6.1).

L'API indique les méthodes proposées par une base de descriptions `3db` en précisant les paramètres, leurs types et les valeurs retournées. En plus des types de données primitifs, on emploie les types `Ressource` et `Document`, qui désignent respectivement une URI de ressource et une URL de document [1].

Gestion de document Les fonctions suivantes permettent de gérer le chargement de documents existants dans une base de descriptions et la création de nouveaux documents.

`3db.load(d : Document)` chargement d'un document. Cette méthode prend en paramètre un identifiant de description, composé de l'URI de la ressource décrite et de l'URL du document support. La base de descriptions charge le document ainsi que ses dépendances dans sa base de connaissances.

`3db.createDoc(share : boolean)` → `d : Document` création d'un document. Cette méthode a pour effet de créer un nouveau document vide et de le charger dans la base de connaissances. Elle retourne ensuite une URL qui permet d'identifier ce document. Elle prend un paramètre qui indique si le document doit être partagé ou non.

`3db.addImport(d : Document, e : Document)` ajout d'une déclaration d'importation. Cette méthode prend en paramètre l'URL d'un document *doc* et l'URL d'un document à importer *import*, et ajoute une déclaration indiquant que *doc* importe *import*. Toutes les informations contenue dans *import* sont alors accessible lors du chargement de *doc*.

1. On peut considérer que les URI et URL sont employés comme des pointeurs

Interrogation de la base de descriptions Les fonction suivantes permettent d'interroger une base de descriptions.

`3db.ask(s : Ressource, p : Ressource, o : Ressource)` → `t : boolean` recherche du triplet (s p o) dans la base de connaissances. Cette méthode retourne `vrai` si le triplet est présent dans la base de connaissances, et `faux` sinon.

`3db.select(s : Ressource, p : Ressource, o : Ressource)` recherche du triplet (s p o) dans la base de connaissances, en laissant certains éléments du triplet indéterminés (ce sont des variables). Cette méthode est ainsi utilisée pour naviguer dans les connaissances. Dans ce cas, l'opération retourne une instanciation correcte pour les variables.

`3db.selectAll(s : Ressource, p : Ressource, o : Ressource)` fonctionne comme `select`, mais fournit toutes les instanciations contenues dans la base de connaissances.

Écriture de descriptions Une fonction permet d'écrire de nouvelles informations dans une base de descriptions

`3db.write(doc : Document, s : Ressource, p : Ressource, o : Ressource)` ajout d'un triplet dans un document. Cette opération prend en paramètres l'URL d'un document et un triplet (s p o) à ajouter dans le document.

Annexe B

Détails de l'intégration d'infrastructures de services existantes

L'intégration d'infrastructures de services existantes est l'un des aspects importants de FCAP. La section 5.3 a ainsi définit les notions de médiateur de découverte, de contrôleur de fonctionnalité et de contrôleur de connecteur grâce auxquelles FCAP utilise des infrastructures de services diverses.

Cette annexe détaille l'intégration de deux infrastructures de services utilisées dans le cadre de nos expérimentations. La section B.1 porte sur l'intégration de l'infrastructure de services pour les environnements domestiques développée dans le cadre du projet européen Amigo (appelée infrastructure de services Amigo). La section B.2 porte sur l'intégration d'une autre infrastructure de services, basée sur le protocole de communication sans fil Zigbee. Cette infrastructure de services permet à des objets communicants de faible capacités de proposer des services dans la sphère domestique.

B.1 Intégration de l'infrastructure de services Amigo

L'infrastructure de services Amigo est destinées à permettre l'interconnexion de dispositifs issues de quatre domaines : dispositifs mobiles (téléphone, PDA), dispositifs informatiques (ordinateur, imprimante), dispositifs multimédia (télévision, appareil photo numérique) et appareils électroménagers. Le fonctionnement de cette infrastructure de services est en particulier décrite dans (Georgantas & Thompson, 2007).

B.1.1 Modèle de description sémantique

La première étape de l'intégration d'une infrastructure de services est la définition d'un modèle de description des services réalisés dans cette infrastructure de services. Un tel modèle permet d'exprimer le lien entre le modèle de description de fonctionnalité de FCAP (5.2.1) et les fonctionnalités qui proposent des services dans cette infrastructure de services.

La figure B.1 présente le modèle de description adopté pour décrire les services Amigo :

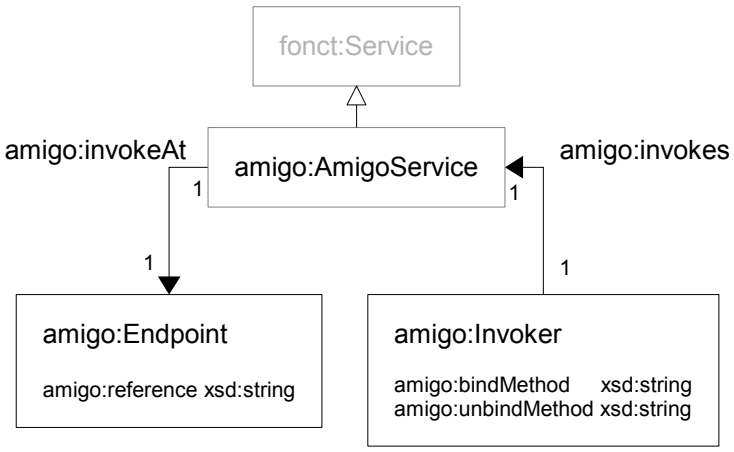

FIGURE B.1 – Modèle de description pour les services Amigo

amigo:Endpoint représente un point d'accès à un service Amigo. Ce point d'accès est caractérisé par une référence Amigo, spécifiée à l'aide de la propriété reference. Dans l'infrastructure Amigo, chaque service possèdent en effet une référence décrite sous forme d'une chaîne de caractères (Georgantas & Thompson, 2007). Cette référence contient notamment l'adresse réseau à laquelle le service est accessible.

amigo:Invoker représente un point de connexion à un service Amigo. Ce point de connexion est lui même un service Amigo particulier, qui propose des méthodes pour fournir la référence d'un service Amigo auquel se connecter. Il est caractérisé les noms des méthodes qui permettent de connecter/déconnecter un service en fournissant sa référence Amigo. Comme les autres services, il possède un point d'accès, pour lequel la description indique une référence Amigo.

Le modèle précédent permet de décrire uniquement une fonctionnalité qui ne nécessite pas d'initialisation. Dans certains cas, une fonctionnalité peut fournir des services non fonctionnels, destinée à configurer son fonctionnement avant l'utilisation. C'est par exemple le cas du serveur multi-média présenté dans la section 8.2.2. La figure B.2 décrit une extension du modèle précédent destinée à décrire l'initialisation d'une session d'utilisateur.

amigo:SessionEndpoint représente un point d'accès défini dynamiquement lors de l'initialisation d'une session. Il s'agit donc d'une sous-classe des points d'accès définis précédemment.

amigo:SessionInit représente un service d'initialisation de session. Il s'agit d'un service Amigo spécialisé pour lequel une méthode d'initialisation de session est précisée. Pour les besoins de nos expérimentations, on considère que l'initialisation d'une session est réalisée simplement par l'invocation d'une méthode prenant un unique paramètre (généralement le nom de l'utilisateur). La description défini donc la procédure d'initialisation de session en indiquant uniquement le nom de la méthode (amigo:methodName) et le nom du paramètre à fournir (amigo:paramName), et la valeur du paramètre (amigo:paramValue).

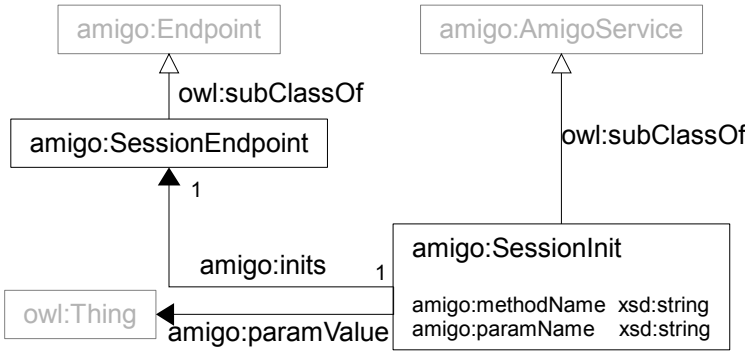

FIGURE B.2 – Modèle de description de services Amigo avec gestion de sessions

B.1.2 Découverte de descriptions

Le médiateur de découverte a pour rôle d'effectuer la découverte de descriptions de fonctionnalités fournissant des services Amigo. L'infrastructure de services Amigo s'appuie sur un protocole de découverte décentralisé et asynchrone (Georgantas & Thompson, 2007). Le médiateur de découverte utilise ce protocole de découverte pour découvrir dynamiquement les fonctionnalités de l'environnement. Il détermine ensuite une description sémantique pour ces fonctionnalités à partir des informations découvertes.

Dans le cadre du protocole de découverte Amigo, chaque service présente un ensemble de propriétés décrites par des couples attribut-valeur. Pour l'utilisation dans le cadre de FCAP, un service définit une propriété « description », dont la valeur est le couple (URI du service, URL du document décrivant la fonctionnalité). Le document contenant la description est accessible sur le réseau à l'URL indiquée. Il est éventuellement fournit par le dispositif proposant la fonctionnalité lui-même (mais pas nécessairement). Grâce à cette information, le médiateur obtient la description de la fonctionnalité qui fournit ce service. Le médiateur complète la description en y inscrivant la référence Amigo du service (cette référence est générée dynamiquement est n'est donc pas inscrite dans la description).

Lorsqu'une nouvelle description de fonctionnalité est découverte, le médiateur de découverte la compare à une description recherchée. Cette comparaison repose sur le mécanisme d'interprétation de type de fonctionnalités décrits dans la section 6.1.1. Lorsqu'une fonctionnalité correspond parfaitement à une fonctionnalité recherchée, le médiateur transmet la description au découvreur principal avec une priorité élevée. Lorsqu'une fonctionnalité correspond partiellement à une fonctionnalité recherchée, le médiateur transmet la description au découvreur principal avec une priorité faible.

B.1.3 Contrôle de fonctionnalité

Dans le cadre de l'infrastructure de services Amigo, on définit un contrôleur de fonctionnalité générique, qui permet de contrôler n'importe quelle fonctionnalité fournissant des services Amigo. Ce contrôleur de fonctionnalité utilise le modèle de description défini dans la section précédente pour effectuer les opérations élémentaires de composition. Pour générer un contrôleur de fonctionnalité pour une fonctionnalité particulière, il suffit donc de paramétrer le contrôleur générique avec les informations contenues dans la description de cette fonctionnalité.

Obtention d'une référence pour un service fourni L'algorithme lookupAmigo() permet d'obtenir la référence d'un service fourni. Dans cet algorithme, une référence du service est simplement obtenue en consultant la référence Amigo du point d'accès grâce auquel le service peut être invoqué :

Procédure Implementation de la méthode lookup($\underline{p}$: Resource) $\rightarrow ref$: Reference pour un contrôleur Amigo

Entrées : une ressource $\underline{s}$ décrivant un service fournit
Sorties : la référence Amigo du service ref

```
1 début
2     p ← 3db.select(s, amigo:invokeAt, ?p);
3     ref ← 3db.select(p, amigo:reference, ?r);
4     retourner ref;
5 fin
```

Attachement à un service attendu La procédure bindAmigo() permet d'attacher à un service attendu la référence d'un service fourni. Cette procédure utilise la description du point de connexion correspondant à un service attendu. Il extrait tout d'abord de la description le nom de l'opération de connexion. Il invoque ensuite cette opération de connexion en fournissant la référence du service fourni comme valeur de paramètre.

Procédure Implementation de la méthode bind($\underline{e}$: Resource, ref: Reference) pour un contrôleur Amigo

Entrées : une ressource $\underline{e}$ décrivant un service attendu, une référence ref de service fournit

```
1 début
2     i ← 3db.select(e, amigo:invokedBy, ?i);
3     m ← 3db.select(i, amigo:bindMethod, ?m);
4     p ← 3db.select(i, amigo:invokedAt, ?p);
5     r ← 3db.select(p, amigo:reference, ?r);
6     invokeAmigoService(r, m, {ref});
7 fin
```

Initialisation de l'utilisation Dans certains cas, la fonctionnalité fournit des services qui permettent d'initialiser son utilisation. On considère ici le cas de la mise en place d'une session d'utilisation. Le contrôleur comprend une méthode d'initialisation décrite à l'aide du modèle défini dans la section précédente.

La procédure initAmigo() du contrôleur consiste simplement à invoquer les différents services d'initialisation mentionnés dans la description. Pour chacun de ces services, le contrôleur obtient le nom de la méthode d'initialisation, le nom du paramètre d'initialisation et la valeur de ce paramètre. Il invoque ensuite cette méthode du service, et obtient comme résultat une référence Amigo pour le point d'accès concerné. Il complète alors la description en y inscrivant cette référence.

Procédure Implementation de la méthode `init` pour un contrôleur Amigo

```
1  début
2      S ← 3db.selectAll(?s, rdf:type, amigo:SessionInit) ;
3      pour tous les s ∈ S faire
4          m ← 3db.select(s, amigo:methodName, ?m) ;
5          v ← 3db.select(s, amigo:paramValue, ?v) ;
6          e ← 3db.select(s, amigo:invokeAt, ?e) ;
7          r ← 3db.select(e, amigo:reference, ?r) ;
8          n ← invokeAmigoService(r, m, {v});
9          i ← 3db.select(s, amigo:inits, ?i) ;
10         3db.write(i, amigo:reference, n) ;
11     fin
12 fin
```

B.1.4 Contrôle de connecteur

Dans le cadre de l'infrastructure de services Amigo, on définit deux types contrôleur de connecteur. Le premier permet de mettre en place un connecteur dans le cas où le service Amigo fourni correspond exactement au service Amigo attendu (connecteur direct). Le second permet de mettre en place un connecteur dans le cas où le service Amigo fourni est légèrement différent du servie attendu (connecteur adapté).

Connecteur direct Un connecteur direct est un connecteur établi entre des services Amigo directement compatibles entre eux : aucune adaptation n'est nécessaire. Dans ce cas, la mise en place d'une connexion entre une fonctionnalité qui fournit le service et celle qui attend le même service est très simple : il suffit de transmettre la référence du service fourni. Lors de la génération d'un contrôleur de connexion, les contrôleurs des fonctionnalités à connecter sont fournis, ainsi que les descriptions des services à connecter. À l'aide de ces informations, la procédure connectAmigo() décrit comment s'effectue cette connexion.

Procédure Implementation de la méthode `connect` pour un contrôleur Amigo

Données : La description c du contrôleur, les contrôleurs ctl_a et ctl_b des fonctionnalités à connecter

```
1  début
2      p ← 3db.select(c, conf:provided, ?p);
3      e ← 3db.select(c, conf:expected, ?e);
4      ctl_p ← controlerProviding(p, { ctl_a, ctl_b});
5      ctl_e ← controlerExpecting(e, { ctl_a, ctl_b});
6      ref ← ctl_p.lookup(p);
7      ctl_e.bind(e,ref);
8  fin
```

Connecteur adapté Un connecteur adapté est un connecteur établi entre des services Amigo légèrement différents : une adaptation est nécessaire pour permettre

l'utilisation d'un service fourni lorsque le service attendu est légèrement différent. La figure B.3 présente un modèle simple pour la description de connecteurs qui réalisent une adaptation de service. Ce modèle étend le modèle de description de connecteur de la figure 5.8 en indiquant une correspondance entre des opérations fournies et attendues par les services.

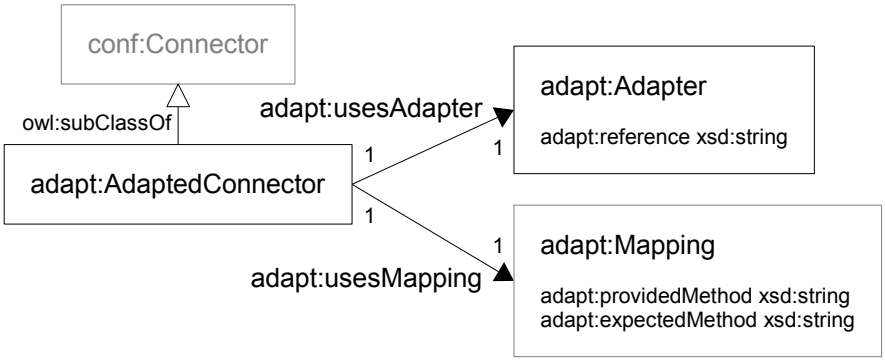

FIGURE B.3 – Modèle de description de connecteur avec adaptation de services

am-zb:AadaptedConnector représente le concept de connecteur adaptée entre des services légèrement différents.

am-zb:Adapter représente le concept d'adaptateur capable d'effectuer la correspondance entre des services légèrement différents.

am-zb:Mapping représente le concept de correspondance entre des services légèrement différents. Ici, on considère une correspondance très simple où seul le nom d'une méthode diffère.

Procédure Implementation de la méthode `connect` pour un contrôleur de connecteur Amigo adapté

Données : La description $\underline{c}$ du contrôleur, les contrôleurs ctl_a et ctl_b des fonctionnalités à connecter

début
  $\underline{p} \leftarrow$ `3db.select`($\underline{c}$, composition:provided, ?p);
  $\underline{e} \leftarrow$ `3db.select`($\underline{c}$, composition:expected, ?e);
  $ctl_p \leftarrow$ controlerProviding($\underline{p}$, { ctl_a, ctl_b});
  $ctl_e \leftarrow$ controlerExpecting($\underline{e}$, { ctl_a, ctl_b});
  $\underline{m} \leftarrow$ `3db.select`($\underline{c}$, adapt:usesMapping, ?p);
  $\underline{a} \leftarrow$ `3db.select`($\underline{c}$, adapt:usesAdapter, ?p);
  $ref_a \leftarrow$ `3db.select`($\underline{a}$, amigo:reference, ?r);
  $ref_p \leftarrow ctl_p$.lookup($\underline{p}$);
  $ref_e \leftarrow$ invokeAmigoService(ref_a, "adapt", $[m, ref_p]$) ;
  ctl_e.bind($\underline{e}$,ref);
fin

B.2 Intégration d'une infrastructure de services basée sur Zigbee

Zigbee est un protocole de communication sans-fil caractérisé par une faible consommation et un faible débit. Zigbee permet ainsi à des dispositifs disposant de faibles capacités de communiquer avec les autres dispositifs de l'environnement. Ils peuvent ainsi proposer des services au travers desquels leurs fonctionnalités peuvent être utilisés. Un exemple est la lampe ambiante décrite dans la section 8.3.2. Dans cette section, on s'intéresse ainsi à l'intégration d'une infrastructure de services basée sur Zigbee, que nous appellerons ZigbeeS.

B.2.1 Description sémantique

Il convient tout d'abord d'introduire un modèle de description simple pour les fonctionnalités de dispositifs utilisant l'infrastructure de services ZigbeeS. La figure B.4 illustre ce modèle.

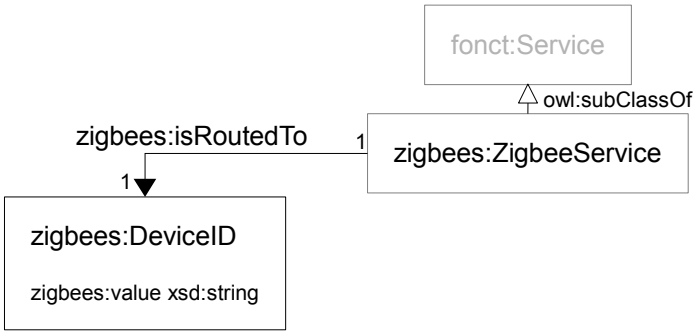

FIGURE B.4 – Modèle de description de fonctionnalités utilisant Zigbee

zigbees:DeviceID représente le concept d'identifiant du dispositif Zigbee considéré. On considère qu'un dispositif possède un identifiant unique grâce auquel il est possible de lui adresser des messages.

zigbees:isRoutedTo indique l'identifiant du dispositif vers lequel diriger une invocation de service. Dans ce modèle simple, on considère que les dispositifs peuvent uniquement fournir des services, et qu'un dispositif ne fournit qu'un seul service.

B.2.2 Contrôle de connecteur

Dans le cadre de nos expérimentations, on s'est intéressé à un connecteur qui effectue une passerelle entre les infrastructures de services Amigo et ZigbeeS. Pour intégrer l'utilisation d'un tel connecteur, on définit donc un modèle de description du connecteur.

Le modèle de description pour les connecteurs qui réalisent une passerelle entre les infrastructures de services Amigo et ZigbeeS est présenté dans la figure B.5. Ce modèle étend le modèle de description de connecteur de la figure 5.8, en indiquant l'adresse d'une passerelle chargée d'effectuer le lien entre les deux infrastructures de services.

am-zb:AmigoZibgeeConnector représente le concept de connecteur entre les infrastructures Amigo et Zigbee.

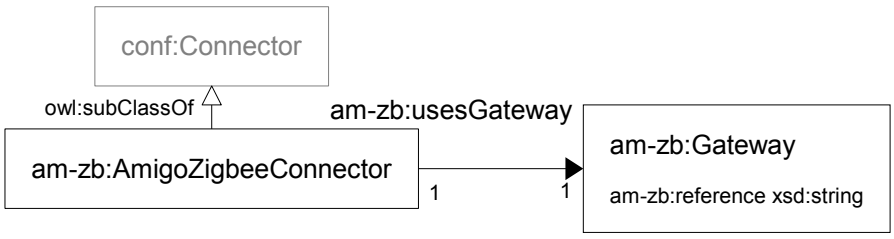

FIGURE B.5 – Modèle de description de connecteur pour les services Zigbee

am-zb:Gateway représente le concept de passerelle entre les infrastructures Amigo et Zigbee. Une passerelle possède une référence Amigo grâce à laquelle elle peut être utilisée.

am-zb:usesGateway indique que le connecteur utilise la passerelle spécifiée.

References

Abowd, Gregory D., Mynatt, E. D., & Rodden, T. 2002. The human experience. *Pervasive Computing, IEEE*, **1**(1), 48–57. 13, 25

Abowd, Gregory D., Hayes, G. R., Iachello, G., Kientz, J. A., Patel, S. N., Stevens, M. M., & Truong, K. N. 2005. Prototypes and paratypes : designing mobile and ubiquitous computing applications. *Pervasive Computing, IEEE*, **4**(4), 67–73. 29

Allen, Robert J. 1997. *A Formal Approach to Software Architecture.* Ph.D. thesis, Carnegie Mellon University. 58

Ambite, J. L., Knoblock, C. A., Muslea, M., & Minton, S. 2005. Conditional constraint networks for interleaved planning and information gathering. *Intelligent Systems*, **20**(March), 25–33. 68

Ankolekar, Anupriya, Burstein, Mark H., Hobbs, Jerry R., Lassila, Ora, Martin, David L., McDermott, Drew, McIlraith, Sheila A., Narayanan, Srini, Paolucci, Massimo, Payne, Terry R., & Sycara, Katia. 2002 (June9–2). DAML-S : Web Service Description for the Semantic Web. *Pages 348–363 of :* Horrocks, I., & Hendler, J. (eds), *ISWC 2002, First International Semantic Web Conference*, vol. 2342 / 2002. 51

Ballesteros, F. J., Soriano, E., Leal, K., & Guardiola, G. 2006 (March). Plan B : an operating system for ubiquitous computing environments. *Page 10 pp. of : Pervasive Computing and Communications (PerCom 2006). Fourth Annual IEEE International Conference on.* 35

Bardram, Jakob. 2005. The trouble with login : on usability and computer security in ubiquitous computing. *Personal and Ubiquitous Computing*, **9**(6), 357–367. 17

Barkhuus, Louise, & Dey, Anind K. 2003. Is Context-Aware Computing Taking Control Away from the User ? Three Levels of Interactivity Examined. *Pages 159–166 of : Ubiquitous Computing (UbiComp 2003), 5th International Conference on.* 24

Beatty, J. et al. 2005 (Apr.). *Web Services Dynamic Discovery (WS-Discovery).* http ://schemas.xmlsoap.org/ws/2005/04/discovery. 49

Becker, Christian, Handte, Marcus, Schiele, Gregor, & Rothermel, Kurt. 2004 (March23–26). PCOM - A Component System for Pervasive Computing. *Pages 67–76 of : Pervasive Computing and Communications (PerCom 2004). Second IEEE Annual Conference on.* 39

Ben Hassine, Ahlem, Matsubara, Shigeo, & Ishida, Toru. 2006. A Constraint-Based Approach to Horizontal Web Service Composition. *Pages 130–143 of : ISWC2006 5th International Semantic Web Conference.* LNCS. 56

Ben Mokhtar, S., Liu, J., Georgantas, N., & Issarny, V. 2005. Qos-aware dynamic service composition in ambient intelligence environments. *In : 20th IEEE/ACM International Conference on Automated Software Engineering(ASE '05).* 53

Ben Mokhtar, Sonia. 2007. *Intergiciel Sémantique pour les Services de l'Informatique diffuse.* Ph.D. thesis, Université Paris 6. 188

Berhe Hagos, Girma. 2006. *Accès et adaptation de contenu multimedia pour les systèmes pervasifs.* Ph.D. thesis, Institut National des Sciences Appliquées de Lyon. 53

Berners-Lee, Tim, Hendler, James, & Lassila, Ora. 2001. The Semantic Web. *Scientific American*, **284**(5), 34–43. 50, 69

Bouchard, Bruno, Giroux, Sylvain, & Bouzouane, Abdenour. 2006. A Smart Home Agent for Plan Recognition of Cognitively-impaired Patients. *Journal of Computers (JCP)*, **1**(5), 53–62. 30

Bruneton, Eric, Coupaye, Thierry, & Stefani, Jean-Bernard. 2002 (June). Recursive and dynamic software composition with sharing. *In : 7th ECOOP International Workshop on Component-Oriented Programming (WCOP'02).* 58

Brush, A., & Inkpen, Kori. 2007. Yours, Mine and Ours ? Sharing and Use of Technology in Domestic Environments. *Pages 109–126 of : UbiComp 2007 : Ubiquitous Computing.* 23

Busetta, Paolo, Donà, Antonia, & Nori, Michele. 2002. Channeled multicast for group communications. *Pages 1280–1287 of : Proceedings of the first international joint conference on Autonomous agents and multiagent systems : part 3.* Bologna, Italy : ACM. 65

Busetta, Paolo, Merzi, Mattia, Zancanaro, Massimo, & Rossi, Silvia. 2003. Group Communication for Real-time Role Coordination and Ambient Intelligence. *In : Ubicomp 2003 Workshop on AI in Mobile Systems (AIMS 2003).* 40

Calvary, Gaëlle, Coutaz, Joëlle, & Thevenin, David. 2001. *Engineering for Human-Computer Interaction.* Springer. Chap. A Unifying Reference Framework for the Development of Plastic User Interfaces, pages 173–192. 60

Calvary, Gaëlle, Coutaz, Joëlle, Dâassi, Olfa, Balme, Lionel, & Demeure, Alexandre. 2005. *Engineering Human Computer Interaction and Interactive Systems.* Springer. Chap. Towards a New Generation of Widgets for Supporting Software Plasticity : The "Comet", pages 306–324. 60

Chakraborty, Dipanjan, Joshi, Anupam, Yesha, Yelena, & Finin, Tim. 2006. Toward Distributed Service Discovery in Pervasive Computing Environments. *IEEE Transactions on Mobile Computing*, **5**(2), 97–112. 53

Charton, Romaric, Boyer, Anne, & Boyer, Anne. 2003. Learning of Mediation Strategies for Heterogeneous Agents Cooperation. *Pages 330–337 of : 15th IEEE International Conference on Tools with Artificial Intelligence - ICTAI'2003.* 69

Cheung-Foo-Wo, Daniel, Tigli, Jean-Yves, Lavirotte, Stéphane, & Riveill, Michel. 2006. WComp : a Multi-Design Approach for Prototyping Applications using Heterogeneous Resources. *In : Proceedings of the 17th IEEE International Workshop on Rapid System Prototyping.* 31

Christensen, E. et al. 2001 (Mar.). *Web Services Description Language (WSDL) 1.1.* http ://www.w3.org/TR/2001/NOTE-wsdl-20010315. 46

Claro, Daniela, Albers, Patrick, & Hao, Jin-Kao. 2006. *Semantic Web Services, Processes and Applications.* Springer US. Chap. Web Services Composition, pages 195–225. 53

Clement, Luc, Hately, Andrew, von Riegen, Claus, & (eds.), Tony Rogers. 2004 (October 19). *The universal description, discovery and integration (UDDI) Version 3.0.2 Specification.* OASIS Specification. OASIS. http ://uddi.org/pubs/uddi-v3.0.2-20041019.htm. 46

Coutaz, Joëlle, Crowley, James L., Dobson, Simon, & Garlan, David. 2005. Context is key. *Communications of the ACM*, **48**(3), 49–53. The Disappearing Computer (special issue). 15

Davies, N., & Gellersen, H.-W. 2002. Beyond prototypes : challenges in deploying ubiquitous systems. *Pervasive Computing, IEEE*, **1**(1), 26–35. 20, 25

Derrett, Nigel. 2006. Is automation automatically a good thing ? *Personal and Ubiquitous Computing*, **10**(2), 56–59. 25

Desnos, Nicolas, Huchard, Marianne, Urtado, Christelle, Vauttier, Sylvain, & Tremblay, Guy. 2007. Automated and Unanticipated Flexible Component Substitution. *Pages 33–48 of : Proceedings of Component-Based Software Engineering (CBSE 2007)*. 59

Dey, Anind K. 2000. *Providing architectural support for Building Context-Aware Applications*. Ph.D. thesis, Georgia institue of Technology. 82

Dey, Anind K., Salber, Daniel, & Abowd, Gregory D. 2001. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction (HCI) Journal, Special Issue on Context-Aware Computing*, **16**(2-4), 97–166. 31

Dey, Anind K., Sohn, Timothy, Streng, Sara, & Kodama, Justin. 2006 (May). iCAP : Interactive Prototyping of Context-Aware Applications. *Pages 254–271 of : Pervasive 2006 : 4th International Conference on Pervasive Computing, Dublin, Ireland*. 34

Dobson, Simon. 2003. Applications considered harmful for ambient systems. *Pages 163–168 of : Invited Workshop on Adaptive Systems for Ubiquitous Computing, Proceedings of the 1st International Symposium on Information and Communication Technologies (ISICT '03), Dublin, Ireland*. 18

Dow, S., MacIntyre, B., Lee, J., Oezbek, C., Bolter, J.D., & Gandy, M. 2005. Wizard of Oz support throughout an iterative design process. *Pervasive Computing, IEEE*, **4**(4), 18–26. 29

Drabble, Brian, & Tate, Austin. 1996. O-Plan : A Situated Planning Agent. *Pages 247–260 of :* Ghallab, Malik, & Milani, A. (eds), *New Directions in AI Planning*. IOS Press (Amsterdam). 68

Dupuis, Rémi, Pierson, Jérôme, & Vallée, Mathieu. 2007. Flexible & Dynamic Device Usage based on Software Agents and Semantic Web Technologies in an Ambient Home Environment. *In : Adjunct Proceedings of Ubicomp 2007*. 13

Elting, Christian, & Hellenschmidt, Michael. 2004 (September 7). Strategies for Self-Organization and Multimodal Output Coordination in Distributed Device Environments. *In : UbiComp 2004 Workshop on AI in Mobile Systems (AIMS 2004)*. 40

Escoffier, Clément, & Hall, Richard. 2007. Dynamically Adaptable Applications with iPOJO Service Components. *In : sc*. 61

Euzenat, Jérôme, Pierson, Jérôme, & Ramparany, Fano. 2008. Dynamic Context Management for Pervasive Applications. *The Knowledge Engineering Review*, **23**(Special Issue 01), 21–49. 100

Fensel, Dieter, & Musen, Mark A. 2001. The semantic web : a brain for humankind. *Intelligent Systems*, **16**(Mar.), 24–25. 2

Fensel, Dieter, Lausen, Holger, Bruijn, Jos, Stollberg, Michael, Roman, Dumitru, Polleres, Axel, & Domingue, John. 2007. *Enabling Semantic Web Services*. Springer Berlin Heidelberg. 45, 52

FIPA. 2002a. *FIPA ACL Message Structure Specification*. FIPA Standard specification 00061. Foundation for Intelligent Physical Agents. http ://www.fipa.org/specs/fipa00061/SC00061G.html. 64, 173

FIPA. 2002b. *FIPA Communicative Act Library Specification*. FIPA Standard specification 00037. Foundation for Intelligent Physical Agents. http ://www.fipa.org/specs/fipa00037/SC00037J.html. 173

FIPA. 2002c. *FIPA SL Content Language Specification.* FIPA Standard specification 00008. Foundation for Intelligent Physical Agents. http ://www.fipa.org/specs/fipa00008/SC00008I.html. 64, 138, 173

Fitton, D., Cheverst, K., Kray, C., Dix, A., Rouncefield, M., & Saslis-Lagoudakis, G. 2005. Rapid prototyping and user-centered design of interactive display-based systems. *Pervasive Computing, IEEE*, **4**(4), 58–66. 29

Garlan, D., Siewiorek, D. P., Smailagic, A., & Steenkiste, P. 2002. Project Aura : toward distraction-free pervasive computing. *Pervasive Computing, IEEE*, **1**(2), 22–31. 24, 37

Georgantas, Nikolaos, & Thompson, Graham. 2007 (May 21). *Amigo Overall Middleware : First Prototype Implementation & Documentation.* Deliverable D3.4. IST Amigo Project. http ://www.hitech-projects.com/euprojects/amigo/deliverables/amigo_d3.4_correctedfinal.pdf. 213, 214, 215

Grimm, R. 2004. System Support for Pervasive Applications. *ACM Transactions on Computer Systems*, **22**(4), 421–486. 20

Grondin, Guillaume, Bouraqadi, Noury, & Vercouter, Laurent. 2006. MaDcAr : An Abstract Model for Dynamic and Automatic (Re-)Assembling of Component-Based Applications. *Pages 360–367 of : Component-Based Software Engineering.* 60

Gruber, Thomas R. 1993 (March). Toward Principles for the Design of Ontologies used for Knowledge Sharing. *In : Workshop on Formal Ontology, Padua, Italy.* 51

Gruber, Tom. 2006. Where the Social Web Meets the Semantic Web. *Page 994 of : ISWC2006 5th International Semantic Web Conference.* LNCS. Invited talk. 70

Heider, Thomas, & Kirste, Thomas. 2002 (June12–14). Architecture Consideration for interoperable multi-modal assistant systems. *Pages 253–267 of :* Forbrig, P., Limbourg, Q., Urban, B., & Vanderdonckt, J. (eds), *9th International Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS 2002).* Lecture Notes in Computer Science, vol. 2545 / 2002. 38

Hellenschmidt, Michael. 2005 (October12–14). Distributed Implementation of a Self-Organizing Appliance Middleware. *Pages 201–206 of : International Joint Conference on Smart Object and Ambient Intelligence (sOc-EUSAI 2005).* 39

Henocque, Laurent, & Kleiner, Mathias. 2007. *Semantic Web Services : Concepts, Technologies, and Applications.* Springer. Chap. Composition : Combining Web Service Functionality in Composite Orchestrations, pages 245–286. 55

Howard, Steve, Kjeldskov, Jesper, & Skov, Mikael. 2007. Pervasive computing in the domestic space. *Personal and Ubiquitous Computing*, **11**(5), 329–333. 22, 23, 30

Hsieh, Gary, Tang, Karen, Low, Wai, & Hong, Jason. 2007. Field Deployment of IMBuddy : A Study of Privacy Control and Feedback Mechanisms for Contextual IM. *Pages 91–108 of : UbiComp 2007 : Ubiquitous Computing.* 12

Hübner, Jomi Fred, Sichman, Jaime Simão, & Boissier, Olivier. 2002. Spécification structurelle, fonctionnelle et déontique d'organisations dans les SMA. *In : Journée Francophones pour l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents 2002 (JFIADSMA'02).* 63

Intille, S. S. 2002. Designing a home of the future. *Pervasive Computing, IEEE*, **1**(2), 76–82. 25

Jennings, Nicholas R., Sycara, Katia, & Wooldridge, Michael. 1998. A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems*, **1**(March), 7–38. 61, 62

Kaisler, Stephen H. 2005. *Software Paradigms.* Wiley. 58

Kameas, A., Bellis, S., Mavrommati, I., Delaney, K., Colley, M., & Pounds-Cornish, A. 2003 (March). An architecture that treats everyday objects as communicating tangible components. *Pages 115–122 of : Pervasive Computing and Communications (PerCom 2003). First IEEE International Conference on.* 34

Kidd, Cory D., Orr, Robert J., Abowd, Gregory D., Atkeson, Christopher G., Essa, Irfan A., MacIntyre, Blair, Mynatt, Elizabeth, Starner, Thad E., & Newstetter, Wendy. 1999 (October). The Aware Home : A Living Laboratory for Ubiquitous Computing Research. *In : Second International Workshop on Cooperative Buildings - CoBuild'99.* 30

Kindberg, T., & Fox, A. 2002. System software for ubiquitous computing. *Pervasive Computing, IEEE*, **1**(1), 70–81. 21, 22, 25

Kindberg, Tim, Barton, John, Morgan, Jeff, Becker, Gene, Caswell, Debbie, Debaty, Philippe, Gopal, Gita, Frid, Marcos, Krishnan, Venky, Morris, Howard, Schettino, John, & Serra, Bill. 2000. People, Places, Things : Web Presence for the Real World. *In : Mobile Computing Systems and Applications, Third International IEEE Workshop on.* 33

Krötzsch, Markus, Vrandecic, Denny, & Völkel, Max. 2006. Semantic MediaWiki. *Pages 935–942 of : ISWC2006 5th International Semantic Web Conference.* LNCS. 70

L. Cabrera, et al. 2004 (Aug.). *Web Services Eventing (WS-Eventing).* http ://schemas.xmlsoap.org/ws/2004/08/eventing/. 49

Lamma, E. 1999. Constraint Propagation and Value Acquisition : Why We Should Do It Interactively. *Pages 468–477 of : IJCAI 99, 16th Int'l Joint Conf. Artificial Intelligence.* 68

Lesser, Victor R., Decker, Keith, Wagner, T., Carver, N., Garvey, A., Horling, Bryan, Neiman, D., Podorozhny, R., Prasad, Nagendra M., Raja, A., Vincent, R., Xuan, P., & Zhang, X. Q. 2002. Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework. *Pages 1–2 of : First International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2002).* (Plenary Lecture/Extended Abstract). 63

Luck, M., McBurney, P., Shehory, O., & Willmott, S. 2005. *Agent Technology : Computing as Interaction (A Roadmap for Agent Based Computing).* AgentLink. 61

Maamar, Zakaria, Sheng, Quan Z., & Benatallah, Boualem. 2003 (July 14). Interleaving Web Services Composition and Execution Using Software Agents and Delegation. *In : AAMAS 2003 Workshop on Web Services and Agent Based engineering (WSABE 2003).* 56

Maamar, Zakaria, Sheng, Quan, & Benatallah, Boualem. 2004. *Extending Web Services Technologies.* Springer US. Chap. Agent-Based Support for Service Composition, pages 139–160. 56

Maamar, Zakaria, Kouadri Mostéfaoui, Soraya, & Yahyaoui, Hamdi. 2005. Toward an Agent-Based and Context-Oriented Approach for Web Services Composition. *IEEE Transactions on Knowledge and Data Engineering*, **17**(5), 686–697. 53

Martin, David, Burstein, Mark, McDermott, Drew, McIlraith, Sheila, Paolucci, Massimo, Sycara, Katia, McGuinness, Deborah, Sirin, Evren, & Srinivasan, Naveen. 2007. Bringing Semantics to Web Services with OWL-S. *World Wide Web*, **10**, 243–277. 51, 52

Martin, David L., Cheyer, Adam J., & Moran, Douglas B. 1999. The Open Agent Architecture : A Framework for Building Distributed Software Systems. *Applied Artificial Intelligence*, **13**(1-2), 91–128. 64, 69

Mathieu, P., Routier, J. C., & Secq, Y. 2002. Dynamic organization of multi-agent systems. *Pages 451–452 of : Proceedings of the first international joint conference on Autonomous agents and multiagent systems : part 1.* Bologna, Italy : ACM. 63

Matthews, Tara, Dey, Anind K., Mankoff, Jen, Carter, Scott, & Rattenbury, Tye. 2004. A Toolkit for Managing User Attention in Peripheral Displays. *In : Proceedings of UIST 2004.* 24

McGovern, James, Tyagi, Sameer, Stevens, Michael, & Mathew, Sunil. 2003. Chapter 2 : Service-Oriented Architecture. *In : Java Web Services Architecture.* Morgan Kaufmann Publishers. 45

McIlraith, Sheila A., & Son, Tran Cao. 2002 (April). Adapting Golog for Composition of Semantic Web Services. *In : Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002).* 55

Modahl, Martin, Agarwalla, Bikash, Saponas, T., Abowd, Gregory D., & Ramachandran, Umakishore. 2006. UbiqStack : a taxonomy for a ubiquitous computing software stack. *Personal and Ubiquitous Computing*, **10**(1), 21–27. 16, 18

Nareyek, A., Freuder, E. C., Fourer, R., Giunchiglia, E., Goldman, R. P., Kautz, H., Rintanen, J., & Tate, A. 2005. Constraints and AI planning. *Intelligent Systems*, **20**(March), 62–72. 68

Norman, Donald A. 1998. *The Invisible Computer.* MIT Press. 27, 32

Noy, Natalya, Chugh, Abhita, Liu, William, & Musen, Mark. 2006. A Framework for Ontology Evolution in Collaborative Environments. *Pages 544–558 of : ISWC2006 5th International Semantic Web Conference.* LNCS. 70

OMG. 1999 (July). *Unified Modeling Language version 1.3.* Tech. rept. Object Management Group. 139

OMG. 2006 (January 4). *CORBA Component Model, version 4.0.* Tech. rept. Object Management Group. Available at : http ://www.omg.org/cgi-bin/doc ?formal/06-04-01. 58

Oreizy, P., Gorlick, M.M., Taylor, R.N., Heimhigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D.S., & Wolf, A.L. 1999. An architecture-based approach to self-adaptive software. *Intelligent Systems, IEEE*, **14**(3), 54–62. 58, 59, 83

Paluska, Justin Mazzola, Pham, Hubert, Saif, Umar, Terman, Chris, & Ward, Steve. 2006. Reducing Configuration Overhead with Goal-oriented Programming. *In : Percom 2006 Workshop on ?* 38

Pauchet, Alexandre, Chaignaud, Nathalie, & Seghrouchni, Amal El Fallah. 2007. A computational model of human interaction and planning for heterogeneous multi-agent systems. *Pages 1–3 of : Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems.* Honolulu, Hawaii : ACM. 69

Pigot, H., Mayers, A., & Giroux, S. 2003 (Apr.). The intelligent habitat and everyday life activity support. *Pages 507–516 of : Proc. of the 5th International conference on Simulations in Biomedicine.* 30

Ponnekanti, Shankar R., & Fox, Armando. 2002 (May). SWORD : A Developper Toolkit for Building Composite Web Service. *In : 11th International World Wide Web Conference (WWW11).* 55

Ponnekanti, Shankar R., Lee, Brian, Fox, Armando, Hanrahan, Pat, & Winograd, Terry. 2001 (September30–October2). ICrafter : A Service Framework for Ubiquitous Computing Environments. *Pages 56–75 of : Ubiquitous Computing (UbiComp2001), proceedings of the third International Conference on.* 33

Preist, Chris. 2004 (November7–11). A Conceptual Architecture for Semantic Web Services. *Pages 395–409 of : ISWC 2004, Third International Semantic Web Conference.* 45

Ranganathan, A., Chetan, S., Al-Muhtadi, J., Campbell, R.H., & Mickunas, M.D. 2005. Olympus : A High-Level Programming Model for Pervasive Computing Environments. *Pages 7–16 of : Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on.* 35

Ranganathan, Anand, & Campbell, RoyH. 2003. An infrastructure for context-awareness based on first order logic. *Personal and Ubiquitous Computing*, **7**(6), 353–364. 12

Röcker, Carsten, Janse, Maddy D., Portolan, Nathalie, Streitz, Norbert, & Streitz, Norbert. 2005. User requirements for intelligent home environments : a scenario-driven approach and empirical cross-cultural study. *Pages 111–116 of : Proceedings of the 2005 joint conference on Smart objects and ambient intelligence : innovative context-aware services : usages and technologies.* Grenoble, France : ACM. 17

Reilly, D., Dearman, D., Welsman-Dinelle, M., & Inkpen, K. 2005. Evaluating early prototypes in context : trade-offs, challenges, and successes. *Pervasive Computing, IEEE*, **4**(4), 42–50. 29

Rey, Gaëtan. 2005. *Contexte en Interaction Homme-Machine : le contexteur.* Ph.D. thesis, Communication Langagière et Interaction Personne-Système - Fédération IMAG - Université Joseph Fourier - Grenoble I. 82

Ricquebourg, Vincent, Durand, David, Menga, David, Delahoche, Laurent, Marhic, Bruno, Logé, Christophe, & Jolly-Desodt, Anne-Marie. 2007. La fusion multi-capteurs dans l'habitat communicant : une approche non-probabiliste. *Pages 9–16 of : Proceedings of the 4th French-speaking conference on Mobility and ubiquity computing.* Saint Malo, France : ACM. 100

Rogers, Yvonne, Connelly, Kay, Tedesco, Lenore, Hazlewood, William, Kurtz, Andrew, Hall, Robert, Hursey, Josh, & Toscos, Tammy. 2007. Why It's Worth the Hassle : The Value of In-Situ Studies When Designing Ubicomp. *Pages 336–353 of : UbiComp 2007 : Ubiquitous Computing.* 29

Roman, M., Ziebart, Brian, & Campbell, Roy H. 2003 (March). Dynamic application composition : customizing the behavior of an active space. *Pages 169–176 of : Pervasive Computing and Communications (PerCom 2003). First IEEE International Conference on.* 31

Roman, Manuel. 2003. *An Application Framework for Active Space Applications.* Ph.D. thesis, University of Illinois, Urbana-Champaign. 12, 31

Roman, Manuel, & Campbell, Roy H. 2003 (June16–20). Middleware-Based Application Framework for Active Space Applications. *In : ACM/IFIP/USENIX 4th International Middleware Conference (Middleware 2003).* 31

Rousset, Marie-Christine. 2004. Small Can Be Beautiful in the Semantic Web. *Pages 6–16 of : ISWC 2004, Third International Semantic Web Conference.* Invited talk. 69

Russell, Daniel M., Streitz, Norbert A., & Winograd, Terry. 2005. Building disappearing computers. *Communications of the ACM*, **48**(3), 42–48. The Disappearing Computer (special issue). 18, 19, 30

Saif, Umar, & Greaves, David J. 2001. Communication Primitives for Ubiquitous Systems or RPC Considered Harmful. *In : Proceedings of ICDCS International Workshop on Smart Appliances and Wearable Computing.* 17

Saif, Umar, Pham, Hubert, Paluska, Justin Mazzola, Waterman, Jason, Terman, Chris, & Ward, Steve. 2003 (October 12). A Case for Goal-oriented Programming Semantics. *In : Ubicomp 2003 Workshop on System Support for Ubiquitous Computing (UbiSys'03).* System Support and AI session. 38

Shaw, Mary, & Garlan, David. 1996. *Software architecture : perspectives on an emerging discipline.* Prentice-Hall, Inc. 58

Sirin, Evren. 2006. *Combining description logic reasoning with AI planning for composition of Web services.* Ph.D. thesis, University of Maryland, College Park. 56

Sirin, Evren, Parsia, Bijan, & Hendler, James. 2004 (March22–24). Composition-driven Filtering and Selection of Semantic Web Services. *In : Semantic Web Services - 2004 AAAI Spring Symposium.* 67

Sousa, João Pedro, & Garlan, David. 2002 (August25–1). Aura : an Architectural Framework for User Mobility in Ubiquitous Computing Environments. *Pages 29–43. of :* Bosch, Jan, Gentleman, Morven, Hofmeister, Christine, & Kuusela, Juha (eds), *Software Architecture : System Design, Development, and Maintenance, Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture, Montreal, Canada.* 37

Sqalli, M.H., & Freuder, E.C. 1996. Inference-Based Constraint Satisfaction Supports Explanation. *Pages 318–325 of : AAAI 96, 13th Nat'l Conf. Artificial Intelligence.* 68

Sycara, Katia, Widoff, Seth, Klusch, Matthias, & Lu, Jianguo. 2002. Larks : Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. *Autonomous Agents and Multi-Agent Systems*, **5**(2), 173–203. 65

Szyperski, Clemens A. 1998. *Component Software - Beyond Object-Oriented Programming.* Addison-Wesley. 57

Tambe, Milind. 1997. Towards Flexible Teamwork. *Journal of Artificial Intelligence Research*, **7**, 83–124. 63

Taylor, Alex, Harper, Richard, Swan, Laurel, Izadi, Shahram, Sellen, Abigail, & Perry, Mark. 2007. Homes that make us smart. *Personal and Ubiquitous Computing*, **11**(5), 383–393. 29

Terrenghi, Lucia, Hilliges, Otmar, & Butz, Andreas. 2007. Kitchen stories : sharing recipes with the Living Cookbook. *Personal and Ubiquitous Computing*, **11**(5), 409–414. 13

Tigli, Jean-Yves, Cheung-Foo-Wo, Daniel, Lavirotte, Stéphane, & Riveill, Michel. 2006. Adaptation au contexte par tissage d'aspects d'assemblage de composants déclenchés par des conditions contextuelles. *RTSI Série ISI - Adaptation et Gestion du Contexte*, **11**(5), 89–114. 31

Truong, Khai N., Huang, Elaine M., & Abowd, Gregory D. 2004. CAMP : A Magnetic Poetry Interface for End-User Programming of Capture Applications for the Home. *Pages 143–160 of : UbiComp 2004 : Ubiquitous Computing.* 13, 34

UPnP Forum. 2002. *UPnP Device Architecture.* 48

UPnP Forum. 2006. *UPnP Device Architecture.* Http ://www.upnp.org/specs/arch/UPnP-DeviceArchitecture-v1.0.pdf. Dated July 20, 2006. 47, 48, 52

Vercouter, Laurent. 2000. *Conception et mise en œuvre de systèmes multi-agent ouverts et distribués.* Ph.D. thesis, École des Mines de St-Etienne. 65

Verfaillie, G., & Schiex, T. 1994. Solution Reuse in Dynamic Constraint Satisfaction Problems. *Pages 307–312 of : AAAI 94, 12th Nat'l Conf. Artificial Intelligence.* 68

W3C. 2004a. *SAWSDL.* W3C Candidate Recommendation. W3C. Available at : http ://www.w3.org/2002/ws/sawsdl/. 51

W3C. 2004b. *SWRL : A Semantic Web Rule Language Combining OWL and RuleML.* W3C Submission. W3C. 114

W3C. 2004c (February 11). *Web Services Architecture.* W3C Working Group Note. W3C. http ://www.w3.org/TR/2004/NOTE-ws-arch-20040211/. 46

W3C. 2008 (January 15). *SPARQL Query Language for RDF.* W3C Recommendation. W§C. 101

Want, R., Pering, T., Borriello, Gaetano, & Farkas, K. I. 2002. Disappearing hardware. *Pervasive Computing, IEEE*, **1**(1), 36–47. 16

Weiser, Mark. 1991. The Computer for the 21st Century. *Scientific American*, **165**(3), 94–104. 2

Wickler, Gerhard. 1999. *Using Expressive and Flexible Representations to Reason about Capabilities for Intelligent Agent Cooperation.* Ph.D. thesis, University of Edinburgh. 65

Wu, Dan, Parsia, Bijan, Sirin, Evren, Hendler, James, & Nau, Dana S. 2003 (October20–23). Automating DAML-S web services composition using SHOP2. *Pages 195–210 of : ISWC 2003, Second International Semantic Web Conference.* 56

Yau, S. S., Karim, F., Wang, Yu, Wang, Bin, & Gupta, S. K. S. 2002. Reconfigurable context-sensitive middleware for pervasive computing. *Pervasive Computing, IEEE*, **1**(3), 33–40. 18

Yokoo, Makoto, & Hirayama, Katsutoshi. 2000. Algorithms for Distributed Constraint Satisfaction : A Review. *Autonomous Agents and Multi-Agent Systems*, **3**(2), 185–207. 63

École Nationale Supérieure des Mines
de Saint-Étienne

N° d'ordre : 514 I

Mathieu VALLÉE

A MULTI-AGENT MIDDLEWARE FOR FLEXIBLE APPLICATION COMPOSITION IN AMBIENT INTELLIGENCE

Speciality : Computer Science

Keywords : Ambient Intelligence, Multi-Agent System, Composition, Semantic Services, Context-Awareness

Abstract :

The concept of ambient intelligence results from a recent evolution of information systems. While information systems and computers were primarily used as tools, they are now transforming into a kind of environment, which literally surrounds humans in their daily lives. As computer devices become smaller, more diverse and more interconnected, they now become an integral part of our physical environment, thus opening new possibilities for interaction and experiences. More specifically, the domestic environment can progressively become a person-aware environment, capable of assisting and enhancing daily life activities.

However, designing applications for such a person-aware environment (called person-aware applications) raises numerous challenges. Especially, devices are *heterogeneous*, environments are unstable and *evolving*, and users' needs are variable and *imprecise*. In order to facilitate the design of person-aware applications, we focus on the notion of *flexible composite applications* (FCAP): applications working continuously in the background of our attention with the goal of aggregating functionalities in order to support our activities. We thus investigate a middleware for flexible composition of applications (called FCAP). FCAP provides a generic support for integrating heterogeneous functionalities, adapting applications to an unstable environment and adjusting the balance between automation and end-user control. Our approach builds on principles derived from the fields of service-oriented architecture, semantic Web and multi-agent systems, as each of these fields present relevant properties for our environments.

We have implemented and validated FCAP by implementing several person-aware applications and making them evolve. Our experimentations demonstrate that our approach for flexible composition of application is suitable for improving the design of person-aware applications.

École Nationale Supérieure des Mines
de Saint-Étienne

N° d'ordre : 514 I

Mathieu VALLÉE

UN INTERGICIEL MULTI-AGENT POUR LA COMPOSITION FLEXIBLE D'APPLICATIONS EN INTELLIGENCE AMBIANTE

Spécialité: Informatique

Mots clefs : Intelligence Ambiante, Système multi-agent, Composition, Services Sémantiques, Sensibilité au contexte.

Résumé :

La notion d'intelligence ambiante découle d'une évolution des systèmes informatiques, qui passent progressivement du statut d'outil au statut d'environnement, dans lequel les utilisateurs se trouvent immergés. Les dispositifs informatiques, de plus en plus petits, diversifiés et interconnectés, se fondent en partie dans l'environnement physique et ouvrent de nouvelles possibilités d'interaction et d'expériences. En particulier, l'environnement domestique se transforme en un environnement attentif, capable d'accompagner les activités de la vie quotidienne.

Cependant, la conception d'applications pour un environnement attentif (les applications attentives) soulève de nombreux défis, en particulier liés à l'*hétérogénéité* des objets communicants, à l'*instabilité* de l'environnement et à la *variabilité* des besoins. Afin de faciliter la conception d'applications attentives, les travaux présentés ici s'intéressent à la notion d'*application composite flexible* : des applications qui fonctionnent en continu, en arrière plan de l'attention, et agrègent des fonctionnalités fournies par les objets communicant, dans le but d'accompagner l'interaction des utilisateurs. Un intergiciel pour la composition flexible d'applications, nommé FCAP, est présenté. FCAP fournit un support générique pour intégrer les fonctionnalités, s'adapter à un environnement instable et ajuster l'équilibre entre l'automatisation et le contrôle par les utilisateurs. Cet intergiciel développe des principes issus de l'architecture orientée-service, du Web sémantique et des systèmes multi-agents, dont les caractéristiques sont particulièrement pertinentes pour les environnements considérés.

FCAP a été implémenté et validé par la réalisation de plusieurs applications attentives. Nos expérimentations indiquent ainsi l'intérêt de l'approche de composition flexible pour faciliter la conception d'applications attentives.