Université Abderrahmane Mira de Bejaïa
Faculté des Sciences exactes
Département d'Informatique

Mémoire de Master en Informatique

Option

Systèmes complexes et Technologies de l'Information et du Contrôle (SC TIC)

Thème

Composition dynamique de services sensible au contexte en environnements ubiquitaires

---

Présenté par :

Mlle Nesrine MAZOUZ.

Encadreurs :

Mlle Amel KOUICEM.

Dr Abdelkamel TARI.

Membre du jury :

Mlle Souhila GHANEM.

Mlle Salima SABRI.

Président du jury :

Mr Achour ACHROUFENE.

Juin 2012.

Remerciements

Mes vifs remerciements sont adressés :

En premier lieu à mes parents pour m'avoir soutenus et pour tous les sacrifices consentis, ainsi qu'à tous mes proches et amis.

A mes encadreurs pour m'avoir guidé tout au long de ce projet.

Aux membres de la commission pour avoir jugé mon modeste travail.

Et à tous ceux qui ont participé de près ou de loin à l'accomplissement de ce projet.

"Yes, there are two paths you can go by, but in the long run, there's still time to change the road you're on..."

Robert Plant.

Sommaire

LISTE DES TABLEAUX V

LISTE DES FIGURES VI

INTRODUCTION GENERALE 1

CHAPITRE I : INFORMATIQUE UBIQUITAIRE ET ORIENTATION SERVICE 4

I.1. INTRODUCTION 5
I.2. L'INFORMATIQUE UBIQUITAIRE 5
I.2.1. Définition et historique 5
I.2.2. Caractéristiques des environnements ubiquitaires 5
I.3. LES MIDDLEWARES DE COMMUNICATION 8
I.3.1. Définition et rôle d'un middleware 8
I.3.2. Les middlewares à objets répartis 9
I.3.2.1. CORBA (Common Object Request Broker Architecture) 9
I.3.2.2. RMI (Remote Method Invocation) 10
I.3.2.3. DCOM (Distributed Component Object Model) 10
I.3.3. Architecture orienté service : SOA (Service Oriented Architecture) 11
I.3.3.1. La notion de service 11
I.3.3.2. L'orientation service 12
I.3.3.3. La notion de SOA 13
I.3.3.4. Le principe de SOA 13
I.3.3.5. Les avantages de SOA 14
I.4. L'ORIENTATION SERVICE DANS LES ENVIRONNEMENTS UBIQUITAIRES 15
I.5. STANDARDS ET TECHNOLOGIES UTILISES 15
I.5.1. Les web services 15
I.5.1.1. Les standards des web services 16
I.5.1.2. Description sémantique des web services 18
I.5.2. Notion de répertoire de service 19
I.5.3. Découverte de service (passive et active) 19
I.5.4. Description de services 20
I.5.5. Invocation ou contrôle 20
I.5.6. La sélection de services 20
I.6. CONCLUSION 21

CHAPITRE II : SENSIBILITE AU CONTEXTE DANS LES ENVIRONNEMENTS UBIQUITAIRES 22

II.1. INTRODUCTION 23
II.2. NOTION DE CONTEXTE 23
II.2.1. Que disent les dictionnaires ? 23
II.2.2. Contexte et informatique ubiquitaire 24
II.2.3. Catégories de contexte 25
II.3. NOTION DE SENSIBILITE AU CONTEXTE 26

II.4. ARCHITECTURE D'UN SYSTEME SENSIBLE AU CONTEXTE 27
II.4.1. Couche de capture du contexte 28
II.4.1.1. Capteurs physiques 29
II.4.1.2. Capteurs virtuels 29
II.4.1.3. Capteurs logiques 29
II.4.2. Couche d'interprétation et agrégation du contexte 30
II.4.3. Couche de stockage et historique du contexte 30
II.4.3.1. L'approche attribut-valeur 31
II.4.3.2. Les approches orientées modèle 31
II.4.3.3. Les approches basées sur XML 31
II.4.3.4. Les approches basées sur les ontologies 32
II.4.4. Couche dissémination du contexte 32
II.4.5. Couche application 32
II.5. LES PRINCIPALES PLATEFORMES DE GESTION DE CONTEXTE 33
II.5.1. CAMidO 33
II.5.2. Context Toolkit 35
II.5.3. Gaia 36
II.5.4. SOCAM 37
II.5.5. CoBrA 38
II.5.6. MyCampus 39
II.6. Conclusion 41

CHAPITRE III : COMPOSITION DE SERVICES. 43

III.1. INTRODUCTION 44
III.2. DEFINITION DE LA COMPOSITION DE SERVICES 44
III.3. ARCHITECTURE GENERALE D'UN MODELE DE COMPOSITION DE SERVICES 45
III.4. CLASSIFICATION DES METHODES DE COMPOSITION DE SERVICES 47
III.4.1. Selon le degré d'interaction avec l'utilisateur 47
III.4.1.1. La composition manuelle 47
III.4.1.2. La composition semi-automatique 47
III.4.1.3. La composition automatique 48
III.4.2. Selon le mode de définition des schémas de composition 48
III.4.2.1. La composition statique 48
III.4.2.2. La composition dynamique 50
III.5. COMPOSITION DE SERVICE DANS UN ENVIRONNEMENT UBIQUITAIRE 56
III.5.1. Particularités et contraintes d'un environnement ubiquitaire 56
III.5.2. Quelques approches de composition de services 57
III.5.2.1. Composition de Web services base sur la planification adaptative pour la sensibilité au contexte 57
III.5.2.2. Un protocole de composition distribué pour les environnements prevasifs .. 59
III.5.2.3. Composition flexible de services d'objets communicants 60
III.5.2.4. Composition flexible de services d'objets communicants pour la communication ambiante 62

III.5.2.5. Approche de composition de service tolérante aux pannes en environnement ubiquitaire: WS-Pro/ASCT 63
III.5.2.6. Un modèle de QoS pour la composition de services base sur les requêtes ... 64
III.5.2.7. Approche de composition de services basé sur les règles pour les environnements ubiquitaires 64
III.5.2.8. Framework de composition de services base sur la QoS pour la robotique ubiquitaire 66
III.5.2.9. Approche de composition de services sensibles au contexte pour les environnements prevasifs 67
III.6. COMPARAISON 68
III.7. CONCLUSION 72

CHAPITRE IV : COMPOSITION FLEXIBLE DE SERVICES SENSIBLE AU CONTEXTE. 73

IV.1. INTRODUCTION 74
IV.2. SUPPOSITIONS DE DEPART 75
IV.3. MODELE DE CONTEXTE 75
IV.3.1. Description du contexte 75
IV.3.2. Structuration du contexte 75
IV.3.3. Modélisation du contexte 76
IV.4. ARCHITECTURE DU SYSTEME DE COMPOSITION DE SERVICES 78
IV.4.1. Principe de la composition flexible de services 78
IV.4.1.1. Modélisation du service concret 79
IV.4.1.2. Modélisation du service abstrait 80
IV.4.2. Mode de fonctionnement du système 80
IV.4.2.1. Formulation de la requête 81
IV.4.2.3. Sélection des services concrets et établissement du plan d'exécution 82
IV.4.2.4. Exécution du service composite 84
IV.4.3. Exécution automatique de certains services selon une base de règles 85
IV.4.4. Mécanisme de tolérance aux pannes 86
IV.5. CONCLUSION 87

CHAPITRE V : ASPECTS TECHNIQUES ET MISE EN ŒUVRE DE L'APPROCHE. 88

V.1. INTRODUCTION 89
V.2. CONCEPTION DU SYSTEME DE COMPOSITION 89
V.2.1. Concepts de base des agents 89
V.2.1.1. Les agents 89
V.2.1.2. La communication entre agents 91
a) Les actes de langage 91
b) Structure des messages 93
c) Le protocole 94
d) Le langage de contenue 94
V.2.2. Comportement des agents de l'application 95

V.2.2.1. Agent de planification 95
V.2.2.2. Agent de sélection 96
V.2.2.3. Agent d'exécution 98
V.2.2.4. Agent de scrutation 99
V.3. MISE EN ŒUVRE DE L'APPROCHE 100
V.3.1. Framework de gestion de contexte 100
V.3.2. JADE 100
V.3.2.1. Architecture logicielle de la plate-forme JADE 101
V.3.2.2. Comportement des agents dans la plate-forme JADE 101
V.3.3. Description de l'environnement de simulation 102
V.3.3.1. Les dispositifs disponibles 103
V.3.3.2. les services disponibles 104
V.3.4. Scénario 1 (cas normal) 105
V.3.5. Scénario 2 (illustrer la tolérance aux pannes) 108
V.3.6. Scénario 3 (illustrer le fonctionnement de l'agent de scrutation) 110
V.4. SYNTHESE 112

CONCLUSION GENERALE ET PERSPECTIVES 114

BIBLIOGRAPHIE 117

Liste des tableaux

Tableau 1: Exemples de capteurs physiques. 29
Tableau 2 : Tableau comparatif entre les différentes plateformes de sensibilité au contexte. 41
Tableau 3 : Tableau comparatif entre les différentes approches de composition de services. 69
Tableau 4 : Actes de communication du modèle FIPA-ACL, classés par catégories. 92
Tableau 5 : Eléments d'un message FIPA-ACL. 94

Liste des figures

Figure 1 : Middleware de communication. 8
Figure 2 : Architecture orientée service. 14
Figure 3 : Les standards des Web services 16
Figure 4 : Architecture générale d'un système sensible au contexte. 28
Figure 5 : Architecture de CAMidO. 34
Figure 6 : Architecture du middleware Context Toolkit de Anid K.Dey. 36
Figure 7 : Architecture du middleware GAIA. 37
Figure 8 : Architecture de la plateforme SOCAM. 38
Figure 9 : Architecture d'un espace actif basé sur l'agent CoBra. 39
Figure 10 : Architecture de MyCampus. 40
Figure 11 : Architecture générale d'un modèle de composition de web services. 45
Figure 12 : Web Service Orchestration. 49
Figure 13 : Chorégraphie de Web services. 49
Figure 14 : Le calcul de situation. 52
Figure 15 : Le problème de planification. 54
Figure 16 : Modélisation du contexte. 76
Figure 17 : Architecture générale du système. 78
Figure 18 : Processus de composition dynamique. 81
Figure 19 : Génération du plan abstrait d'une tâche. 82
Figure 20 : Génération du plan concret d'une tâche. 84
Figure 21: Exécution du service composite. 85
Figure 22 : Facettes d'un agent, du point de vue de la coordination. 90
Figure 23 : Les différents niveaux de langage dans la théorie de Searle. 91
Figure 24 : Algorithme de génération du plan abstrait. 96
Figure 25 : Algorithme de sélection du meilleur service concret. 97
Figure 26 : Algorithme de fonctionnement de l'agent d'exécution. 99
Figure 27 : Interactions entre les agents durant l'exécution du premier scénario. 107
Figure 28 : Schéma d'interaction entre les agents durant le déroulement du scénario 1 108
Figure 29 : Interactions entre les agents durant l'exécution du deuxième scénario. 109
Figure 30 : Schéma d'interaction entre les agents durant le déroulement du scénario 2. 110
Figure 31 : Fonctionnement standard de l'agent de scrutation. 111
Figure 32 : Schéma d'interaction entre les agents durant le déroulement du scénario 3. 112

Introduction générale

Introduction générale

L’informatique ubiquitaire se base sur un concept fondamental : avoir un monde numérisé chargé d’assister les utilisateurs dans leur quotidien, le tout, sans être intrusif. Cette vision porte néanmoins un paradoxe important qui est à la base de plusieurs travaux de recherche dans le monde entier. L’un des principaux défis des chercheurs est de faire en sorte que les ordinateurs soient suffisamment nombreux et interconnectés pour devenir invisibles dans l’environnement. Le fait d’avoir un grand nombre d’équipements offre une grande capacité de calcul et de communication, et cela permet de faciliter l’aide aux utilisateurs en toute situation et en tout lieu, tandis que les techniques de raisonnement sur les activités des utilisateurs garantissent la liberté souhaitée universellement. L'environnement quotidien doit devenir l'interface naturelle d'interaction avec l'utilisateur.

L’objectif principal de l’informatique ubiquitaire est de satisfaire les besoins de l’utilisateur. Cela est rendu possible grâce à la coopération des différents équipements électroniques de l’environnement. En effet, les besoins des utilisateurs sont souvent complexes et réaliser les tâches demandées exige parfois de pouvoir combiner plusieurs fonctionnalités afin de réaliser une seule et même tâche.

Les fonctionnalités des différents dispositifs peuvent être représentées sous forme de services. L’orientation service parait un choix évident dans le domaine de l’informatique ubiquitaire car elle permet de gérer les différentes fonctionnalités de manière indépendantetout en offrant la possibilité de les combiner et de les composer si nécessaire. En effet, les principales caractéristiques des architectures orientées services comme la distribution, le faible couplage des services ainsi que l’utilisation de standards pour la description et la communication fournissent une base pour la construction de systèmes ubiquitaires et l’utilisation de la composition de services. Cela permet de construire des systèmes flexibles et adaptables aux changements pouvant survenir dans l’environnement.

L’objectif de ce travail est de développer un mécanisme de composition qui fonctionne de manière dynamique et décentralisée, et cela dans un environnement ouvert et mobile. Le résultat recherché par un tel mécanisme est la satisfaction de la requête de l’utilisateur tout en prenant en compte les contraintes spécifiées par l’utilisateur et le contexte dans lequel cette application sera exécutée.

Les systèmes multi-agents sont l'outil idéal pour résoudre le problème de distribution de la composition de services dans les environnements ubiquitaires, ils offrent le compromis idéal entre coordination et indépendance des fonctionnalités.

Les agents sont au cœur du système.Ils seront chargés d'effectuer toutes les tâches nécessaires à l'accomplissement du processus de composition dont l'aboutissement est la génération d'un plan d'exécution composé de plusieurs services qui ,ensemble, vont exécuter l'application demandée par l'utilisateur et ce plan peut inclure l'exécution parallèle de certains services.

Pour contrôler si le processus d'exécution se déroule sans erreurs, nous devons disposer d'un mécanisme de contrôle et cette tâche sera également prise en charge par un agent qui pourra en cas d'erreurs réagir de façon réactive et proposer une alternative à un éventuel problème. Un tel mécanisme permet de rendre notre approche tolérante aux pannes.

Notre contribution est destinée à la domotique et plus particulièrement à l'aide à la personne.

Ce mémoire est divisé en cinq chapitres. Le premier chapitre est un état de l'art sur les environnements ubiquitaires. On y passe en revue les différents aspects qui touchent aux environnements ubiquitaires avec les défis et les solutions techniques existantes en détaillant quelques-uns des standards et technologies utilisés.

Le deuxième chapitre concerne la notion de contexte et de sensibilité au contexte dans les environnements ubiquitaires.Nous rappellerons, d'abord, les principales définitions de contexte dans la littérature classique et scientifique et donnerons les principales catégories de contexte. Nous nous intéresserons, ensuite, à la sensibilité au contexte et aux différentes approches de modélisations et, enfin,nous étudierons quelques middlewares de gestion de contexte.

Le troisième chapitre est consacré à la composition de services.Nous définirons d'abord ce concept et décrirons les principales techniques utilisées et les contraintes que l'on peut rencontrer dans l'adaptation aux environnements ubiquitaires. Par la suite, nous présenterons quelques approches de composition de services en environnement ubiquitaire.

Dans le quatrième chapitre, nous décrirons, d'une part, l'approche de composition dynamique que nous proposons et, d'autre part, notre modélisation du contexte, les principaux aspects de notre approche ainsi que les différents types d'agents.

Le cinquième chapitre est, quant à lui, consacré à la mise en œuvre de notre approche de composition.Nous y décrirons les techniques utilisées ainsi que les algorithmes qu'utilise chaque agent de l'architecture et exposerons des scénarii qui permettent d'évaluer le comportement de l'approche de composition dans différents cas.

Nous terminerons par une conclusion générale qui ouvre sur quelques perspectives.

Chapitre I : Informatique ubiquitaire et orientation service

I.1. Introduction

Avec l'arrivée de la miniaturisation, de la mobilité et de l'accès ausans fil, l'informatique omniprésente et invisible, appelée également informatique ubiquitaire (Ubiquitous Computing), est apparue.

L'informatique ubiquitaire est une vision du futur proche, dans laquelle un nombre croissant d'appareils (capteurs, processeurs, actionneurs) inclus dans divers objets physiques participent à un réseau d'information global. La mobilité et la reconfiguration dynamique sont des traits dominants de ces systèmes, imposant une adaptation permanente des applications. Les principes d'architecture applicables aux systèmes d'informatique ubiquitaire font partie des champs de recherche actuels dans le domaine de l'informatique.

L'informatique ubiquitaire a donc pour objectif d'intégrer les technologies informatiques au quotidien de l'homme le plus simplement possible. Plus précisément, elle devrait rendre familier et instinctif l'outil informatique et en faciliter l'utilisation dans de nombreux domaines tels que l'information, la formation, le transport et dans tout ce qui touche à l'amélioration du confort des utilisateurs.

Ce chapitre est un état de l'art sur les composants des différentes couches constituant l'architecture d'un environnement ubiquitaire.

I.2. L'informatique ubiquitaire

I.2.1. Définition et historique

Mark Weiser est le père de l'informatique ubiquitaire, il dessine la structure générale de ce type d'environnements. Pour lui, le déploiement et l'organisation des équipements électroniques a pour principal objectif l'assistance de l'utilisateur dans ses activités du quotidien. [1]

Apres lui, Mahadev Satyanarayanan, éditeur en chef de la revue « IEEE Prevasive Computing » définit la notion principale de l'informatique ubiquitaire, qui est la construction d'environnements qui ont des capacités de calcul et de communication intégrées de façon transparente et harmonieuse et qui seront utilisées par l'humain. [2]

I.2.2. Caractéristiques des environnements ubiquitaires

La principale caractéristique d'un environnement ubiquitaire est son ouverture. En effet, plusieurs types d'entités qui intègrent des équipements électroniques et logiciels variés et qui

évoluent de façon dynamique peuvent appartenir à un même environnement. N'importe quel réseau connecté peut être considéré comme un environnement ubiquitaire dès lors que les applications qui tournent sur ce réseau impliquent l'interaction de plusieurs types d'équipements électroniques ayant une forte mobilité. D'autres caractéristiques des environnements ubiquitaires peuvent être citées :

- **L'hétérogénéité :** elle peut être matérielle, architecturale ou protocolaire(les protocoles de communication).

  Un environnement ubiquitaire est composé de plusieurs logiciels, développés dans des langages différents et déployés sur plusieurs plateformes sans compter le fait que les dispositifs sont très variés et hétérogènes (ordinateurs, téléphones…etc.). Chaque dispositif a ses propres spécifications (capacité, configuration matérielle) et chaque dispositif s'exécute sur une plateforme différente. Tous ces dispositifs doivent interagir dans l'environnement de la façon la plus transparente possible et ce, malgré l'hétérogénéité matérielle et logicielle. Pour ce faire, il faut une infrastructure pour maintenir les connaissances sur les caractéristiques des dispositifs et qui va gérer l'intégration de nouveaux dispositifs dans le système de manière cohérente. Pour faire communiquer les différents dispositifs entre eux, on utilise une infrastructure réseau hybride qui peut contenir des connexions filaire ou sans fil. Cette infrastructure est contraignante à cause de l'aspect dynamique d'un environnement ubiquitaire, ce qui provoque de fréquents changements de topologies.En plus de cela, il y a la grande variété de dispositifs et de protocoles de communications utilisés.
- **La dynamique :** Dans la mesure où un environnement ubiquitaire est mobile, cela le rend dynamique. De nouveaux dispositifs peuvent rejoindre le système à tout moment, et d'autres peuvent le quitter. On peut aussi perdre la connexion avec un dispositif, cette perte de connexion peut être due soit à un départ du dispositif, soit à un problème de réseau. Cela introduit le problème du maintien de la connectivité et la manière de faire face aux déconnexions fréquentes.

  Pour résoudre le problème de la dynamique des dispositifs et des serviceset, ainsi, permettre aux membres du système de profiter de toutes les fonctionnalités qu'il peut offrir, il faudrait pouvoir intégrer dynamiquement les ressources. En effet, une application de l'informatique ubiquitaire doit être capable de s'adapter aux besoins évolutifs des utilisateurs.

- **La distribution :** La philosophie de l'informatique ubiquitaire se base sur les systèmes distribués. Plusieurs middlewares de communication tels que CORBA et les services web permettent de rendre transparente la distribution des communications. Cependant, ils ne prennent pas en compte la mobilité, la synchronisation et la coordination.
- **Contraintes sur les ressources :** les dispositifs qui font partie de ces environnements sont en général des terminaux mobiles qui souhaitent exécuter des tâches en utilisant les dispositifs et les services disponibles dans l'environnement. Ces dispositifs sont limités en terme de ressources, capacité de stockage, affichage,…etc. Ces limitations influent sur le développement des applications. Ces dernières doivent être capables de s'adapter aux changements de capacités matérielles et à l'hétérogénéité des services disponibles.
- **Limitations réseaux :** la plupart des connexions utilisées dans ce type de réseau sont sans fil, ce qui limite la bande passante et la rend instable. Cette lenteur est une barrière empêchant l'exécution des applications gourmandes en débit ou de transférer des données volumineuses sur les dispositifs mobiles.
- **La sensibilité au contexte :** La sensibilité au contexte est une caractéristique importante de l'informatique ubiquitaire. La mobilité de ce type d'environnement provoque des changements dans le contexte des applications, ce qui nécessite le développement d'applications sensibles au contexte. Une application sensible au contexte doit s'adapter dynamiquement aux changements dans le contexte. Cela implique que, dans différents contextes, les utilisateurs accèdent aux même données et aux mêmes services mais reçoivent des réponses qui peuvent être différentes ou présentées différemment ou à des niveaux de détails différents. Beaucoup de travaux ont été faits dans cette direction.Nous détaillons cela dans le chapitre 2.

Les caractéristiques des environnements ubiquitaires font de la satisfaction des besoins fonctionnels des tâches de l'utilisateur un objectif difficile à atteindre. En effet, les réponses aux requêtes des utilisateurs doivent être dynamiques et dépendantes des dispositifs disponibles dans l'environnement à un instant donné. Les réponses doivent satisfaire les requêtes efficacement en répondant aux besoins fonctionnels tout en respectant les contraintes des ressources matérielles et en s'adaptant à l'hétérogénéité des technologies déployées dans l'environnement. Tous ces problèmes d'hétérogénéité dus aux matériaux et aux infrastructures de communication, de ressources et de réseaux font l'objet de plusieurs

travaux de recherches. Certains ont abouti à la proposition des middlewares de communication (CORBA, DCOM, …). Dans la prochaine section, nous présenterons quelques-uns de ces middlewares.

I.3. Les middlewares de communication

I.3.1. Définition et rôle d'un middleware

Comme cité précédemment, un environnement ubiquitaire est composé de plusieurs dispositifs hétérogènes formantensemble un système distribué à large échelle qui est ouvert et dynamique. Afin de gérer les différences abstraites entre les besoins de haut niveau des utilisateurs et les fonctionnalités que fournissent les équipements, il faut que les communications de l'environnement soient flexibles et modulaires. L'adaptation aux activités de l'utilisateur sera ainsi le résultat d'une construction dynamique de l'application à partir des ressources appropriées et d'une reconfiguration continuelle des applications pour s'adapter aux changements de l'environnement [7]. Mettre en place ce genre de système nécessite la prise en compte de toutes les caractéristiques dont nous avons parlé dans la section précédente.

Un middleware est la couche logicielle intermédiaire qui fournit un haut niveau d'abstraction de programmation. Il permet de masquer l'hétérogénéité des réseaux de communication, des ressources matérielles, des systèmes d'exploitation et des langages de programmation. Un middleware se trouve entre les couches inférieures et les applications de haut niveau.La figure ci-dessous en est l'illustration :

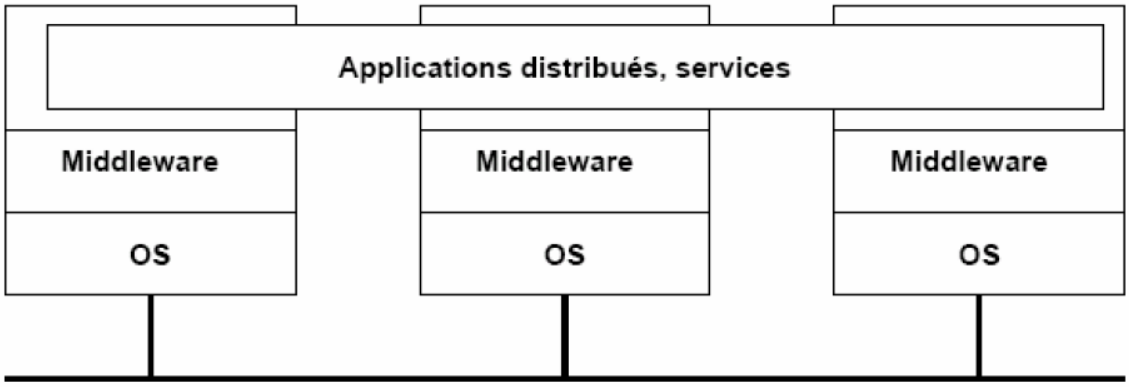

Figure 1 : Middleware de communication.

Dans ce qui suit, nous commencerons par présenter quelques middlewares à objets distribués suivi de à l'architecture SOA.

I.3.2. Les middlewares à objets répartis

I.3.2.1. CORBA (Common Object Request Broker Architecture)

CORBA est une architecture à objets distribués spécifiée par le groupe OMG (Object Management Groupe) [9]. C'est une norme de communication qui est utilisée pour l'échange entre objets logiciels hétérogènes. Un langage IDL (Interface Definition Language)décrit les traitements effectués et les formats de données en entrée et en sortie. Un bus applicatif, ORB (Object Request Broker*)* constitue le cœur de CORBA par lequel les requêtes sur les objets transitent.

Le principal rôle de l'ORB est de répondre aux requêtes d'une application ou d'un autre ORB. C'est donc un genre de bus par lequel transitent les objets distribués entre clients et serveurs. Cette communication reste cependant totalement transparente pour le développeur CORBA. En effet, une fois l'ORB initialisé, les objets distribués s'utilisent comme s'ils étaient locaux, quel que soit le langage dans lequel ils ont été implémentés. Assurer l'interopérabilité entre objets implantés sur des plates-formes hétérogènes nécessite de disposer d'un mécanisme permettant de décrire l'interface de ces objets indépendamment de leur implémentation. C'est là le rôle du langage IDL. CORBA qui comprend deux types de modèles : le modèle objet et le modèle composant :

- Le modèle objet : Lorsqu'un client invoque un objet distant, une implémentation de l'interface de l'objet, appelé souche (stubs) permet d'empaqueter les paramètres de la méthode invoquée et d'appeler l'objet distant en passant en premier par les squelettes (skeletons) qui font l'opération inverse (dépaquetage des données), et transmet l'appel à l'interface de l'objet. Au retour, les données sont empaquetées par les squelettes et transmis au client par l'intermédiaire de la souche qui dépaquette les résultats fournis.
- Le modèle composant : La notion de composant étend la notion d'objet et se place à un niveau plus haut de modularité. Cette notion est apparue après le constat que l'approche objet a une granularité très fine, qui la rend très peu adaptée aux systèmes complexes et aux systèmes distribués en particulier. Ces derniers sont composés de plusieurs entités coopérants entre elles, et une fine granularité rendrait la conception et la maintenabilité difficiles, surtout si le développement de l'application est réalisé par plusieurs parties. Le modèle composant de CORBA CCM (Corba Component Model)

qui est compatible avec les EJB (Enterprise Java Beans) décrit le déploiement, la configuration et la composition des composants.

I.3.2.2. RMI (Remote Method Invocation)

RMI [10] permet de créer des applications Java distribuées. Ces dernières implémentent les méthodes des objets Java distants qui peuvent être invoquées à partir de n'importe quelle machine qui possède la JVM (Java Virtual Machine).Un programme Java peut faire un appel à un objet distant une fois qu'il a obtenu sa référence, soit en cherchant dans un annuaire de services fourni par RMI, ou bien en recevant la référence comme argument en retour d'un appel. Les souches et squelettes, qui représentent respectivement le client et le serveur, sont des classes compilées par le compilateur RMI (RMIC). Un client peut alors invoquer un objet distant qui se trouve dans un serveur. Le serveur lui-même peut devenir client pour d'autres objets distants. RMI utilise la sérialisation d'objets pour empaqueter et dépaqueter les paramètres. Le protocole utilisé pour la communication est JRMP (Java Remote Methode Protocol).

La programmation avec RMI est facile une fois que le développeur a acquis une certaine expérience avec le langage de programmation Java et les applications distribuées. Il ne contient pas un langage abstrait tel qu'IDL pour décrire les objets distants. De plus RMI supporte un nettoyage distribué des objets en mémoire. Cependant, RMI peut être utilisé uniquement avec le langage Java de part et d'autre de la connexion. Et la facilité d'utilisation du RMI provient du fait que ce middleware est maintenu simple et ne fournit pas des services contrairement à CORBA.

I.3.2.3. DCOM (Distributed Component Object Model)

DCOM [11] est une version distribuée du modèle de Microsoft COM[12] qui a été créé pour permettre la communication entre les applications Windows dans un environnement composite. DCOM permet d'appeler des objets distants en utilisant une couche de haut niveau au-dessus du mécanisme DCE RPC (Remote Procedure Call)qui interagit avec les services de COM. Un serveur DCOM publie ses méthodes pour les clients sous forme d'interfaces. Ces dernières sont écrites en IDL qui est similaire au langage C++. Le compilateur IDL crée des souches et des squelettes comme le compilateur IDL de CORBA. Mais la particularité de DCOM c'est que toutes les configurations de ses applications sont enregistrées dans la base de registre du système. L'utilisation de cette base de registre et le fait que DCOM soit une

extension de COM rendent exploitables les applications COM dans un environnement réparti. De plus, des librairies sont créées sous forme de fichiers qui décrivent l'objet distant qui peut être invoqué par le mécanisme COM. Un autre élément essentiel dans l'architecture DCOM est le Service Control Manager qui permet de gérer les composants du système (localisation, activation, enregistrement de référence). Le protocole utilisé est ORPC (Object Remote Procedure Call).

Le niveau de spécification binaire de DCOM permet l'utilisation de plusieurs langages pour coder les objets du serveur. Les interfaces binaires de DCOM peuvent être considérées comme des tables de pointeurs vers les implémentations des méthodes de l'interface. Comme RMI, DCOM supporte le nettoyage distribué des objets distants en mémoire. Cela est réalisé par un mécanisme qui vérifie si le client est encore actif. Du coté serveur, un compteur de référence est maintenu pour les clients. S'il atteint zéro, l'objet sera détruit. La plupart des utilisateurs associent DCOM au système d'exploitation Windows. Toutefois, il existe des ports pour les plateformes Unix, VMS, Apple, et Macintosh.

Les middlewares que nous venons de présenter reposent sur des infrastructures bien particulières qui nécessitent que les dispositifs se conforment à des contraintes spécifiques. Cependant, les dispositifs constituant les environnements ubiquitaires peuvent être d'origines très diverses et il est nécessaire de prendre en compte leur hétérogénéité. Par ailleurs, une propriété essentielle à la mise en place de systèmes à grande échelle est la capacité d'ouverture, c'est-à-dire la possibilité d'intégrer facilement avec un dispositif qui n'a pas été conçu spécifiquement pour fonctionner dans ce système.

I.3.3. Architecture orienté service : SOA (Service Oriented Architecture)

Le problème des systèmes repartis hétérogènes et ouverts est en général abordé dans le cadre des architectures orienté service (SOA). SOA n'est certes pas un concept récent, mais l'arrivée des services web lui a redonné un second souffle et a fait d'elle un outil très important pour le développement d'applications distribuées [8]. En effet, même si plusieurs standards ont été créés pour répondre aux problèmes d'intégration, le coût de la mise en œuvre de ces middlewares étant très important.

I.3.3.1. La notion de service

Un service est une brique logicielle autonome qui fournit une fonction bien définie [13]. Cette fonction peut être par exemple une analyse ou recherche d'informations,...etc. La notion

d'autonomie d'un service concerne le fait qu'il puisse se suffire à lui-même, il peut néanmoins être publié et rendu disponible pour que quiconque puisse l'utiliser. Ainsi des services peuvent être utilisés seuls ou être composés pour mener à terme un processus complexe et atteindre un objectif précis de plus haut niveau. D'un point de vue implémentation, un service peut être implémenté en utilisant différents paradigmes : objet, composant ou encore agent. Il est décrit et utilisé indépendamment de sa réalisation grâce aux mécanismes d'encapsulation et de liaison retardée. La liaison proprement dite n'est effectuée qu'au moment de l'exécution. Il est décrit par son descripteur de service, cette description est en fait sa spécification. Le descripteur est composé de trois types d'informations :

- **Informations fonctionnelles** : la sémantique des opérations, le comportement du service (pré-conditions, post-conditions, constantes, exceptions, propriétés), l'interface du service.
- **Informations non fonctionnelles** : prix, politique, paramètres de QoS, information de déploiement, etc.
- **Informations additionnelles** : composées d'informations sur le service, non spécifiées par le fournisseur du service telles que les notes, les rapports d'utilisation, etc.

I.3.3.2. L'orientation service

La principale notion de l'orienté service est le faible couplage entres les différentes applications de l'environnement, la définition de l'orientation service diffère d'un point de vue à l'autre. Certains considèrent par exemple les avantages de la neutralité technologique, le faible couplage et la transparence de la localisation. Alors que la neutralité technologique dépend du choix des web services, la transparence de la localisation n'est pas vraiment un critère qui définit réellement l'architecture orientée service.

Le faible couplage reste néanmoins la principale caractéristique qui définit toute architecture orientée services. [5]

[6] décrit l'orientation service selon trois critères fondamentaux qui assurent le faible couplage :

- Abstraction : l'interface d'un service masque ses fonctionnalités et ses aspects non fonctionnels.

- Dynamique : la liaison tardive entre les services permet une composition dynamique.
- Distribution de l'administration : plusieurs entités chargées de l'administration du réseau permet de renforcer le caractère temporaire des liaisons entre les services (applications).

I.3.3.3. La notion de SOA

L'architecture SOA est un modèle ou un style d'architecture qui répond bien aux problèmes d'interopérabilité entre applications dans des environnements distribués.

Cette architecture convient parfaitement aux environnements ubiquitaires, car elle permet d'utiliser les composants logiciels hétérogènes présents dans l'environnement de manière homogène.

Les ressources logicielles qui sont disponibles dans l'environnement sont représentées comme des services.

I.3.3.4. Le principe de SOA

SOA [13] est une architecture composée de clients et de fournisseurs de services qui interagissent. Un client communique avec un service soit avec des interactions synchrones ou asynchrones selon le contrat du service, ce contrat étant exprimé dans un langage déclaratif indépendant des langages de programmations. Le client n'a pas un accès direct au service mais plutôt à une description de ses caractéristiques et fonctionnalités. En utilisant cette description, les clients peuvent localiser et découvrir dynamiquement les services dans l'environnement en interrogeant un service de découverte (SDP : Service Discovery Protocol) [14]. Cette découverte peut être centralisée ou distribuée [15]. Dans le cas centralisé, elle se fait à l'aide d'un annuaire contenant les descriptions des services disponibles dans l'environnement. Dans le cas distribué, les fournisseurs de services participent au processus de découverte en suivant soit le modèle de découverte passif ou actif.

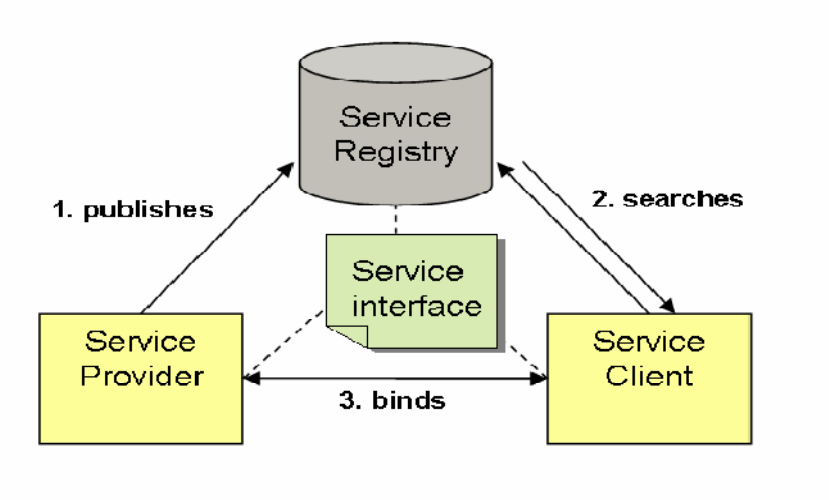

Figure 2 : Architecture orientée service.

I.3.3.5. Les avantages de SOA

L'architecture SOA offre trois principaux avantages :

- **L'interopérabilité**

SOA permet de faire communiquer les clients et les services quelle que soit la plateforme sur laquelle ils sont exécutés. Cela peut ne être réalisé que si les clients et les services sont reliés par un moyen de communication standard. Cela est rendu possible en introduisant les services web dans SOA. Les services web offrent une panoplie de protocoles et de technologies largement utilisés sur le web.

- **La *scalabilité* (passage à l'échelle)**

L'un des principaux avantages de l'architecture SOA est le faible couplage entre les services. En effet la dépendance entre services est quasiinexistante. Cela fait que les applications utilisant ces services peuvent être facilement *scalables* beaucoup plus facilement que dans un environnement fortement couplé et cela grâce au fait qu'il y a moins de dépendance entre l'application et les services qu'elle utilise.

- **La flexibilité**

Dans une architecture fortement couplée, les différents composants de l'application sont fortement liés entre eux, partageant les mêmes sémantiques et bibliothèques. Ceci rend l'éventuelle évolution d'une application difficile afin qu'elle prenne en compte des changements technologiques ou des exigences de métiers. Le faible couplage qu'offre SOA permet aux applications d'être flexibleset facilement adaptables aux éventuels changements.

I.4. L'orientation service dans les environnements ubiquitaires

La philosophie de conception des systèmes ubiquitaires implique de faire interagir plusieurs entités dynamiquement pour fournir des services aux clients. Cela implique une grande hétérogénéité des composants dans l'environnement. Cette hétérogénéité peut être logicielle, comportementale ou contextuelle :

- hétérogénéité logicielle : chaque application a été implémentée avec un langage de programmation différent et intègre ses propres protocoles de communication.
- hétérogénéité comportementale : le fonctionnement d'une entité peut interférer sur celui d'une autre ; il n'a aucune homogénéité des comportements ou des algorithmes de fonctionnement.
- hétérogénéité contextuelle : le contexte des applications est changeant. L'utilisateur, les équipements et les éléments environnants infèrent sur le contexte.

Les concepts de l'architecture orientée service sont donc appliqués dans le domaine de l'informatique ubiquitaire afin de répondre à toutes ces contraintes. En effet, les dispositifs peuvent être modélisés comme des fournisseurs de service ou des demandeurs de service.

I.5. Standards et technologies utilisés

I.5.1. Les web services

Le terme Web service est très utilisé de nos jours, si on se réfère à [16] : « les Web services sont des applications modulaires accessibles via le Web construites avec des langages standards, fournissant plusieurs fonctionnalités pour des utilisateurs simples ou pour le business ». Cette définition est très simple ; néanmoins, elle manque de précision. Par exemple, elle s'applique parfaitement aux composants de CORBA, pourtant, ce ne sont pas des Web services.

Un raffinement de cette définition est donné par W3C (World Wide Web Consortium) [17] : « Un Web service est un système logiciel identifié par un URI et dont l'interface publique et les constructeurs sont définis et décrits en utilisant le langage XML. Sa définition peut être découverte par d'autres systèmes logiciels. Ces systèmes peuvent alors interagir avec

le Web service d'une manière prédéterminé dans sa description via des messages basés sur XML et convoyés par des protocoles de communication internet ».

I.5.1.1. Les standards des web services

Plusieurs standards basés sur XML [18], [19] ont été développés (figure 3) pour implémenter l'architecture Web service précédemment décrite. Ces standards sont présentés dans la figure ci-dessous :

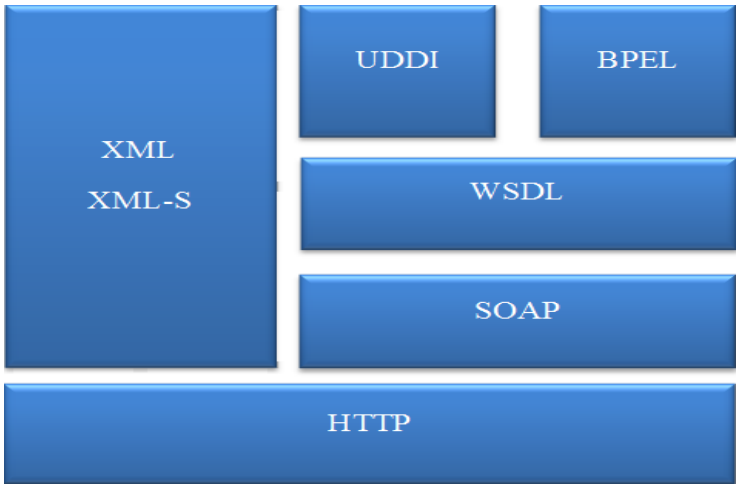

Figure 3 : Les standards des Web services [18].

- **XML-S**

XML Schema [17] offre une infrastructure sous-jacente pour la définition des standards Web services et des variables/objet/type de donnée…etc, qui sont échangés entre les services.

- **UDDI**

Universal Discovery, Description and Integration [20] est une plateforme indépendante représentant un registre basé sur XML où les fournisseurs de services peuvent enregistrer et publier leurs services.

Un registre UDDI est constitué de trois composants : les pages blanches, les pages jaunes et les pages vertes. Les pages blanches regroupent les informations de contact et d'adresse d'un service ; les pages jaunes donnent un classement thématique standardisé des différents types de services disponibles ; les pages vertes sont utilisées pour décrire la manière d'accéder au Web service.

- **SOAP**

Simple Object Acces Protocol [17] est un standard W3C pour l'échange d'informations dans un environnement décentralisé et distribué. C'est un protocole basé sur XML qui est constitué de trois parties : une enveloppe, un registre d'encodage et RPC. L'enveloppe définie un Framework pour décrire le contenu du message et la manière dont ce contenu doit être traité. Les règles d'encodage offrent un mécanisme de sérialisation pour des échanges d'instances d'applications avec des types de données préalablement définies ; Finalement, RPC (Remote Procedure Call) est utilisé pour représenter des requêtes d'appel de procédure et les réponses. SOAP peut être utilisé en combinaison avec une variante d'autres protocoles comme http.

- **WSDL**

Web Service Description Language [17] est un standard de description W3C. Un service y est décrit comme un ensemble de points d'entrées dans un réseau interagissant à travers des messages. Ce langage permet la description du protocole de communication utilisé, le format des messages requis pour communiquer avec le service, les méthodes que le client peut invoquer et la localisation du service. Les Web services sont décrits en deux phases : une phase abstraite et une phase concrète. WSDL est extensible pour la description des points d'entrées sans prendre en considération les formats des messages ou les protocoles de communication utilisés.

- **BPEL4WS**

Un service peut être définie comme un Workflow décrivant la chorégraphie de plusieurs opérations, chaque Workflow peut déterminer l'ordre d'exécution des opérations, quel opération est exécutée à l'instant courant, et un chemin d'exécution alternatif (si des opérateurs conditionnels sont inclus dans le langage de modélisation du Workflow). Par convention, des Workflows sont requis pour orchestrer l'exécution de plusieurs services qui peuvent être composés pour former un service plus complexe. Plusieurs langages de chorégraphie et d'orchestration ont été proposés, l'un d'entre eux est BPEL4WS (Business Process Execution Language For Web Services) [21] qui combine les standards XLANG [22] et WSFL (Web Service Flow Language) [23].

I.5.1.2. Description sémantique des web services

Comme expliqué dans [17], [24] le standards WSDL fournit uniquement une description de niveau syntaxique et manque de l'expressivité sémantique nécessaire à la représentation des besoins et capacités des Web services. Les Web services sémantiques offrent des descriptions riches en sémantique des capacités des Web services. La description sémantique de services (coût, temps d'exécution, format des entrées/sorties..) peut faciliter la composition automatique, la découverte et l'invocation des Web services dans des environnements ouverts.

La description sémantique des Web services est réalisée à travers des ontologies de services qui permettent à la machine d'interpréter leurs capacités et d'interagir avec un domaine de connaissance [18]. Le terme ontologie est défini comme une description formelle et explicite de concepts dans un domaine de discours et de relations entre ces concepts [25]. L'ontologie permet au programmeur de spécifier d'une manière ouverte et expressive des concepts et des relations qui, collectivement, caractérisent un domaine.

Il y a aujourd'hui différents langages de spécification d'ontologies avec des degrés variant d'expressivité, comme RDF (Resource Description Framework) [17], DAML (DARPA Agent Markup Language) [26], DAML+OIL [26] et OWL (Web Ontology Language) [17]. Les descriptions sémantiques de Web services sont basées sur ces langages.

Trois grandes approches ont dirigé le développement d'infrastructures de Web services sémantiques : IRS-II [27], WSMF [28] et OWL-S [17].

- IRS-II (Internet Reasoning Service) est une approche basée sur la connaissance pour les Web services sémantiques : elle est née à partir de la recherche sur les composants réutilisables de la connaissance [29].
- WSMF (Web Service Modeling Framework) est une approche orientée business pour les Web services sémantiques : elle est centrée sur les besoins du e-commerce.
- OWL-S (Web Ontology Language for Web Services) est une ontologie de services qui permet à des utilisateurs et à des agents logiciels de découvrir, d'invoquer, de composer et de gérer des ressources Web offrant des services particuliers et ayant des propriétés particulières. L'ontologie est constituée de trois parties : le ServiceProfile, le ServiceModel et le serviceGrounding. Le ServiceProfile décrit ce que fait le service,

le ServiceModel, comment le service marche et le ServiceGrounding, comment accéder au service.

I.5.2. Notion de répertoire de service

Les services étant très nombreux dans un environnement, nous devons disposer d'un espace pour que les utilisateurs de cet environnement puissent les localiser et y accéder facilement. Les répertoires de services dits annuaires de services sont en fait des serveurs contenant une partie des services ou du moins leurs descriptions.Il peut y avoir plusieurs répertoires de services dans un même environnement.

Un répertoire de service ou plusieurs répertoires de services peuvent être déployés selon les exigences techniques visées par le protocole :

- Fiabilité par redondance : déploiement de plusieurs répertoires identiques. Les fournisseurs doivent enregistrer leurs services dans tous les répertoires découverts
- Passage à l'échelle et séparation de domaine d'administration par spécification : spécialisation des répertoires déployés dans un domaine d'administration donné
- Passage à l'échelle par hiérarchisation : la hiérarchisation permet le passage à l'échelle de grands réseaux (internet par exemple).

I.5.3. Découverte de service (passive et active)

Un protocole de découverte active définit les requêtes qu'envoie le demandeur de service au répertoire de service. Il définit aussi les réponses que le demandeur reçoit de manière synchrone de la part du répertoire de service ou, directement de la part de tous les fournisseurs correspondant à la requête active.

La découverte est dite passive si le demandeur de service peut écouter les publications spontanées de services. Deux situations se distinguent :

- Le protocole de découverte définit un mode de souscription aux événements de publications auprès du registre de services. Le répertoire de services garde en mémoire les requêtes indiquées persistantes et envoie une réponse de manière asynchrone dès qu'un service publié correspond à la requête.
- Le protocole de découverte indique que les fournisseurs de services publient spontanément leurs services dans les annuaires. Les demandeurs de services doivent donc s'abonner aux mises à jour de ces répertoires pour recevoir passivement les annonces

I.5.4. Description de services

La phase de description est nécessaire dans les middlewares où les clients et serveurs ne se connaissent pas au préalable. Elle est pertinente en Architecture Orientée Service puisque la recherche d'un faible couplage repose sur la définition publique de tous les requis du client par rapport à un serveur.

La plupart des middlewares de communication répartie décrivent les entités à l'aide d'interfaces, qui sont dites "interfaces de service" en orientation service. Une interface de service est un ensemble d'opérations abstraites. Une opération abstraite est une opération sans son implémentation, c'est-à-dire la définition de son nom, de ses arguments typés dans un espace de nommage. Une interface de service mentionne généralement des espaces de noms utilisés, des types, des messages des opérations ou encore des évènements. D'autres éléments peuvent être disponibles comme des variables d'états…etc.

I.5.5. Invocation ou contrôle

Les opérations d'un service sont appelées par le client par appel distant selon un protocole de communication. SOAP, RMI sont des exemples connu de protocoles d'appels d'opérations distantes au-dessus de TCP/IP. Les middlewares de communication répartie spécifient le contenu des en-têtes et du message utile – payload – de ces protocoles pour les messages d'entrée, les messages de retour et les messages d'erreur. Par exemple, UPnP, utilise SOAP, Jini utilise RMI, Corba recommande IIOP et RMI, etc.

I.5.6. La sélection de services

Le caractère ouvert des environnements exacerbe le besoin des applications en dynamisme et de comportement autonomiques. Ces environnements sont caractérisés par la variabilité et la mobilité des entités actives environnantes. Sélectionner dynamiquement des services tout en garantissant la continuité d'exécution aux utilisateurs est un défi important de l'Informatique ubiquitaire.

La sélection de services est basée sur les paramètres de contexte et de QoS. Elle est locale si elle concerne un service atomique et globale lorsqu'il s'agit de composer plusieurs services.

L'objectif de ce travail est justement de concevoir un mécanisme de composition dynamique de services dans un environnement ubiquitaires tout en prenant en charge les paramètres de contexte et de QoS.

I.6. Conclusion

Dans ce chapitre, nous avons présenté un panorama général de ce qu'est un environnement ubiquitaire, en expliquant d'abord l'ubiquité, puis nous avons décrit quelques points historiques de ces environnements et enfin étudier les contraintes de ces environnements.

Nous avons ensuite vu quelques middlewares qui tentent de résoudre le problème d'hétérogénéité dans ces environnements : nous avons vu qu'il y avait des middlewares à objet repartis (CORBA, RMI, DCOM) et des approches orienté service ; dans ce sens, nous avons opté pour l'architecture orientée service car elle offre des avantages compatibles avec les environnements ubiquitaires.

Ayant choisi les architectures SOA comme style de programmation, nous avons défini les principales composantes de ces architectures (la description de services, l'invocation, la découverte de services, et les répertoires de services).

Il est vrai que les architectures orientées services assurent l'interopérabilité et la distribution dans les environnements ubiquitaires ; néanmoins, elles ne traitent pas la notion de sensibilité au contexte qui est un des paramètres principaux de l'informatique diffuse. Pour répondre à cette difficulté, plusieurs travaux de recherches sont consacrés à la gestion du contexte. Les aboutissements de ces travaux sont des middlewares de sensibilité au contexte. Le chapitre suivant est un état de l'art sur la gestion de contexte et de la sensibilité au contexte.

Chapitre II : sensibilité au contexte dans les environnements ubiquitaires

II.1. Introduction

L'avancée rapide de la technologie fait que l'informatique ubiquitaire a fait des pas de géant ces dernières années. La plupart des applications ubiquitaires sont très sensibles au contexte environnant dans le sens où celui-ci peut affecter le fonctionnement même des systèmes,ce qui a accéléré la nécessité de s'intéresser à la sensibilité au contexte. Il n'en reste pas moins que ces systèmes restent assez difficile à concevoir et mettre en place. En effet, avant de mettre en place une quelconque application sensible au contexte, nous devons considérer les questions fondamentales liées à la sensibilité au contexte qui sont qu'est-ce que le contexte, comment l'acquérir, comment le modéliser et adapter cette application au contexte.

Ce chapitre est un état de l'art dans le domaine de la sensibilité au contexte des applications de l'informatique ubiquitaire. On y trouve une définition de chacune des notions abordées plus haut. Nous aborderons également les différentes difficultés auxquelles il faut faire face dans ce type de système.

II.2. Notion de contexte

II.2.1. Que disent les dictionnaires ?

Parmi les dictionnaires de référence nous citons les définitions suivantes de la notion de contexte :

- Le Petit Robert : "ensemble des circonstances dans lesquelles s'insère un fait".
- L'encyclopédie Larousse : "ensemble des conditions naturelles, sociales, culturelles dans lesquelles se situe un énoncé, un discours"; ou encore : "ensemble des circonstances dans lesquelles se produit un événement, se situe une action".
- Hachette Multimédia : "ensemble des éléments qui entourent un fait et permettent de le comprendre".

Ces définitions partagent l'idée d'ensemble d'informations associé à quelque chose et la nature même de ce « quelque chose » dépend de l'utilité du contexte ; et cette utilité nous permet de comprendre, d'agir en conséquence ou de donner un sens aux données contextuelles.

II.2.2. Contexte et informatique ubiquitaire

Dans le domaine de l'informatique ubiquitaire, la notion de contexte est un domaine très populaire chez les chercheurs. En effet, chaque chercheur a donné sa propre définition basée sur ses antécédents de recherches.

Schilit et Theimer [30] furent en1994, les premiers à proposer une définition du contexte comme étant la localisation de l'utilisateur, les identités et les états des personnes et objets qui l'entourent.

Brown, Bovey et Chen [31] proposent : en 1997 une définition semblable : *«le contexte est l'identité de l'utilisateur, des personnes et objets qui l'entourent, sa localisation géographique, son orientation, la saison, la température où il évolue...* » .

Ryan, Pascoe, et Morse [32] définissent en 1998 le contexte en tant que la localisation, l'environnement, l'identité, et le temps de l'utilisateur.

Dey et Abowd [33] en 2000 le définisse comme ceci :*« Le contexte est toute information qui peut être utilisée pour caractériser la situation d'une entité. Une entité est une personne, un lieu ou un objet qui est considéré pertinente pour l'interaction entre un utilisateur et une application, y compris l'utilisateur et les applications elles-mêmes. »*

Plus récemment, s'inspirant des sciences sociales, quelques chercheurs soutiennent l'hypothèse que les contextes sont socialement et physiologiquement les résultats construits à partir de l'activité humaine plutôt que des ensembles stables.

Le contexte, définit par Schilit, inclut la localisation et l'identité des personnes et des objets à proximité ainsi que les modifications que peuvent subir ces objets [30], [34]. Étudier le contexte, c'est répondre aux questions "Où sommes-nous? ", "Avec qui ?", "Quelles sont les ressources qu'on utilise?"…etc. Il définit donc le contexte comme les changements de l'environnement physique, de l'utilisateur et des ressources dont on dispose.

Brown quant lui, restreint en 1996 le contexte aux éléments qui entourent l'utilisateur [35].Il introduit, par la suite,en 1997, l'heure, la saison, la température, l'identité ainsi que la localisation de l'utilisateur. [31]En parallèle, Ryan introduit en 1997 explicitement le temps et la notion d'état dans sa définition du contexte.Pour lui, le contexte dépend de l'environnement, de l'identité et de la localisation de l'utilisateur. [36]

En 1997, Ward voit le contexte comme les états des environnements possibles de l'application [37].

En 1998, Pascoe introduit la notion de pertinence des informations contextuelles et donne une définition du contexte comme étant un sous-ensemble d'états physiques et conceptuels ayant un intérêt pour une entité donnée [38]. Puis, Dey [39] insiste sur la notion de pertinence de l'information en proposant une définition dont le but est de spécifier la nature des entités relatives au contexte :

« Le contexte couvre toutes les informations pouvant être utilisées pour caractériser la situation d'une entité. Une entité est une personne, un lieu, ou un objet qui peut être pertinent pour l'interaction entre l'utilisateur et l'application, y compris l'utilisateur et l'application».

Cette définition résume toutes les définitions vue précédemment, du moment qu'elle est très générique. On peut expliquer cela par le fait que les paramètres contextuelles sont soit fournis par l'utilisateur lui-même ou par des capteurs déployés dans l'environnement de l'utilisateur et de l'application qui utilise le contexte. Les paramètres du contexte peuvent être déduits d'une interprétation plus ou moins complexes des données qui sont fournis par les entités citées précédemment.

En 2001, Winograd [40] reprend la définition de Dey et confirme qu'elle encapsule tous les travaux qui existe sur le contexte .Il considère néanmoins que les termes utilisés*« toute information »* et *« caractériser une entité »* sont trop généraux et ne limitent aucunement la notion de contexte. Afin de remédier à cela, Winograd définit le contexte comme un ensemble d'informations et cet ensemble est structuré et partagé ; il évolue et sert l'interprétation. Il donne plus de détails sur sa définition en disant : *« La considération d'une information comme contexte est due à la manière dont elle est utilisée et non à ses propriétés inhérentes ».*

II.2.3. Catégories de contexte

Etant donnée la diversité des informations contextuelles, il est indispensable de les classer par catégorie et ce, dans le but de simplifier leur utilisation. Nous présenterons dans cette section une classification qui résume toutes les informations du contexte habituellement utilisées par les chercheurs. Les catégories des informations contextuelles sont l'environnement, l'utilisateur et les objets. L'environnement représente des lieux de l'espace géographique comme une chambre, un bureau, une rue...etc. Les utilisateurs sont des

individus ou des groupes d'individus. Les objets, quant à eux, peuvent êtres des entités physiques, des composants logiciels ou alors des applications, des ressources, des fichiers…etc.

En prenant en compte les informations contextuelles fréquemment utilisées dans les travaux existant, nous pouvons les classer en quatre principales catégories : utilisateur, dispositif, environnement et temps.

L'environnement inclus les caractéristiques intrinsèques des éléments qui interviennent dans le système. Par exemple, dans un lieu donné, la température peut être plus ou moins élevée.

Les dispositifs sont les équipements électroniques déployés dans l'environnement. Ils peuvent être ou non disponibles et ils sont aussi évalués selon leurs performances en termes de capacité de calcul et d'énergie.

Le tempspeut également être considéré comme une catégorie de contexte car il peut définir une entité. Il permet également d'établir un historique de valeurs qui permet d'enrichir le contexte. En effet, l'enchaînement des actions dans le temps peut être utilisé dans le processus de prise de décision des applications.

Les données qui sont comprises dans ces quatre catégories peuvent être interprétées et réunies afin d'obtenir des informations contextuelles supplémentaires et ce, pour avoir une meilleure vue d'une situation. Par exemple, si nous connaissons le nombre de personne qui sont dans une pièce précise et que nous évaluons le bruit, nous pouvons déterminer si ces gens sont en réunion ou non ; de là, nous pouvons décider de déclencher ou non un service qui serait par exemple affichage d'un message personnel.

II.3. Notion de sensibilité au contexte

La notion de sensibilité au contexte caractérise la capacité d'un système à s'adapter aux variations de contexte. Dey et Abowd définissent un système sensible au contexte comme étant un système qui utilise le contexte pour fournir des informations et des services pertinents pour l'utilisateur, où la pertinence dépend de la tâche demandée par l'utilisateur [33]. Cette définition a été reprise par tous les chercheurs du domaine car elle met en lumière trois

catégories de fonctions liées à la représentation d'information, à l'exécution de services et au stockage d'informations dépendantes du contexte.

La représentation d'information concerne les applications qui se chargent de la représentation du contexte. Ces applications peuvent proposer des choix appropriés à l'utilisateur. Il y a beaucoup de travaux dans ce domaine : nous pouvons citer les travaux de Schilit en 1994[41] qui donne la liste des imprimantes à proximité de l'utilisateur.

L'exécution de services décrit les applications qui déclenchent une commande ou reconfigurent le système automatiquement en lieu et place de l'utilisateur dès qu'il y a un changement de contexte. Dans cette catégorie nous pouvons, entre autres, citer les travaux en 1994 de Bennet,le concepteur du système Teleport qui assure le transport automatique de profil utilisateur s'il change de machine. [42]

Le stockage d'informations dépendantes du contexte concerne les applications qui associent les données à leur contexte d'utilisation. Par exemple, dans une réunion, une application peut étiqueter les notes prises par l'utilisateur avec le lieu et l'instant de l'observation. [38]

Les recherches dans le domaine de la sensibilité au contexte ne sont pas encore achevées. Plusieurs éléments sont encore à étudier. Dans cette optique, Winograd a énuméré trois lacunes à corriger :

1. Il n'y a pas de définition universelle du contexte
2. Il n'y a pas de modèle définit ni de méthode de conception
3. Il n'y a pas d'outil dédié au développement et à l'hébergement d'applications sensibles au contexte.

Dans la section qui suit, nous donnons l'architecture générale d'un système sensible au contexte.

II.4. Architecture d'un système sensible au contexte

Dey fut le premier en 1999[39] à faire la séparation entre l'acquisition de contexte et son utilisation dans les applications. Du moment que les problèmes auxquels nous sommes confronté dans les applications sensibles au contexte sont les mêmes que ceux qu'on rencontre lors du développement des systèmes interactifs mobiles, Dey a trouvé qu'il serait intéressant de séparer la gestion du contexte de l'application elle-même, et ce, dans le but de

pouvoir développer une plateforme générique de développement et de déploiement d'applications sensibles au contexte. Selon lui, la seule chose qui sépare les applications sensible au contexte d'un système interactif mobile est que ce dernier manipule des données explicites qui sont soit des variables internes de l'application soit des entrées explicites de l'utilisateur. De là, il propose une plateforme semblable aux systèmes interactifs mais il généralise les types d'entrées afin de prendre en compte les données du contexte. Il a défini quelques abstractions qui aident à inférer une interprétation de haut niveau du contexteet qui supporte la séparation entre la gestion du contexte et les applications. Plusieurs travaux dans le domaine de la sensibilité au contexte ont repris cette idée et proposent des architectures en couches. Ces architectures diffèrent dans les noms, les fonctions et le placement des couches.Nous remarquons néanmoins que toutes les propositions sont basées sur cinq couches principales qui sont : capture du contexte, interprétation/agrégation du contexte, stockage/historique du contexte et enfin la couche application. La figure 4 est une représentation de l'architecture d'un système sensible au contexte.

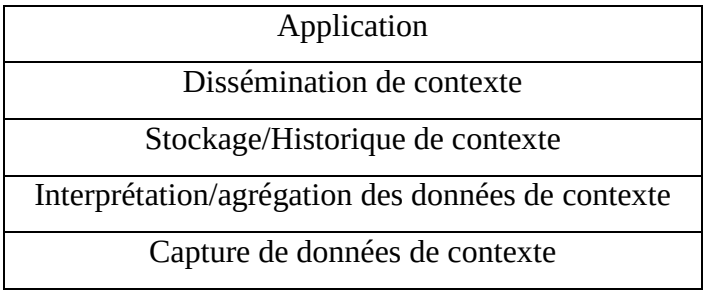

Figure 4 : Architecture générale d'un système sensible au contexte.

Dans ce qui suit, nous détaillerons les fonctions et les différents éléments qui composent chacune de ces couches.

II.4.1. Couche de capture du contexte

Cette couche est composée de capteurs. Un capteur est un composant matériel ou logiciel qui peut générer une information contextuelle. Nous distinguons trois types de capteurs : physiques, virtuels et logiques [43].

II.4.1.1. Capteurs physiques

Les capteurs physiques sont des dispositifs matériels qui sont capables de fournir des données de contexte. Le tableau suivant donne quelques exemples de capteurs physiques selon le type d'information qu'ils fournissent.

Type d'information fournie	Capteurs disponibles
Lumière	Photodiodes, capteurs de couleurs, capteurs d'infrarouge et d'ultra-violet…
Contexte visuel	Caméras numériques
Audio	Microphones
Localisation	GPS (Global Positionning System), GSM (Global System for Mobile Communications),
Température	Thermomètres numériques

Tableau 1: Exemples de capteurs physiques.

II.4.1.2. Capteurs virtuels

Les capteurs virtuels fournissent des informations contextuelles à partir d'applications ou services logiciels. Par exemple, il est possible de détecter l'emplacement d'un livreur de marchandises en consultant son carnet électronique de rendez-vous sans avoir recours à des capteurs physiques. Nous pouvons aussi détecter l'activité de l'utilisateur sur un microordinateur en analysant les évènements de la souris ou les saisies à partir du clavier. Les capteurs virtuels sont beaucoup moins coûteux que les capteurs physiques puisqu'ils sont basés sur des composants logiciels qui sont généralement moins chers que des appareils électroniques.

II.4.1.3. Capteurs logiques

Ce type de capteurs utilise généralement plusieurs sources d'information contextuelles pour fournir une autre information de synthèse plus précise. Ces capteurs peuvent réutiliser des capteurs physiques et virtuels pour fournir un contexte de plus haut niveau. Par exemple, un capteur logique peut fournir l'emplacement d'un caissier dans un grand magasin en analysant les différentes sessions ouvertes sur les caisses du magasin. Dans certain travaux, les capteurs logiques sont considérés comme des interpréteurs de contexte. [44]

II.4.2. Couche d'interprétation et agrégation du contexte

Cette couche offre le moyen d'interpréter les données contextuelles que fournissent les capteurs de contexte. Elle permet d'analyser et de transformer des données brutes fournies par la couche de capture de contexte en format haut niveau plus facile à manipuler et à utiliser. En effet, les capteurs fournissent généralement des données techniques qui ne sont pas appropriées pour une utilisation directe par l'application. Les transformations effectuées sur les données brutes fournies par les capteurs peuvent être réalisées par plusieurs opérations : extraction, quantification, raisonnement, agrégation... Par exemple, les coordonnées GPS d'une personne peuvent être moins significatives qu'une adresse physique sous forme de numéro de rue et de ville. La complexité des interprétations de contexte peut varier d'une simple agrégation de valeurs qui proviennent de plusieurs capteurs à des raisonnements ou analyses statistiques complexes. Par exemple, la localisation de plusieurs personnes dans une seule salle peut inférer le fait qu'ils sont en réunion. Dans ce cas, le niveau de bruit peut aussi être une information importante pour savoir s'ils sont en réunion de travail ou de loisir.

Cette couche doit aussi assurer la résolution de conflits causés par l'utilisation de plusieurs sources de contexte. En effet, ces sources peuvent donner des résultats contradictoires ou peuvent aboutir à des situations de confusion. Cette couche doit donc avoir une certaine intelligence d'interprétation pour résoudre ces conflits [45], [46].

II.4.3. Couche de stockage et historique du contexte

La troisième couche "stockage et historique du contexte" organise les données capturées et interprétées et les stocke pour une utilisation ultérieure. Ce stockage peut être centralisé ou distribué. La solution centralisée est l'option la plus répandue et la plus utilisée puisqu'elle facilite la gestion des mises à jour et des variations des valeurs du contexte. La gestion distribuée du contexte est beaucoup plus complexe puisqu'elle inflige des fonctions additionnelles de découvertes de ressources et d'actualisation des valeurs du contexte. De plus, cette gestion distribuée alourdit la tâche de l'application qui doit gérer la collecte des différentes informations contextuelles d'une façon interne.

Pour stocker une information, nous avons besoin de définir un modèle pour la décrire. Ainsi, un modèle de contexte est nécessaire pour pouvoir l'utiliser dans l'application. Strang et Linnhoff-Popien [47] ont résumé les approches de modélisation de contexte les plus intéressantes de la littérature. Ils ont proposé une classification des approches de modélisation

basée sur la structure de données utilisée pour la description et l'échange du contexte. Nous présentons ces approches en trois catégories : l'approche Attribut/Valeur, les approches orientées modèles, les approches basées sur XML et les approches basées sur les ontologies.

II.4.3.1. L'approche attribut-valeur

Plusieurs architectures modélisent le contexte sous forme de paires (attribut, valeur) où l'attribut est l'identifiant de l'information contextuelle et la valeur est sa valeur actuelle. L'avantage de cette méthode est la facilité d'implémentation. En effet, la gestion du contexte revient à parcourir la liste des contextes disponibles. Cependant, ce modèle manque d'expressivité et de complétude. Sa structure trop lisse ne permet pas de définir tous les aspects contextuels d'une application. Ce modèle prête aussi souvent à confusion car il y a des lacunes dans la définition. Par exemple, on prend les deux contextes num1 et num2 définies comme suit : (Name="num1", User="touriste01", Localisation="Hôtel le Tais", Time="July 15 2012") le contexte num1 est défini par « l'utilisateur x est localisé dans un emplacement y à temps t » si on prend (Name="num2", User="x", Localisation="z", Time="t") avec l'emplacement z est dans l'hôtel Tais (par exemple la chambre 20 de l'hôtel), on ne peut pas dire que num2 est un sous contexte de context1 et que toutes les fonctionnalités offertes par l'application dans num1 doivent aussi exister dans num2.

II.4.3.2. Les approches orientées modèle

L'approche orientée modèle utilise des modèles formels pour modéliser les informations de contexte. Son objectif principal est d'offrir la possibilité d'encapsuler le contexte et permettre sa réutilisation. On trouve Unified Modeling Language (UML) : Sheng et Benatallah ont proposé un méta-modèle basé sur une extension d'UML qui permet de modéliser le contexte auquel des services web sont sensibles.

II.4.3.3. Les approches basées sur XML

Les modèles basés sur XML utilisent une structure de données hiérarchique. La profondeur de cette structure dépend du contexte décrit. Les langages à base de profils sont largement utilisés dans cette catégorie de modèles. Par exemple, nous pouvons citer le profil CC/PP (Composite Capabilities / Preference Profile) [19] et le profil UAProf (User Agent Profile) [49].

II.4.3.4. Les approches basées sur les ontologies

Les ontologies représentent une autre solution pour modéliser le contexte [50].
[50] justifie l'utilisation des ontologies par trois arguments:

- Une ontologie permet de partager les connaissances dans un système distribué
- Une ontologie comprend des sémantiques déclaratives permettant d'élaborer des raisonnements sur les informations contextuelles
- Avec une représentation explicite d'une ontologie commune, l'interopérabilité des applications et des terminaux est assurée.

II.4.4. Couche dissémination du contexte

Cette couche est chargée de transmettre les informations contextuelles à l'application. Elles peuvent provenir de plusieurs types de dispositifs déployés sur une grande zone géographique.C'est pour cette raison qu'il y a une couche spécialisée dans la transmission de ces données aux applications. Elle assure une parfaite transparence de la communication avec les applications ; de ce fait, l'implémentation des applications devient simple car elle masque l'hétérogénéité des protocoles de communications des différents dispositifs qui fournissent le contexte.

Cette couche offre également des moyens de communication standards pour notifier les applications des éventuels changements de contexte.

II.4.5. Couche application

Cette couche est en fait l'application offrant les différents services requis par les clients. Elle se charge de récupérer les informations de contexte auxquelles l'application est sensibleet implémente les différents mécanismes à actionner lors d'un changement de contexte. L'application est connectée à la couche de dissémination du contexte qui lui permet d'accéder aux informations de contexte et être notifiée des éventuelles mises à jour sur ces dernières. Une application peut accéder à ces informations de deux manières : synchrone et asynchrone.

- Accès synchrone : dans ce cas, c'est l'application qui demande à la couche de dissémination de lui fournir l'information contextuelle dont elle a besoin. Cela se fait généralement par un envoi de requête à la couche dissémination.

- Accès asynchrone : L'application sélectionne les évènements auxquelles elle choisit de s'abonner et ces évènements correspondent à des valeurs de contexte. Si un de ces évènements est actionné, l'application sera notifiée.

II.5. Les principales plateformes de gestion de contexte

Cette section est consacrée à la présentation de quelques middlewares de gestion de contexte. Nous analysons les approches abordées par chaque plateforme pour l'implémentation des couches principales d'un système sensible au contexte.

II.5.1. CAMidO

CAMidO [51] (Context-Aware Middleware based on Ontology meta-model) est comme son nom l'indique, un middlewarefondé sur un méta-modèle, lui-même basé sur les ontologies pour décrire le contexte. En effet, sa description du contexte permet la collection des informations contextuelles et la détection des changements dit pertinents sur celles-ci. Elle permet également d'automatiser les processus d'adaptation au contexte. Dans ce modèle, c'est au concepteur de décrire le contexte et les différentes politiques d'adaptation que CAMidO doit prendre en compte lors de l'automatisation du processus de sensibilité au contexte. Il prend en compte deux types d'adaptation : l'adaptation réactive et l'adaptation proactive.

L'architecture de CAMidO est constitué de trois couches : les capteurs, la couche middleware et la couche application.

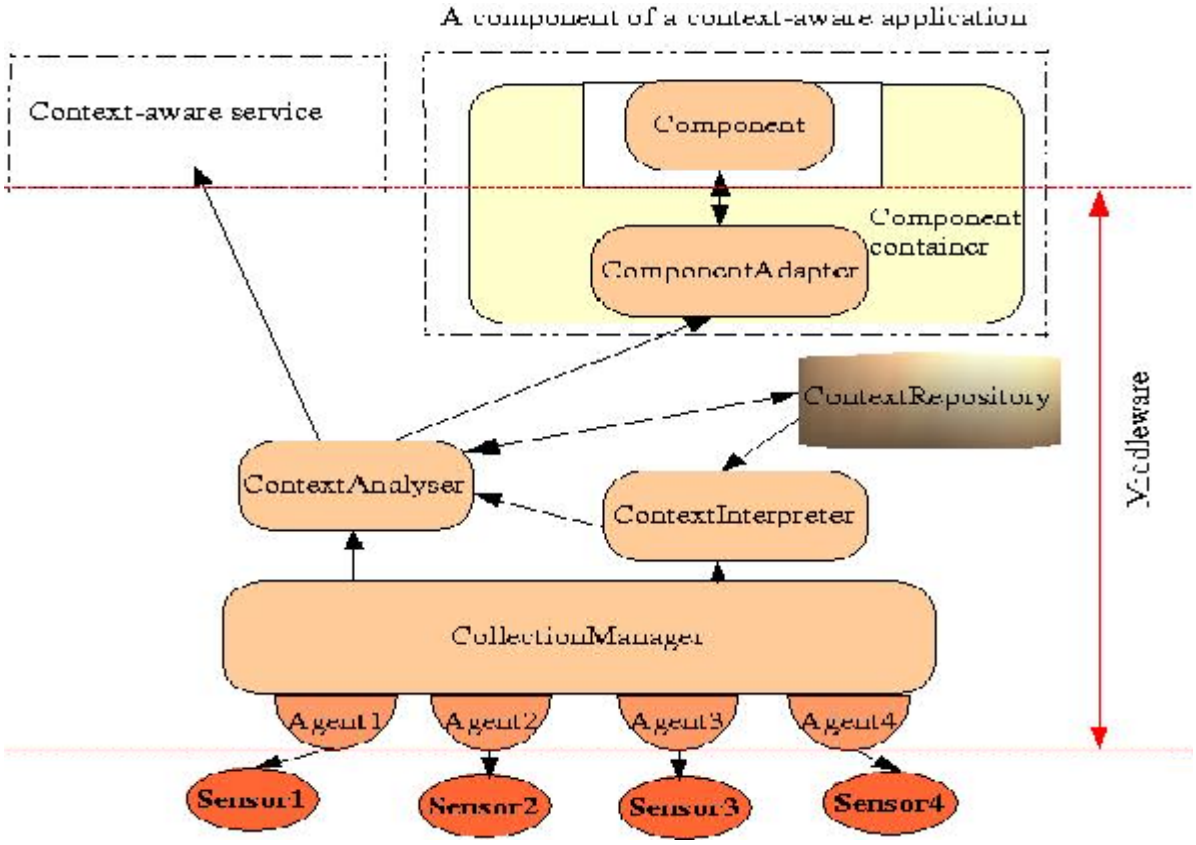

Figure 5 : Architecture de CAMidO.

- CollectionManager : Chargé de la collecte des informations contextuelles provenant de capteurs. On associe un agent à chaque capteur. Le rôle de ces agents et d'interagir avec leur capteur. Les données collectées sont ensuite transférées au ContextAnalyser et au ContextInterpreter.
- ContextAnalyser : son rôle est d'analyser les informations contextuelles et de détecter les changements pertinents. L'analyse es faite en comparant les valeurs des informations contextuelle collectées avec celles qui sont stockées dans ContextRepository, s'il détecte de nouvelles information contextuelles pertinentes, elles seront stockées dans ContextRepository.
- ContextInterpreter : il déduit les données contextuelles formalisées à partir des informations brutes.
- ContextRepository : pour le stockage des informations du contexte.
- ComponentAdapter : déclare au ContextAnalyser les changements de contexte qui le concernent. Si un contexte pertinent est détecté, le contextAnalyser notifie le Componentadapter qui va exécuter le code approprié.

II.5.2. Context Toolkit

Context Toolkit [52] facilite le développement et le déploiement d'applications sensibles au contexte.il est responsable de l'acquisition, l'interprétation et la fourniture d'information contextuelles aux applications.

Comme le montre la figure 6, le composant principal de l'architecture est le «context widget». Deux autres types de composants complètent l'architecture : les « interpréteurs » et les « serveurs ».

- Un « context widget », ou widget de contexte, est un composant autonome qui encapsule un capteur physique. Son rôle est de communiquer les informations perçues à un (ou plusieurs) serveur(s) ou interpréteur(s) en fonction des données reçues par le capteur.
- Les interpréteurs ont pour objectif de donner un sens aux signaux qui leur sont transmis par les « context-widgets ».
- Les serveurs ont pour objectif principal la collecte des signaux émis par les autres composants et de faire le lien entre les applications et les « context-widgets ».

Dans le Context Toolkit, nous retrouvons les différentes couches du modèle générique présenté dans la figure 6 : Les context-widgets permettent un accès simple au capteur (couche capture de contexte). Les interpréteurs donnent un sens au contexte capturé (couche interprétation du contexte), et les serveurs (couche stockage et historique de contexte et aussi couche dissémination) font le lien entre les context widgets et les applications sensibles au contexte. Cependant, les context-widgets ne sont que de simples pilotes logiciels pour les capteurs auxquels ils sont liés. Ils ne fournissent pas de sens aux informations qu'ils transmettent et ont besoin pour cela des interpréteurs. Les serveurs, quant à eux, regroupent le gros du travail. Ils doivent synthétiser de nouvelles informations avec celles que leur fournissent les context-widgets et faire le lien entre les applications et les context-widgets.

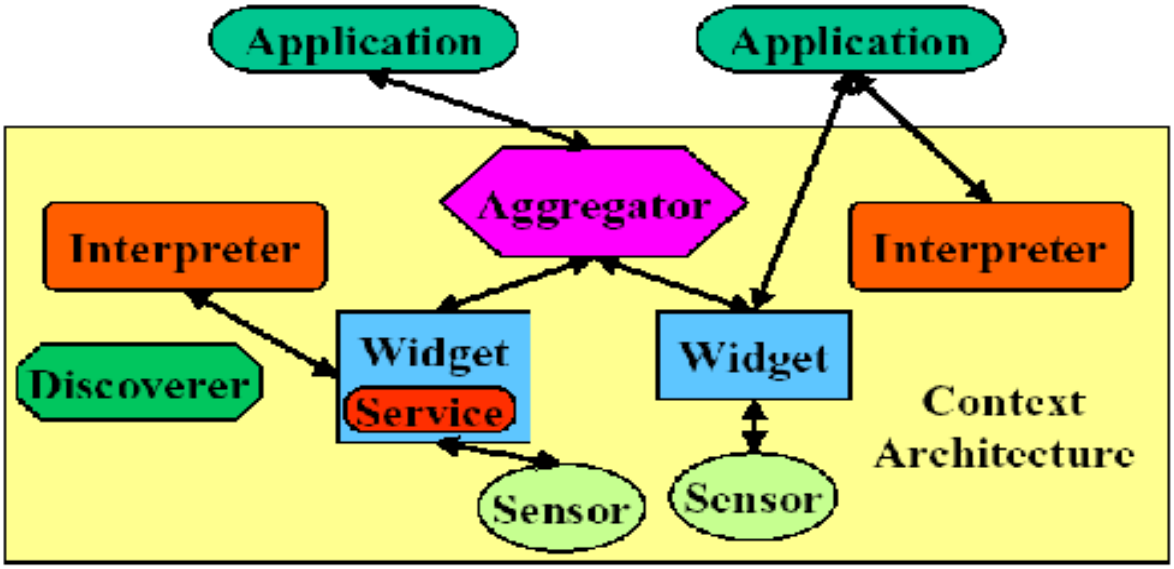

Figure 6 : Architecture du middleware Context Toolkit de Anid K.Dey.

II.5.3. Gaia

Gaia [53] est un middleware pour la gestion des espaces intelligents. Il permet d'exécuter plusieurs actions (contrôler la lumière, ouvrir ou fermer les portes automatiquement…etc.) [53].Dans l'environnement intelligent Gaia, les utilisateurs peuvent interagir avec les équipements disponibles et même intégrer leurs propres équipements comme des ressources additionnelles à l'environnement. Gaia offre également des mécanismes d'adaptation aux changements de l'environnement ubiquitaire.

Gaia s'appuie sur un modèle de contexte basé sur des prédicats du type : Contexte (TypeContexte, Sujet, Prédicat, Objet) où le paramètre TypeContexte désigne le type de l'attribut contextuel, Sujet l'entité concernée (une personne, un emplacement ou un objet) par l'attribut contextuel et Objet une valeur associée au Sujet. Les prédicats sont écrits en langage de marquage d'ontologie DAML. Ils sont utilisés par les agents fournisseurs pour l'acquisition de connaissances contextuelles et par les agents Synthétiseurs à travers une base de règles pour déduire par inférence de nouvelles connaissances contextuelles. Pour la résolution de conflit entre règles, la règle d'arbitrage utilisée consiste à associer à chaque règle une priorité.

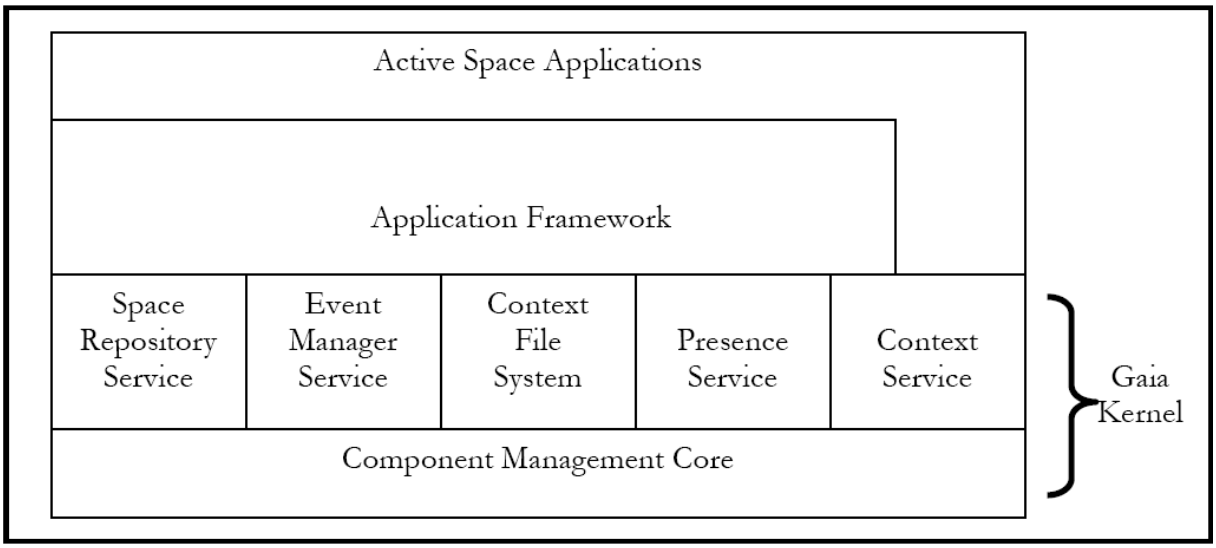

Figure 7 : Architecture du middleware GAIA.

II.5.4. SOCAM

SOCAM (Service Oriented Context-Aware Middleware) [54] est basé sur le développement et le prototypage rapide de services sensibles au contexte dans des environnements intelligents. C'est un middleware distribué qui convertit les divers espaces physiques d'où le contexte est capturé en un espace sémantique où le contexte peut être partagé et fourni aux services sensibles au contexte. Ce middleware se base sur des ontologies pour modéliser le contexte.

L'architecture de SOCAM, présentée dans la figure 5, comprend les composants suivants :

- context providers (fournisseurs de contexte) : leur rôle est de capturer les informations contextuelles utiles à partir de dispositifs hétérogènes de l'environnement de l'utilisateur et de l'application. ils les convertissent par la suite en des représentations OWL afin que le contexte puisse être partagé et réutilisé par d'autres composants de l'architecture.
- context interpreter (interprète de contexte) : fournit des services de raisonnement logique sur les représentations OWL du contexte. Il fournit aussi un service d'interrogation intelligent qui permet de résoudre les conflits d'interprétation du contexte.

- context database (base de données de contexte) : Elle stocke les différents éléments de l'ontologie qui décrit l'environnement de l'application et les instances des ontologies spécifiques au domaine de l'application décrivant l'environnement de l'utilisateur.
- context-aware services (services sensibles au contexte) : ils utilisent les différentes informations stockées dans context database pour modifier leur comportement selon le contexte courant.
- service-locating service (service de localisation de services) : il fournit un mécanisme avec lequel des utilisateurs ou des applications peuvent localiser les fournisseurs et les interpréteurs de contexte.

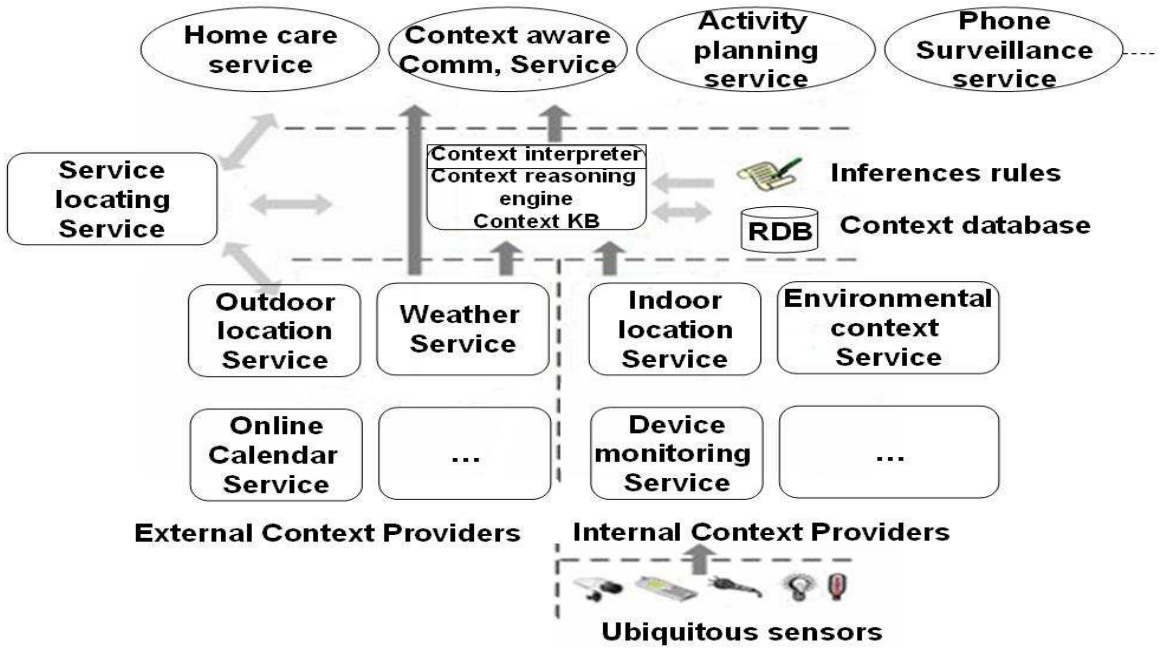

Figure 8 : Architecture de la plateforme SOCAM.

II.5.5. CoBrA

L'architecture de CoBrA (Context Broker Architecture) [55], [56] est basé sur système multi-agents avec un agent central « Context Broker Agent ». Cet agent est un serveur central de contexte qui conserve une base de connaissance partagé par tous les agents.

Les agents peuvent être des applications qui s'exécutent localement, des services offerts par des équipements de l'environnement mobile...etc. Le rôle de l'agent CoBra consiste en :

- l'acquisition du contexte depuis les différentes sources de l'environnement via les différents capteurs et l'inférence de nouvelles données de contexte ;

- la vérification de la cohérence de la base de connaissances du contexte et protection de la vie privée des utilisateurs en mettant en place des mécanismes d'accès et de sécurité.

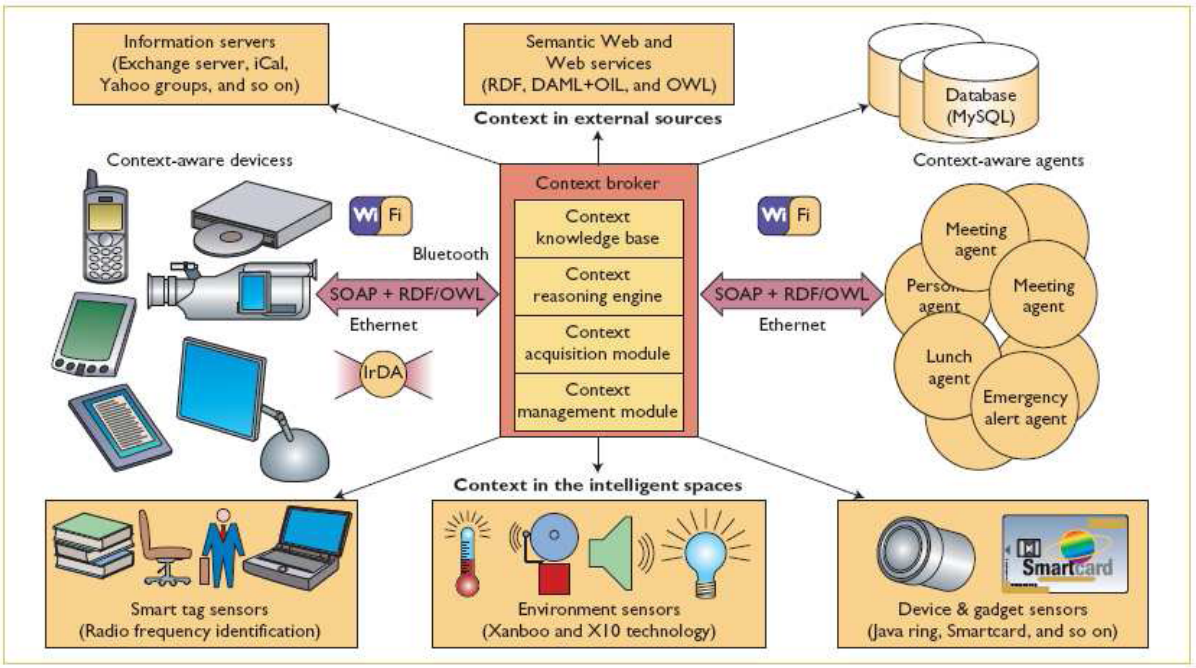

Figure 9 : Architecture d'un espace actif basé sur l'agent CoBra.

L'agent Broker est composé de trois modules [50] :

1- Une base de connaissances : contient les différentes connaissances contextuelles acquises.
2- Un moteur d'inférence : permet à l'agent CoBra de déduire le contexte courant de l'environnement actif et de détecter les incohérences de la base de connaissances contextuelles.
3- Un gestionnaire de politiques associées au contexte : permet à l'agent CoBrA d'analyser les politiques de gestion du contexte définies par les utilisateurs et de décider des actions correspondantes.

CoBra modélise le contexte grâce aux ontologies ; néanmoins, le fait qu'il soit centralisé le rend non extensible.

II.5.6. MyCampus

MyCampus [56] a été développé pour offrir des services sensibles au contexte permettent d'assister des utilisateurs dans leurs activités quotidiennes sur le campus de l'université de

Carnegie mellon (CMU).Il offre entre autres le service de recommandation de restaurants dans le campus, d'organisation de réunion entres les utilisateurs,etc. Il est basé sur une architecture multi-agents qui s'articule autour d'un agent central « e-Wallet ». Ce dernier assure les taches suivantes :

- l'acquisition et le traitement des connaissances contextuelles.
- le respect des règles de la vie privée de l'utilisateur.

Un agent e-Wallet gère le contexte de plusieurs utilisateurs de l'environnement MyCampus.

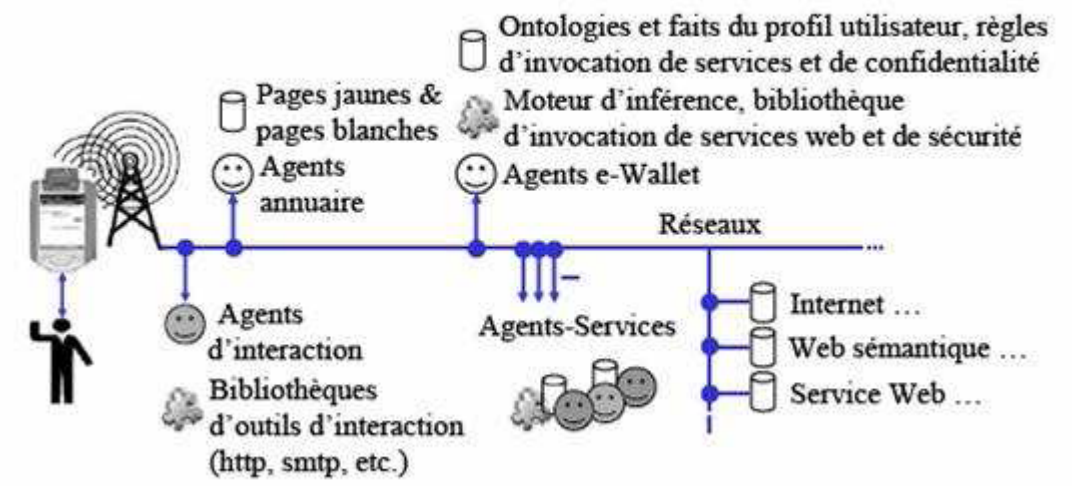

Figure 10 : Architecture de MyCampus.

L'agent e-Wallet est structuré en trois couches :

- la couche noyau maintient un modèle des connaissances contextuelles concernant l'environnement et l'utilisateur.
- la couche de services inclut les règles qui décrivent la correspondance entre les types de connaissances supportées par la couche noyau et les services qui permettent de fournir ces connaissances.
- la couche confidentialité encapsule les règles de contrôle d'accès et les règles de révision des connaissances contextuelles. Ces règles ont pour rôle de vérifier le contenu des connaissances qui seront fournies par l'e-wallet aux autres agents.

Mycampus contient d'autres agents :

- Agent d'interaction : son rôle est d'interagir avec les utilisateurs ;
- Agent annuaire : permet aux utilisateurs la recherche des services ;
- Agent de services : son rôle est de fournir des services.

II.6. Conclusion

Dans ce chapitre, nous avons vu qu'ils y a plusieurs définitions de la notion de contexte selon l'application et l'environnement de déploiement. Apres avoir présenté les principales définitions de contexte, nous avons ensuite explicité la notion de sensibilité au contexte. Nous avons aussi présenté l'architecture générale d'un middleware de sensibilité au contexte utilisé par la plupart des chercheurs dans le domaine. Par la suite, nous avons présenté quelques middlewares de sensibilité au contexte. Dans ces plateformes, il y a une séparation de l'acquisition du contexte d'une part et de son utilisation d'autre part. En effet, il est très important d'encapsuler les mécanismes d'acquisition du contexte dans des composants qui offrent une interface de communication standard. Chacune de ces plateformes propose son modèle d'acquisition et de collecte du contexte.

Le tableau suivant synthétise les aspects principaux de chaque architecture des plateformes étudiées : Context Toolkit, CoBrA, Gaia, MyCampus, CAMidO et SOCAM.

	Context Toolkit	**CoBrA**	**Gaia**	**CAMidO**	**SOCAM**	**MyCampus**
Technique utilisée	basée sur des widgets	basée sur des agents	Serveur centralisé (Agents)	Middleware distribué	middleware distribué	Serveur centralisé (Agents)
méthode de capture	widget de contexte	module d'acquisition	Agent provider	Capteurs	fournisseurs de contexte	Agent eWallet
modèle de contexte	attribut, valeur	ontologies (OWL)	ontologies (DAML)	ontologies	ontologies (OWL)	Ontologies (OWL)
interprétation du contexte	transformation et agrégation	moteur d'inférence et une base de connaissances	Non disponible	Moteur d'inférence	Moteur d'inférence	Agent eWallet
Historique et stockage du contexte	Dans un serveur	Distribué	Oui	Disponible dans le répertoire de contexte	disponible dans une base de données	Non disponible

Tableau 2 : Tableau comparatif entre les différentes plateformes de sensibilité au contexte.

Comme le montre le tableau 2, la plupart des middlewares utilisent les ontologies pour la modélisation du contexte car cela offre une grande richesse sémantique et la possibilité d'interprétation et d'inférence du contexte en comparaison aux autres modélisations du contexte.

Un autre aspect important dans les systèmes sensibles au contexte, c'est la gestion de l'historique du contexte. Il offre la possibilité d'implémenter des algorithmes d'apprentissage et ce, pour que les services soient adaptables au contexte,ce qui permet d'instaurer une semi-automatisation dans le lancement de certains services sans qu'une intervention de l'utilisateur soit nécessaire. L'historique peut également être utilisé pour prédire les valeurs futures du contexte. Une information contextuelle même si elle n'est pas utilisé sur le moment pourra être utilisé par la suite comme information de l'historique du contexte.

Finalement, nous constatons que certains des middlewares n'implémentent pas de mécanisme de découverte de nouvelle source d'informations contextuelles. Il est pourtant indispensable de pouvoir découvrir de nouvelles sources d'information de contexte et avoir également un moyen d'y accéder dans un environnement ubiquitaire.

Les middlewares de sensibilité au contexte ont pour tâche de visualiser la situation du point de vue de l'utilisateur par rapport à son environnement et, ainsi, d'adapter les services qu'ils lui offrent par rapport à cette situation. De ce fait, ils permettent de simplifier le processus de développement d'applications sensibles au contexte pour les environnements ubiquitaires. Ces applications sont en général une composition de plusieurs services que fournissent les dispositifs présents dans l'environnement. Pour réaliser cette composition, il faut avoir des outils qui permettent la composition de service et ce, afin de répondre au mieux aux requêtes émises par l'utilisateur qui sont pour la plupart composite. Dans le prochain chapitre, nous détaillerons la notion de composition de services.

Chapitre III : Composition de services.

III.1. Introduction

Un environnement ubiquitaire est composé de plusieurs dispositifs offrant des services aux utilisateurs ou à d'autres applications logicielles. Le fait que ces environnements soient mobiles fait de la découverte et de la composition des services atomiques une problématique importante.

Dans l'optique d'offrir un service personnalisé, il faut que les différents services soient intégrés les uns avec les autres et ce, pour enrichir l'offre et la rendre plus intéressante aussi bien pour les applications que pour les utilisateurs. En effet, les requêtes des utilisateurs sont souvent complexes.Cela fait qu'un service seul peut rarement répondre entièrement à une requête.C'est pour cela qu'il faut composer ces services qui sont, à l'origine, des services atomiques pour répondre à la requête tout en prenant en compte les informations contextuelles. Il est important d'intégrer ce paramètre au processus de composition de services car il permet d'adapter au mieux cette composition aux besoins réels de l'utilisateur.

La composition de services offre aux utilisateurs la possibilité d'avoir des réponses satisfaisantes à leurs requêtes complexes. Afin de rendre un service « composable », on utilise sa description.On se sert en général des langages du web sémantique comme OWL-s pour décrire les services. Le processus de composition va ensuiteadopter ces descriptions et proposer un plan de composition en suivant le modèle de composition qui a été choisi.

La plupart des approches de compositions qui ont été développées pour les systèmes ubiquitaires sont inspirées des approches de compositions des services web.Dans ce chapitre, nous allons d'abord présenter les méthodes de composition des services web qui existent et quelques-unes des approches développées. L'inconvénient de ces approches est leur manque de dynamique. Pour pouvoir les adapter et les rendre compatibles avec des environnements aussi mobiles que les environnements ubiquitaires, il fallait les décentraliser et ajouter de nouvelles contraintes comme la sensibilité au contexte. Puis, nous allons présenter quelques approches développées dans le domaine de l'informatique ubiquitaire.

III.2. Définition de la composition de services

La composition de service consiste à assembler des services pour obtenir un résultat précis, et ce, en utilisant des primitives de contrôle (boucles, tests, traitement des exceptions…etc.) et en échangeant des messages entre les services. Dans les services web, la composition précise les services qui ont besoin d'être invoqués, l'ordre de leurs invocations et la façon dont on peut gérer les exceptions. Le processus de composition inclut la découverte

et la sélection automatique des services tout en garantissant la compatibilité des données. Les techniques de composition de services sont utilisées dans divers domaines comme le e-commerce, les Grid de calcul ou simplement en utilisant le web.

Malgré les efforts déployés, la composition de services reste une problématique de recherche très complexe. Cette complexité est due au fait que les solutions proposées pour la composition de services doivent tenir compte du fait que ces services qui sont déployés dans un environnement sont de plus en plus nombreux et ils sont constamment mis à jour. Du moment que les environnements peuvent être modifiés, le système chargé de la composition doit être en mesure de détecter et de s'adapter à ces changements.

Un autre défi pour la composition, c'est l'hétérogénéité des services.En effet, chaque service est créé par une organisation différente et ces organisations n'ont pas toujours les mêmes modèles de conceptions pour décrire leurs services. Par exemple, deux services qui ont la même fonctionnalité peuvent être décrits de deux façons différentes. [58]

III.3. Architecture générale d'un modèle de composition de services

L'architecture générale d'un modèle de composition de services est présentée dans la figure 11[59]. Cette architecture est une abstraction de haut niveau de la méthode de composition de serviceset cela, sans prendre en compte les détails de l'approche comme la plateforme ou le langage de description utilisée. En réalité, cette architecture a pour objectif de pouvoir comparer et visualiser les différences entre les différentes approches de compositions qui existent.

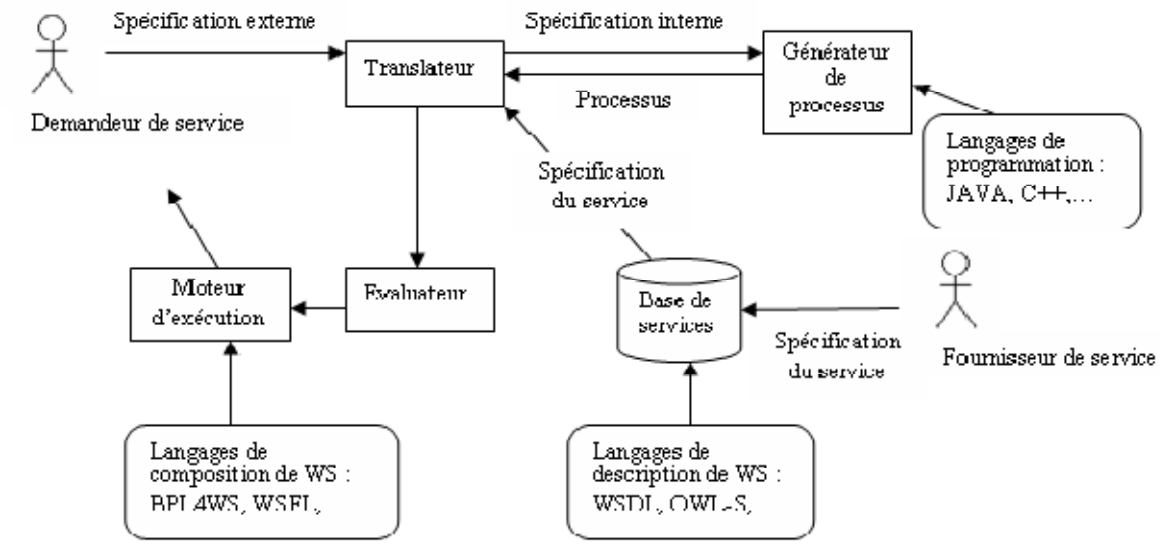

Figure 11 : Architecture générale d'un modèle de composition de web services.

Le système de composition a deux types de participants : le fournisseur de service qui propose les services pour l'utilisation et le demandeur de services qui utilise les services fournis par le fournisseur. Le système est constitué aussi des composants suivants : un translateur, un générateur de processus, un évaluateur, un moteur d'exécution et une base de services.

Chaque composant est utilisé lors du processus de composition. Ce dernier inclut les phases suivantes :

- **Présentation d'un service** :

Chaque fournisseur publie la description des services qu'il propose. Pour cela, ils peuvent utiliser plusieurs langages: UDDI, DAML-S, etc. Les principaux attributs qu'on utilise pour décrire un service web sont : la signature, les états et les paramètres non fonctionnels. La signature est constituée des entrées, des sorties et des exceptions pour un service. Les états sont les *préconditions* et les effets. Les paramètres non fonctionnels sont des données qu'on peut utiliser pour évaluer un service (le coût, le temps de réponse…).

- **Translation de langages :**

Un système de composition distingue les langages de spécification interne et externe des services. Tout ce qui est externe représente le langage que va utiliser un utilisateur pour formuler sa requête. Ces langages sont évidemment différents des langages internes que va utiliser le processus de composition car ceux-là sont plus formelles (les langages de programmation logiques…).C'est pour cela qu'il faut des composants de translation pour lier les langages internes et externes.

- **Génération de modèle de processus de composition**

Le client qui demande un service peut formuler sa requête avec un langage de spécifications de services. Le rôle du générateur de processus est de satisfaire cette requête en composant des services atomiques qui ont été préalablement publiés par des fournisseurs de services. Il prend en entrée les paramètres fonctionnels du service et fournit en sortie le modèle de processus qui décrit le service composite. Ce modèle contient plusieurs services atomiques reliés par des flots de données de contrôles.

- **Évaluation de service composite**

On peut avoir plusieurs services offrants les mêmes fonctionnalités. De là, le générateur de processus peut produire plusieurs services composites pouvant répondre à la requête de l'utilisateur. Ainsi, nous évaluerons ces services composites en comparant leurs paramètres non fonctionnels. En général, on utilise les fonctions d'utilité pour évaluer un service. Le

client devra alors spécifier le poids de chacun des paramètres non fonctionnels pour pouvoir classer les services et le meilleur service sera celui que se classera en première position.

- **Exécution de service composite :**

Une fois que le processus de composition est élaboré, on pourra exécuter le service composite. Cette exécution peut être vue comme une séquence de messages émis en suivant le modèle de processus. Le flux de données de ce service composite est en fait des actionsdont les données de sorties d'un service exécuté seront passées en entrées pour un service atomique qu'on pourra exécuter par la suite.

III.4. Classification des méthodes de composition de services

Il est possible de classifier les approches de composition selon deux critères :

- en fonction du degré d'interaction avec l'utilisateur : manuelle, semi-automatique ou automatique ;
- en fonction du mode de définition des schémas de composition : statique ou dynamique.

III.4.1. Selon le degré d'interaction avec l'utilisateur

On peut classer les approches de composition en trois catégories :

III.4.1.1. La composition manuelle

La composition manuelle des services Web suppose que l'utilisateur génère la composition à la main via un éditeur de textes et sans l'aide d'outils dédiés. Un exemple d'approche de composition manuelle est Triana proposée par Taylor et al. en 2003 [78].

III.4.1.2. La composition semi-automatique

Les techniques de composition semi-automatiques sont un pas en avant en comparaison avec la composition manuelle dans le sens où ils font des suggestions sémantiques pour aider à la sélection des services Web dans le processus de composition. Un exemple de ce type d'approche est Stevens et al. en 2003 [79].

III.4.1.3. La composition automatique

La composition totalement automatisée prend en charge tout le processus de composition et le réalise automatiquement sans qu'aucune intervention de l'utilisateur ne soit requise.Un exemple d'une approche de composition automatique est le protocole de coordination d'agents introspectifs pour la chorégraphie dynamique de services [75].

III.4.2. Selon le mode de définition des schémas de composition

Les techniques de composition de services Web peuvent être classées en deux grandes catégories : les techniques de composition statiques et les techniques de composition dynamiques.

III.4.2.1. La composition statique

Les techniques de composition statiques sont définies à l'aide de processus métier ; dans ce cas, le service composite est défini par un ensemble de services atomiques et par la façon dont ils communiquent entre eux.

Dans les méthodes de composition statiques, les services à composer sont présélectionnés et le flot de contrôle préalablement spécifié.

L'orchestration et la chorégraphie sont des compositions statiques de services web permettant de créer ce processus métier (workflow).

L'orchestration décrit, du point de vue d'un service Web, les interactions de celui-ci ainsi que les étapes internes (par exemple, transformations de données, invocations à des modules internes) entre ses interactions.

L'orchestration explique la manière avec laquelle les services Web peuvent interagir entre eux,selon un scénario prédéfini au niveau des messages dans une vision opérationnelle, avec des structures de contrôle, incluant la logique métier et l'ordre d'exécution des interactions. Un service Web prend le contrôle du déroulement de la composition et coordonne, donc, les différentes opérations des différents services Web.

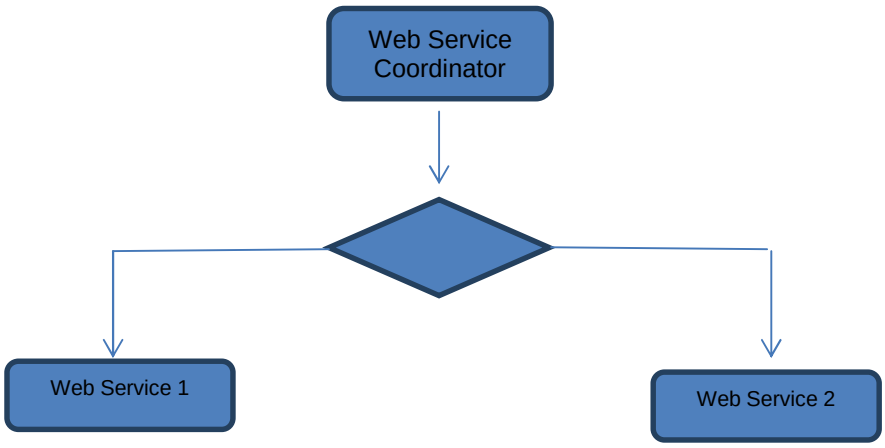

Figure 12 : Web Service Orchestration.

- **La chorégraphie**

La chorégraphie de services Web, est une description des interactions visibles entre un ensemble de services Web. Elle décrit les différents messages qui transitent entre les différents acteurs des services Web. Chacun des services intervenant dans la composition sait exactement ce qu'il doit faire, quand il doit le faire et avec qui. Donc, ils ont tous une connaissance plus ou moins globale de l'environnement dans lequel ils se retrouvent.

L'orchestration diffère de la chorégraphiecar elle décrit un ensemble des services Web contrôlé par un seul service Webalors que la chorégraphie décrit un modèle plus collaboratif, engendrant un échange de messages entre plusieurs services Web, sans qu'aucune de ces services Web ne contrôle cet échange.

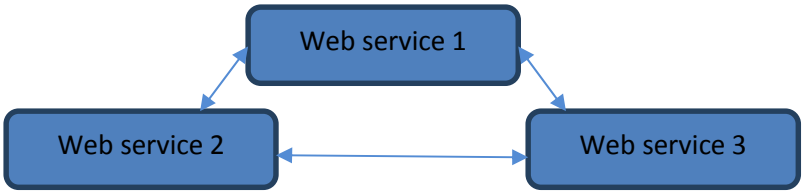

Figure 13 : Chorégraphie de Web services.

Une différence importante entre l'orchestration et la chorégraphie est que l'orchestration offre une vision centralisée, c'est-à-dire que le procédé est toujours sous le contrôle de la perspective d'un des partenaires. En revanche, la chorégraphie offre une vision globale et plus

collaborative de la composition des services Web. Elle décrit le rôle que joue chaque service Web impliqué dans l'application.

- **Limites des approches statiques**

Ces méthodes sont basées sur le fait que les schémas de compositions sont définis d'avance.Si un des services participant devient indisponible pour une raison ou une autre, cela conduira à une invalidation du schéma de composition. De plus, les compositions sont faites selon des suppositions sur les besoins de l'utilisateur.Or, ces besoins sont non seulement variables mais imprévisibles. Un utilisateur peut donc formuler une requête qui n'a pas été prévue par les concepteurs ; de ce fait, même si les services sont disponibles dans l'environnement, il n'y a pas de mécanisme pour les composer et ainsi répondre à sa requête. A cela, on peut ajouter le caractère mobile de ce type de réseau.En effet, si un de ces services composant le schéma tombe en panne ou quitte l'environnement, cela provoquera un échec à chaque appel. D'où la nécessité de dynamiser les approches de composition.

III.4.2.2. La composition dynamique

Différentes approches ont été proposées pour composer automatiquement les services Web.La plupart des recherches peuvent se regrouper selon les axes suivants [60]:

1. **Approches de composition basées sur les Workflows**

Le concept d'un Workflow est que tous les participants dans un processus travaillent ensemble pour l'exécution de celui –ci et ce, du début à la fin. Un workflow est composé d'un ensemble d'activités basiques et formalisées incluant leurs dépendances aux données et l'ordre d'exécution de chaque activité. Un service composite peut donc être perçu comme un Workflow car, comme ce dernier, il inclut un ensemble de services atomiques avec des contrôles et des échanges de données entre eux [63].

Les méthodes de composition de services basées sur les workflows sont classées en deux catégories : statique et dynamique. Dans les méthodes statiques, l'utilisateur du service doit construire un modèle abstrait de processus avant de lancer le processus de composition. Le modèle abstrait inclut une liste de tâches et leurs données fonctionnelles, chaque tâche pouvant être exécutée par un service web différents .Dans la composition statique, seules la sélection et la liaison des web services sont faites de manière automatique. L'approche de

composition de service eFlow [61] est basée sur les workflows statiques qui sont représentés en interne sous forme d'un graphe modifiable dynamiquement durant l'exécution. Il définit l'ordre d'exécution de chaque nœud du processus. Les nœuds peuvent représenter des services ou alors des décisions ou des évènements. Les nœuds qui représentent les services consistent en fait en l'invocation du service atomique ou bien composé. On lance la méthode de recherche à chaque fois qu'un nœud de service est activé et cela est due au fait que la disponibilité de ce service peut être variable (environnements dynamiques). Les nœuds de décisions sont les règles de contrôle du flux d'exécution et les nœuds d'évènements sont les différents types d'événements. La tâche de ces nœuds est d'accroitre la robustesse du système.Si un service web est indisponible, il sera automatiquement replacé par un autre service web qui sera équivalent.

Dans les méthodes de composition dynamiques, la création du modèle de processus et la sélection de services atomiques se font de manière automatique. Cette opération nécessite l'intervention de l'utilisateur pour spécifier les contraintes du système, comme ses préférences,…etc. Un exemple d'une approche basée sur les workflows dynamiques est la méthode implémentée dans le modèle de processus polymorphique (PPM) [52].

2. Approches de composition basées sur l'intelligence artificielle

▪ Le calcul de situation

Le Calcul de Situation [66] [67] est un langage basé sur la logique du premier ordre permettant de représenter et de raisonner sur un domaine dynamique. Ce formalisme a été introduit par John McCarthy et Patrick J. Hayes en 1969 [68]. Les principaux éléments sont les actions, les fluents et les situations. Le monde est conçu comme un arbre de situations, débutant par la situation initiale s_0, et évoluant jusqu'à la nouvelle situation par l'application d'une ou plusieurs actions. Une situation s donnée correspond toujours à un historique de l'ensemble d'actions réalisées sur s_0.

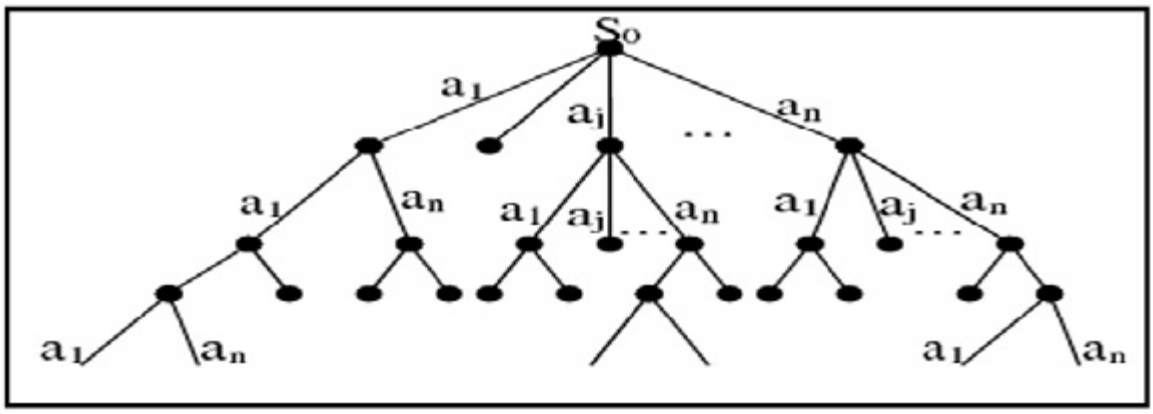

Figure 14 : Le calcul de situation.

Dans cette approche, la dynamique du monde provient d'une succession de situations résultant de l'application d'actions diverses. Comme on peut voir dans la figure 14, une situation représente l'historique des occurrences d'actions. Une situation où aucune action n'a été exécutée auparavant laisse inchangé le monde et correspond à la situation s0. Une nouvelle situation est calculée par la fonction *faire*. Donc, si a est une action et s une situation, le résultat de l'exécution de a dans s est calculé par la fonction *faire* (a, s).

Le problème de la composition de services Web est abordé de la façon suivante : la requête de l'utilisateur et les contraintes des services sont représentées en termes de prédicats du premier ordre dans le langage de calcul situationnel. Les services Web sont transformés en actions (primitives ou complexes) dans le même langage. Puis, à l'aide de règles de déduction et de contraintes, des modèles sont ainsi générés et sont instanciées à l'exécution à partir des préférences de l'utilisateur.

- **Les systèmes multi-agents**

Un agent est une entité autonome soit concrète soit abstraite qui peut raisonner sur elle-même et sur l'environnement dans lequel elle se trouve. Un agent peut communiquer avec d'autres agents et il peut avoir un niveau d'autonomie plus ou moins élevé selon l'environnement dans lequel il évolue et les contraintes que celui-ci lui impose. Chaque agent fournit un service et les agents communiquent en utilisant des messages. On peut voir les web services comme des agents car ils sont autonomes et ont des fonctionnalités très variées.

Les approches de composition basées sur les systèmes multi-agents sont très populaires chez les chercheurs car elles s'adaptent facilement à des environnements dynamiques. En

effet, le processus de composition peut être effectué par des agents. En 2005, Singh propose de considérer la composition comme un problème de planification repartie et les agents de plusieurs systèmes travaillent ensemble pour construire un plan du service composite.

En 2007, Bourdon [65] propose une architecture multi-agents pour faire la composition automatique de services web dont les données et les processus sont décrits en se basant sur la sémantique. En fait, l'idée est qu'un planificateur reparti sera utilisé et les agents peuvent supporter une charge limitée est préfixée par le concepteur. Un service abstrait est représenté par un agent et les agents travaillent ensemble pour construire le plan d'exécution qui doit être formalisé en OWL-S. Le principal avantage est que la base de connaissance d'une organisation peut être mise à jour sans affecter les autres (la base de connaissance doit être distribuée).

En 2007, Charif et Sabouret [75] proposent une architecture orientée service basée sur les SMAs. C'est en fait une chorégraphie de services et l'approche est structurée en couches composées d'agents introspectifs, chacun offrant un service particulier à l'architecture. Cette approche est divisée en quatre étapes : les exigences de l'utilisateur, la découverte de services, la composition de services (chorégraphie) et enfin, le résultat proposé à l'utilisateur.

Les agents sont, ici, les services décrits en VDL-XML. Chaque agent est composé de trois couches : la couche de connaissance (données stockées et publiables),la couche de communication (pour transporter des messages)et la couche traitement des requêtes et gestion de protocole. Chaque agent a un ensemble de données et d'actions (sous forme d'un arbre).Le protocole de coordination pour la chorégraphie des services est basé sur les SMA ; il tient compte de la décomposition dynamique des tâches et aussi des dépendances entre elles, ce qui permet aux agents de se coordonner d'une manière décentralisée.Il utilise deux types d'agents :

- agents participants : ils sont impliqués dans la coordination et interagissent avec d'autres agents afin de satisfaire les requêtes des utilisateurs.
- agents de l'utilisateur : chargés d'interagir avec l'utilisateur et de diffuser sa requête aux agents participants.

Les agents interagissent et coordonnent en fonction des messages reçus et de ce qui se sont déjà échangés. En effet, les agents sauvegardent un historique pour chaque conversation. A chaque envoi ou réception d'un message, la table d'historique est mise à jour. Les auteurs ont défini un ensemble de règles dialectiques dépendant du type de la

requête reçue pour déterminer à chaque fois quels agents devront recevoir les messages et cela, pour éviter la diffusion.

- **La planification**

La planification est un des domaines de l'Intelligence Artificielle qui permet de choisir et d'organiser des actionsen fonction d'un but donné. Elle produit un plan qui est présenté sous la forme d'une collection de descriptions d'opérateurs. Le mot planification est également utilisé dans plusieurs autres domaines tels que l'économie, la politique, l'architecture et encore dans la vie de tous les jours. La planification peut aussi être considérée comme un choix et une organisation d'actions pour changer l'état d'un environnement. Les êtres humains sont toujours en train de faire une action ; il est plus fréquent d'agir que de planifier (Voir la Figure 15). Normalement, la planification est faite quand les humains sont confrontés à des nouvelles situations ou quand les tâches ou les objectifs sont complexes, ou encore quandils ne connaissent pas beaucoup les actions [66].

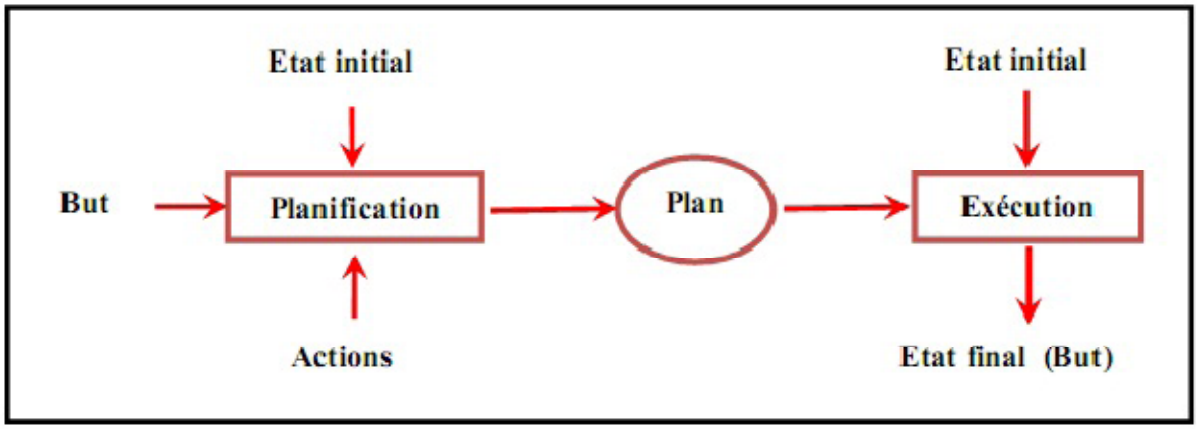

Figure 15 : Le problème de planification.

La planification en Intelligence Artificielle peut aussi être conceptualisée comme un choix et une organisation d'actions pour changer l'état d'un système. Par extension, elle produit un plan qui est présenté sous la forme d'une collection de descriptions d'opérateurs. Le planificateur ou générateur de plans est le système qui produit un plan. L'exécution est la réalisation des actions du plan.

L'exécution d'un plan modifie les propriétés du monde en le faisant évoluer de l'état initial jusqu'au but désiré [67].

La figure 15 représente un problème de planification [69]. La plupart des approches de planification utilise un modèle conceptuel de transition d'état. Plus formellement, le problème de la planification correspond à un quintuplé < S, S_0, G, A, T >, où S est l'ensemble de tous

les états possibles du domaine, $S_0 \subset S$ indique l'état initial de ce domaine, $G \subset S$ représente l'état final du monde que le système de planification veut atteindre, A est l'ensemble d'actions que le planificateur doit exécuter pour transiter d'un état à un autre, et la relation de transition $T \subseteq S \times A \times S$ définie les pré-conditions et les effets pour l'exécution de chaque action. Dans les services Web, S_0 et G sont respectivement les états initiaux et les états finaux spécifiés par le demandeur du service Web, A est l'ensemble des services Web disponibles.

La planification cherche ce qui doit être faitet quand les actions doivent être faites. L'ordonnancement détermine, quant à lui, comment ordonner les actions et également comment allouer les ressources.

Maintenant que nous avons défini le problème de la planification, nous allons établir la relation avec la composition des services Web. L'état initial et l'état final sont associés à la requête d'un utilisateur. Les actions sont des services Web. Cette relation entre la planification et les services Web a permis l'évolution des langages tels que l'OWL-S, qui permet de décrire les services Web ainsi que ses *préconditions* et ses effets. Cette approche a pour but de construire une composition qui constitue une solution à une requête d'un utilisateur.

Un exemple d'une approche basée sur la planification de l'IA est l'outil WSPlan présenté par PEER [70]. Il transforme le problème de composition de services Web en un problème de planification de l'IA. Après avoir créé le domaine de planification en se basant sur la description sémantique, le planificateur chargé de planifier une tâche donnée est sélectionné en fonction du domaine ou de la complexité de l'objectif escompté. Comme la plupart des moteurs de planification, WSPlan utilise le langage PDDL qui est un langage compatible avec l'expression des problèmes de planification. Il permet de décrire efficacement les besoins techniques des domaines de planification ainsi que les capacités du planificateur et ce, de manière uniforme. On peut donc sélectionner le meilleur planificateur pour la composition des services d'une tâche donnée. Cet outil offre la possibilité de faire face au problème de l'information incomplète et hétérogène des différents services web et de spécifier structurellement le but à atteindre. PEER propose donc de faire la planification et l'exécution en simultané : on acquiert des informations, puis on fait la planification, puis on réacquiert des informations…etc. Le cycle se répète jusqu'à obtenir un pan exécutable.

Le principal avantage de cette méthode est de pouvoir *replanifier* en partant d'un échec, c.-à-d. qu'à chaque fois qu'on acquiert de nouvelles informations, elles sont utilisées pour *replanifier* et construire de nouveaux plans plus efficaces et plus directs pour atteindre le but. On recommence jusqu'à ce qu'un plan exécutable soit trouvé ou qu'aucune information ne

soit acquise. Le planificateur qui gère ce processus peut être externe, dans le sens où n'importe quel planificateur qui est en mesure de prendre en charge les informations capturées pouvant être utilisées pour faire la planification et l'exécution. Il y a un inconvénient à cette méthode qui est l'importante charge de calcul qui est due à la fréquence des *replanifications*.

III.5. Composition de service dans un environnement ubiquitaire

III.5.1. Particularités et contraintes d'un environnement ubiquitaire

Dans les environnements ubiquitaire, le défi consiste à offrir à l'utilisateur la possibilité d'exécuter sa tâche en composant les services qu'offrent les composants de l'environnement. Les différentes techniques de composition de services Web décrites plus haut tentent de répondre à la problématique simple de composition de services. Le problème est que les environnements ubiquitaires ont certaines particularités qu'il est nécessaire de prendre en considération lors du choix de la méthode à appliquer et des techniques de composition vues précédemment ne prennent pas en considération ces contraintes. De ce fait, elles ne répondent pas à leurs besoins qui sont entre autres, l'hétérogénéité, la sensibilité au contexte, les paramètres de QoS et la tolérance aux pannes.

L'hétérogénéité est une caractéristique très importante des composants d'un environnement ubiquitaires. Les approches de composition de services ne prennent en compte que très rarement cette caractéristique puisqu'elles considèrent que les services qui seront intégrés sont développés de sorte à être compatible au niveau de leurs interfaces et protocoles de communication.

En ce qui concerne le contexte, l'approche de composition doit implémenter un mécanisme d'adaptation du service offert à l'utilisateur selon les données du contexte. Ce mécanisme doit être en mesure de capturer l'intention de l'utilisateur (le but qu'il veut atteindre) et faire en sorte que le résultat après exécution du processus de composition cadre avec cette intention. La plupart des travaux effectués dans le cadre de la composition de services web s'occupent beaucoup plus de la découverte des services pour fournir des réponses aux requêtes des utilisateurs et négligent les données contextuelles.

Les paramètres de QoS sont : la latence, la disponibilité, le débit, le coût et la sécurité.Pour la sécurité, on ne peut plus vraiment la considérer comme un paramètre de qualité de service vu qu'il devient un critère indispensable.Ces paramètres doivent être pris en considération lors de la conception du protocole de composition de service pour répondre au

mieux à la requête de l'utilisateur et lui offrir un service adapté à ses besoins. Beaucoup des recherches dans le domaine de composition de services dans les systèmes ubiquitaires prennent en considération les paramètres de QoS car les environnements ubiquitaires sont orientés utilisateur.

La tolérance aux pannes doit être en compte lors de la définition d'une approche de composition de services. En effet, dans les services web on utilise plutôt des techniques de contrôle passives ; de ce fait, on n'a pas vraiment de garantie de performances de la méthode de composition qu'on a implémentée. Le problème est que les environnements ubiquitaires sont très mobileset cela rend le processus de composition plus complexe à exécuter car il n'y a aucune certitude sur la disponibilité d'un service à un moment donné. Il y a même certains services qui ne fournissent pas les résultats escomptés et les dispositifs qui contiennent ces services peuvent toujours tomber en panne. C'est pour toutes ces raisons que certains chercheurs se sont penchés sur le développement de techniques de composition qui peuvent gérer des situations d'erreur et les traiter et ce, en intégrant des mécanismes intelligents qui seront chargés de découvrir et de sélectionner de nouveaux services et de les intégrer à la composition.Ainsi, si les services qu'on attendait ne sont plus disponibles ou qu'il y a un problème pour atteindre le but, on peut toujours réagir de manière réactive.

Dans ce qui suit, nous allons présenter quelques approches de composition de services.

III.5.2. Quelques approches de composition de services

III.5.2.1. Composition de Web services base sur la planification adaptative pour la sensibilité au contexte

Les auteurs proposent ici une architecture qui permet la composition dynamique de service pour des applications sensibles au contexte [71]. Cette architecture est basée sur des appels séquentiels à des web services. Les changements de contexte peuvent évidemment conduire à la révision du service composite pendant l'exécution.C'est pour cela que l'application doit être capable d'évoluer dynamiquement.

L'approche utilise le planificateur SHOP2 qui utilise HTN (Hierarchical Task network) pour décomposer une tâche abstraite (composite) en un groupe d'opérateurs pour former un plan d'exécution pour la tâche. L'avantage de cette méthode est que si la séquence

de tâche choisit s'avère être irréalisable, SHOP2 va reboucler et essayer une autre méthode applicable.

L'approche de composition utilise les informations sur le domaine d'application et le contexte courant pour composer le service. La logique du service composite est exprimée avec un langage de composition de flux basé sur XML qui est Business Process Execution Language For Web Services (BPEL4WS).

La requête d'un service composite est constituée de données de contexte et du but à atteindre (ces données sont fourni manuellement par l'utilisateur). L'approche va utiliser BPLE4WS pour déterminer les sous-buts de SHOP2.Ces derniers sont en réalité les branches de l'architecture (les web services atomiques). Les informations de contexte et le domaine d'application sont des fichiers sont directement fournis par l'utilisateur.

La première question à soulever est l'aspect interaction avec l'utilisateur. En effet, le principe même de cette méthode est basé sur le fait que l'utilisateur fournit les données de contexte. Cela conduit à une trop forte sollicitation de l'utilisateur.

La deuxième remarque porte sur la faible prise en charge des données de contexte. En effet, les seules données de contexte que cette approche considère sont celles relatives à l'utilisateur à un moment précis et celles du domaine d'application. Or, le contexte environnemental, temporel ou des dispositifs sont tout aussi importants pour la pertinence de la réponse et le bon déroulement de l'exécution .Ces dernières ainsi que la possible évolution de contexte durant l'exécution ne sont pas traitées dans cette approche.

Un autre problème est que l'hypothèse de départ suppose que les services atomiques seront toujours disponibles. Or, dans un environnement aussi dynamique que les systèmes ubiquitaire, rien n'est moins sûr. Cela rend cette méthode peu réactive aux pannes sans compter le fait qu'elle est basée sur un planificateur centralisé, ce qui accroit encore plus les risques de pannes.

La dernière remarque porte sur le fait que la notion de QoS n'est pas du tout citée alors que nous savons parfaitement que ces paramètres sont primordiaux.Sans cela, la réponse risque de ne pas être adaptée aux réels besoins de l'utilisateur.

III.5.2.2.Un protocole de composition distribué pour les environnements prevasifs

Les auteurs proposent ici un protocole distribué dédiés aux environnements ubiquitaires [72]. Ce protocole est basé sur des brokers qui s'occupent de la composition de services.

Ce protocole est constitué de quatre phases :

1- Le demandeur de service initie la phase de sélection de broker qui analyse la requête composite et élit le nœud broker en conséquence.Le processus de sélection du broker est la phase d'arbitrage du broker. Elle est utilisée pour déterminer le nœud le plus à même d'agir comme un coordinateur de services pour une tâche particulière. De plus, chaque tâche peut être assignée à un broker différent. Ce protocole est capable de s'adapter aux changements de topologies du réseau ainsi qu'aux échecs d'exécution des services.
2- Cela est suivi par la phase de découverte de service où le broker utilise la structure de découverte sous-jacente pour découvrir les services nécessaires pour atteindre le but.Pour la découverte des services, le broker utilise une architecture distribuée. Il va se baser sur le protocole GSD et un mécanisme de découverte qui fonctionne selon le principe de la diffusion ce qui accroît fortement l'efficacité de ce processus et permet une plus grande flexibilité.
3- De là, on passe à la phase d'intégration des services où le broker calcule les liaisons service/nœud pour le service composite. C'est là qu'on génère le flow de service à exécute (ESF) à partir du flow des descriptions(DSF).
4- La phase d'exécution du service est enfin chargée d'exécuter l'ESF, c'est aussi dans cette phase que les erreurs d'exécution sont gérées en utilisant des check-points pour surveiller l'exécution.

Il est vrai que la nature décentralisée de ce protocole et le fait qu'il soit basé sur un système multi-agents fait de lui un protocole très tolérant aux pannes. Il n'en reste pas moins qu'il est très difficile à mettre en place dans un environnement ubiquitaire et ce, pour diverses raisons : ce protocole se base sur la localisation physique de l'utilisateur et des différents dispositifs pour choisir le broker et les services atomiques qui participent à l'exécution mais il n'y a aucune notion de données de contextealors que le contexte influe fortement sur le résultat recherché. Cela rend les services offerts moins adaptés aux réels besoins du client.Enfin, les paramètres de QoS n'ont pas été pris en compte dans cette approche, ce qui

fait que les services qu'elle fournit ne seront probablement pas conformes aux réels besoins des clients.

III.5.2.3. Composition flexible de services d'objets communicants

C'est un mécanisme de composition dynamique, ouvert et sensible au contexte [73]. C'est en fait un composant supplémentaire capable d'interpréter une modélisation des services, de l'exploiter pour évaluer toutes les possibilités afin de trouver un schéma de composition approprié à l'interaction courante puis exécuter les services. Cette infrastructure est constituée de quatre composants :

- les objets de l'environnement : leur rôle est de fournir les services ;
- l'infrastructure de gestion de service : elle est chargée de la publication des services, leurs découvertes, et l'interaction entre des services hétérogènes.
- l'infrastructure de gestion des applications : elle fait la composition de services et ce en créant et en faisant évoluer des applications qu'elle va adapter aux besoins des utilisateurs ;
- le système de gestion du contexte : chargé de récupérer les données contextuelles à partir des capteurs et les fournir aux applications.

Au début du processus de composition, un mécanisme de génération de plan abstrait est lancé. Il récupère un plan abstrait à partir de la librairie de plans abstraits. Pour choisir un service, il se base sur sa description et les informations de contexte. De là, le plan concret est généré et il contient les services sélectionnés et leurs relations. Le mécanisme d'exécution qui déclenche les services sera également chargé de réviser le schéma de composition en cas de problème.Cela est rendu possible grâce à une observation permanente du système et une réactivité de l'adaptation de la méthode de composition des services même pendant leurs exécutions.

Le principal avantage de cette approche est la flexibilité des compositions de services par rapport à la dynamique de disponibilité des services et des informations du contexte. Cette approche tente d'automatiser au maximum le processus de composition sauf dans les cas critiques où certaines décisions doivent être prises par l'humain.

La première étape est d'identifier explicitement chaque tâche afin de trouver une liste de services appropriées pour la remplir ; la deuxième est une vérification de la cohérence, le but

étant de choisir un service pour effectuer chaque tâche de façon à rendre la composition globalement cohérente.

L'approche utilise la description sémantique des services (en utilisant le langage OWL-S) ce qui permet d'intégrer des dispositifs hétérogènes dans l'environnement et offre une gestion souple des applications repartis. Cela permet aussi d'enrichir la description des *préconditions* au fur et à mesure.

Pour choisir les services optimaux pour chacune des tâches identifiées, les auteurs ont proposé d'utiliser une technique de calcul de score d'identification entre une description de service et une description de tâche. Pour ce calcul, ils définissent quatre paramètres à utiliser :

- identification des types de la tâche et du service ;
- identification des types de paramètres (les E/S et les types locaux) ;
- l'évaluation des conditions de contexte ;
- la comparaison des propriétés des E/S.

L'ordre d'application est important cat il permet de réduire la liste de services assez rapidement.

La première question est posée sur la disponibilité des informations de description sémantique des services. En effet, ces informations peuvent être fournies soit par les concepteurs du service soit par le dispositif qui offre ce service dans l'environnement. Le problème est qu'il n'est pas évident que tous les services aient des descriptions complètes et précises et ce n'est pas non plus évident de pouvoir comparer entres les ontologies. Les auteurs ont opté pour la solution de la valeur par défaut dans le cas où une information est indisponible mais cela risque d'affaiblir la pertinence du résultat.

Un autre problème est qu'ils considèrent l'existenced'une librairie de plans abstraits prévus pour des situations particulières. Mais il est impossible de prédire toutes les situations sans compter le fait qu'il risque d'y avoir des problèmes dus au fait que les modèles de situations peuvent être infinies. Les auteurs proposent une alternative à ce problème qui est l'utilisation des SMAs pour faire une composition collaborative des schémas au lieu de se baser sur des plans préétablies. On ne peut être sûr de la qualité de cette solution en termede performances car elle n'a pas été implémentée et testée. Il n'y a donc pas de résultat de performances.

III.5.2.4. Composition flexible de services d'objets communicants pour la communication ambiante

Les auteurs présentent ici une infrastructure basée sur les SMAs jouant le rôle de médiateurs entre les besoins des utilisateurs, les fonctionnalités disponibles et les informations de contexte [76]. Cette infrastructure permet la conception d'environnements de communication ambiante qui s'adaptent en composant dynamiquement les fonctionnalités offertes par des environnements variés. Ce Framework est composé de trois couches :

- la couche inferieure est constituée de services d'objets communicants qui fournissent les différentes fonctionnalités disponibles dans l'environnement.
- la couche intermédiaire est une infrastructure de gestion de services d'objets communicants qui permet de publier, de découvrir et d'interagir avec des services hétérogènes.
- la couche supérieure est, quant à elle, un système de gestion flexible des applications.Elle est chargée de créer et de faire évoluer des applications adaptées aux utilisateurs et cela, en faisant de la composition dynamique de services.

Le système de gestion de contexte fournit les informations de contexte aux différents éléments du système.

L'approche proposée est constituée de trois types d'agents :

- l'agent assistant est chargé de récupérer les besoins courants de l'utilisateur, ces besoins pouvant être directement récupérés à partir de règles de déclanchements contextuelle. Par la suite, cet agent va définir un objectif de composition qui répond à cette requête et le transmettre à l'agent compositeur.
- l'agent compositeur va extraire les caractéristiques des fonctionnalités nécessaires et déléguer la recherche de services correspondants aux agents superviseurs.Après cela, il évaluera les compositions possibles et mettra en relation les services concernés.Le mécanisme de composition est basé sur trois concept fondamentaux qui sont l'utilisation d'une modélisation sémantique des services, les comportements attendus étant exprimés sous forme de buts .Enfin, la composition des fonctionnalités est soumise à une surveillance réactive et continue, et ce, dans le but de réagir immédiatement à une éventuelle défaillance.

- l'agent superviseur est responsable de proposer le service le plus approprié pour réaliser la fonctionnalité dont il est responsable. Pour ce faire, il évalue les services offerts par l'infrastructure par rapport au contexte courant et choisit celui qui correspond le mieux en prenant en compte le niveau d'interopérabilité avec les services proposés par d'autres superviseurs.

Il est vrai que l'architecture proposée permet une grande flexibilité ; néanmoins, à part le fait qu'elle est dynamique, elle n'apporte aucune amélioration par rapport à l'approche précédente proposée par les mêmes auteurs.

III.5.2.5. Approche de composition de service tolérante aux pannes en environnement ubiquitaire: WS-Pro/ASCT

Dans cette approche [91], les scénarios des services composites requis sont représentés en utilisant des descriptions abstraites ASCT (Abstract Service Composition Template) et les flux de service composites sont décrit avec FTQSPN (Finite Population Queuing System Petri Nets) qui est une extension des réseaux de Petri classique. Cela permet d'avoir un modèle mathématique pour analyser les performances d'exécution des services.

Pour construire le service composite, l'architecture procède comme suit ; D'abord, l'ASCT est construit en sélectionnant les services appropriés. Ces derniers seront ensuite transformés en FTQSPN pour permettre de sélectionner les meilleurs services et pour que les performances du service composites soient optimales. Cette phase permet également de procéder à une vérification de la correspondance sémantique du service composite.

Les auteurs proposent également d'utiliser des capteurs virtuels qui permettent d'améliorer la disponibilité et la qualité des données. Le Framework proposé ici permet de faire faces aux pannes pouvant être imputées à la défaillance de certains dispositifs ou encore à des changements dans l'environnement ; le mécanisme de tolérance aux pannes étant basé sur le principe de *replanification*.

Il est vrai que cette approche a beaucoup de qualités ; néanmoins, elle ne prend en compte ni le contexte ni la QoS dans le processus de constructions des services composites. Pourtant, ces deux paramètres sont au centre des préoccupations dans les environnements ubiquitaires.

III.5.2.6.Un modèle de QoS pour la composition de services base sur les requêtes

Dans cette approche [92], le système est basé sur l'utilisation du modèle de tâche de l'utilisateur (Task Template) pour la composition automatique de services en environnement ubiquitaire. Le modèle de composition de service est basé sur l'intégration de la QoS au schéma de gestion des ressources.

Le système est constitué de quatre composants principaux : le Service Assembly (SA), le Service Discovery (SD), le Service Composition Template (SCT) et le Service Composite Candidats (SCC). Le SA est chargé d'interpréter la description de la tâche et de la composition du service qui lui correspond tout en maximisant la QoS de la réponse donnée à la requête de l'utilisateur. Pour cela, il prend en compte les ressources appropriées et l'estimation de leurs disponibilités. Il introduit également le coût du processus de composition. SA construit une requête en utilisant la description de la tâche que donne l'utilisateur ainsi que les préférences de ce dernier.

La requête de l'utilisateur contient un modèle de composition de service (SCT) qui décrit les services nécessaires à l'accomplissement de la tâche. Cette requête est envoyée au SD.Celui-ci contient les descriptions statiques des services et des dispositifs et il fournit aussi un matchmaking pour les requêtes. Lorsque le SD reçoit le SCT, il va construire les services composites candidats possible (SCCs) à l'aide du matchmaking. Apres la réception des SCCs, le SA va les classer selon la QoS qu'ils offrent.

En plus de ces quatre composants, le système contient un gestionnaire de ressources qui maintient l'état des ressources déployées dans l'environnement. Il souscrit les informations contextuelles pertinentes à partir des critères de validité du gestionnaire du contexte.

Le principal défaut de cette approche est qu'elle n'implémente pas de mécanisme de tolérance aux pannes, ce qui peut être problématique compte tenu de la nature dynamique des environnements ubiquitaires. En effet, du fait de l'ouverture et de la forte mobilité dans ces environnements, il devient difficile de garantir une stabilité des services offerts et des ressources. Il est donc indispensable de prévoir un moyen de pallier ce problème.

III.5.2.7.Approche de composition de services basé sur les règles pour les environnements ubiquitaires

Cette approche est basée sur la planification dynamique [74]. Elle est modélisée en couches ce qui la rend flexible et tolérante aux failles. Les auteurs sont partis du principe que

tous les services sont déjà découverts et sont disponibles dans un annuaire et qu'ils peuvent interagir entre eux.

Dans cette approche, il y'a deux types de services : les services abstraits et les services concrets. Les auteurs ont définis le service concret (CS_i) comme un service qui fournit une fonctionnalité et ce, en agissant sur des données d'entrés dans le but de fournir des données de sortie ; et un service abstrait (AS_i) comme une tâche pouvant être effectuée par plusieurs services concrets offrant la même fonctionnalité en utilisant les mêmes données d'entrées. Cette infrastructure est composée de trois couches :

- la couche 1 fait la génération automatique du plan général en utilisant des règles d'inférence définit par les auteurs, ce plan va contenir tous les services abstraits qu'on doit utiliser pour fournir le service composé.
- la couche2utilise le plan généré par la couche1 pour d'abord générer un plan abstrait optimal et aussi régénérer un nouveau plan optimal en cas d'échec. Le plan optimal est sélectionné en se basant sur la réputation des services abstrait et de la complémentarité de leurs paramètres de sortie. Après la sélection du plan optimal on sélectionne un service concret pour chaque service abstrait, un service optimal est sélectionné en prenant en considération les données contextuelles et les paramètres de QoS. puis on instancie l'exécution du plan.
- la couche 3 est chargée de gérer cette exécution du plan en l'adaptant automatiquement en cas de panne. L'adaptation du plan est en fait le remplacement d'un service concret par un autre qui appartient à la même classe de service abstrait.

Une question peut être soulevée par rapport à la génération du premier plan abstrait. En effet, si on suppose que dans l'annuaire de services nous avons des centaines voire des milliers de services disponibles et autant de possibilités d'orchestrations, cela pourrait conduire à une explosion de l'arbre.

Enfin, cette approche est très performante mais la supposition de départ qui est d'avoir un annuaire prêt et des services décrits dans un même langage la rend difficile à mettre en œuvre dans n'importe quel environnement.

III.5.2.8.Framework de composition de services base sur la QoS pour la robotique ubiquitaire

Les auteurs proposent ici un Framework de composition de servicesqui permet de prendre en compte l'aspect dynamique et stochastique de l'environnement ubiquitaire [90].

Dans cette approche, on peut composer de deux façons

- la composition verticale consiste à définir un plan approprié de services pour effectuer la tâche de composition. Elle se base sur deux algorithmes : un algorithme de composition FCoSC (Feasibility and Construction of a Services Composition) et un algorithme de découverte automatique SDT (Services Discovery for a Task).
- la composition horizontale consiste à déterminer le service concret le plus approprié parmi tous les services fournissant la même tâche.

Néanmoins, les auteurs se sont plus penchés sur la composition horizontale où on explore le contexte des services et leurs probabilités de réponse. Selon la spécification OWL-S les auteurs ont défini deux types de services à savoir: les services abstraits et les services concrets (représentant les mêmes concepts que dans l'approche [74]).

Un service concret est une action qui peut être effectuée dans un environnement incertain ; pour cela, deux types de réponses peuvent se présentées : positiveavec une probabilité p et négative avec une probabilité q= 1-p. Une probabilité représente l'estimation de réponse, cette estimation pouvant être imprécise ou complètement inconnue. Pour cela, les auteurs ont introduit le mécanisme d'apprentissage Bayésien.

Trois mécanismes sont à la base de la méthode proposée, à savoir : l'apprentissage Bayésien pour l'invocation des services, la découverte automatique des services et la recomposition automatique et dynamique du plan de composition des services.

Ce Framework utilise un modèle en couches pour séparer les trois étapes de processus de composition suivantes :

- composition de services abstraits,
- plan d'exécution,
- découverte et recomposition de services abstraits.

L'algorithme de composition de service basé sur la QoS se base sur l'algorithme FCoSC pour construire le plan de composition abstrait. Par la suite, ce plan passe à l'exécution dans un ordre séquentiel. Chaque composant de ce plan constitue un service abstrait.Son action est sélectionnée en utilisant l'algorithme QoSEA.Ensuite, il intercale l'exécution de l'action et

l'apprentissage Bayésien de leurs probabilités. Quand la réponse d'une action est négative, cet algorithme essaie de la remplacer immédiatement par la meilleure action de service courant.Si toutes ses actions ne répondent pas après l'excès du paramètre (local precision learning parameter), le service abstrait est considéré comme échoué. L'algorithme SDT est alors exécuté à ce niveau dans l'ordre de remplacer le service échoué. Le but ici représente juste le paramètre de sortie (output) de service échoué.Ce paramètre est enregistré dans la table de marquage (marking table) générée par l'algorithme FCoSC. Si aucun service n'est valable, le mécanisme de recomposition est alors exécuté pour générer un nouveau plan et la probabilité de réponse p (recomposition) est calculée en utilisant l'apprentissage bayésien. Le paramètre d'apprentissage global (global precision learning parameter) est défini à ce niveau pour contrôler le processus de recomposition.

Il est vrai que cette approche apporte un plus avec le processus de sélection basé sur le mécanisme d'apprentissage Bayésien, mais le fait de partir des entrées pour tenter d'atteindre le but peut être un problème. En effet, il y a plusieurs risques à cela .Par exemple, il peut y avoir des redondances de services dans le plan de composition et le processus de composition peut tourner infiniment sans atteindre le but.Si celui-ci n'est pas atteignable, le mécanisme mettra beaucoup de temps à le réaliser.

III.5.2.9.Approche de composition de services sensibles au contexte pour les environnements prevasifs

La composition de services sensibles au contexte permet aux systèmes ubiquitaires de fournir à leurs utilisateurs des applications pertinentes et adaptées aux informations contextuelles du réseau, des dispositifs et des utilisateurs. Ces applications sont flexibles et facilement adaptables aux différents changements pouvant survenir dans l'environnement.

Les auteurs définissent ici[77] le contexte comme toute information explicite ou implicite caractérisant une interaction entre un utilisateur et son monde de services. C'est non seulement une information qui caractérise une entité mais il est également utilisé pour initier, contrôler et maintenir une session de tâches. Les auteurs ont divisé le contexte en plusieurs catégories : le contexte utilisateur, le contexte processus, le contexte physique, le contexte service et le contexte dispositif.

L'architecture proposée est divisée en deux parties : la première est chargée de la gestion du contexte et la deuxième de la composition de service.

Les principaux composants de l'architecture sont les suivants :

- d'abord, le « context sensing » récupère les informations contextuelle et les transmet au reste du système ;
- puisle « context modeling » va acquérir ces données et les convertir afin qu'elles soient interprétables ;
- le « context reasoning » sera ensuite, chargé de prendre les décisions en examinant les informations contextuelles ou encore la requête de l'utilisateur et en comparant ces données avec le modèle de contexte prédéfini;
- le « context storage » stocke les anciennes données de contexte, cet historique pouvant être utilisé pour établir des tendances et prédire les prochaines valeurs de contexte ;
- le « composition adaptor » reçoit des commandes à partir du « context reasoning », puis il choisit et exécute la composition de services sensibles au contexte en respectant la coordination des dispositifs, l'adaptation de l'exécution ou encore l'utilité d'adaptation des services.

Les applications sont composées par un processus service qui joue le rôle de compositeur avec des services utiles pour chaque tâche.Un service utile offre des fonctionnalités réutilisables reliées à des données qu'on peut prédire sur d'anciennes applications. Les services utiles peuvent être fournis et maintenus par différents dispositifs. Les processus services lient entre le processus logique et l'interaction avec le service et font la gestion du Workflow.

Dans le développement d'applications CASPC, les dispositifs de services, les services processus et les services d'utilité sont les trois agents de cœur qui sont impliquées dans la composition de services.

Cette approche est plus performante que les autres car elle est basée sur un raisonnement logique sur les informations contextuelles dans le choix des services qui composeront le workflow d'exécution. Néanmoins, elle nécessite une synchronisation parfaite entre les agents qui composent le Framework.En cas d'échec d'un agent, tout le processus est compromis. En plus de cela, cette approche ne prend pas en considération la QoS.

III.6. Comparaison

Le tableau suivant est un récapitulatif des principales caractéristiques de chaque approche, il va nous permettre de les comparer **:**

	Technique utilisée	**automatique**	**Dynamique vs statique**	**Sensibilité au contexte**	**QoS**	**sémantique**	**Tolérance aux pannes**	**Centralisé vs distribué**
[71]	Planification	Oui	dynamique	oui	non	non	non	centralisé
[72]	SMAs	Non	statique	non	non	non	oui	distribué
[73]	workflows	Oui	statique	oui	non	oui	non	centralisé
[74]	Planification	Oui	dynamique	oui	oui	oui	oui	centralisé
[76]	SMAs	Oui	dynamique	oui	non	oui	non	distribué
[77]	SMAs	Oui	dynamique	oui	non	oui	oui	distribué
[90]	Planification	oui	dynamique	oui	oui	oui	oui	centralisé
[91]	FTQSPN ASCT	oui	dynamique	non	non	oui	oui	centralisé
[92]	Template	oui	dynamique	oui	oui	non	non	centralisé

Tableau 3 : Tableau comparatif entre les différentes approches de composition de services.

- **La technique utilisée pour la composition**

Dans [71], [74] et [90], les auteurs utilisent la planification de l'IA.Dans [72], [76] et [77] , les systèmes multi-agents. Dans [73] , les workflows et, dans [91] et [92], des techniques de modélisation.

Le problème des workflows, est qu'il faut que l'utilisateur définisse lui-même le modèle du processus pour qu'ensuite un programme automatique puisse se charger de trouver les services qui peuvent répondre à la requête.

Par contre les techniques de planification de l'IA peuvent être utilisées dans le cas où il n'y a pas de modèle de processus précis, l'utilisateur va juste spécifier ses préférences et quelques contraintes à respecter.De là, un modèle de processus sera automatiquement généré. Il y a néanmoins un problème avec ces techniques : le fait qu'elles se basent sur une théorie où toute action externe au planificateur n'est pas prise en considération.Or,, les environnements ubiquitaires sont très dynamiques ; donc, certaines actions externes à l'environnement peuvent se déclencher sans que le planificateur ait assez d'informations sur elles.

Les SMAs, quant à eux, se basent sur le fait que les agents s'unissent et travaillent en collaboration pour construire un service composite demandé par un utilisateur.Cela rend ces systèmes plus flexibles et plus ouverts.

- **Le degré d'automatisation**

Comme on peut le voir dans le tableau, toutes les techniques sont automatiques sauf celles qui utilisent les SMAs [72]. Il est vrai qu'une certaine interaction entre l'utilisateur et le mécanisme de composition est utile pour spécifier ses besoins et ses préférences ; néanmoins, c'est à l'encontre de la philosophie d'un environnement ubiquitaire qui est de fournir un service informatique transparent pour ce dernier.

- **Dynamique VS statique**

Comme on peut le constater, dans [71], [74], [76], [77], [90], [91] et [92] la construction du schéma de composition est dynamique ;contrairement à [72] et [73] , elle est statique.

Le problème avec les techniques statiques c'est qu'elles ne sont efficaces que dans des environnements fermés ou les services sont préalablement connus. Or la principale caractéristique des environnements ubiquitaires c'est leur ouverture et la forte mobilité des dispositifs ; de ce fait, même si les chorégraphies qui ont été établies pour un mécanisme sont très performantes, il suffit qu'un des services atomiques quitte l'environnement ou qu'un service offrant les mêmes fonctionnalités avec de meilleures performances rejoignent le système pour que cette technique devienne obsolète.

- **Sensibilité au contexte**

On constate que toutes les approches étudiées a l'exception de [72] et [91] sont sensibles au contexteet ce, pour une bonne raison. En effet, la prise en compte des données contextuelles permet une meilleure adaptation et une personnalisation des services offerts selon les besoins de l'utilisateur car un service peut répondre à la requête en termes de paramètres fonctionnels.Cependant, il est inutilisable pour l'utilisateur car il est incompatible avec le contexte (exemple : afficher un message personnel sur l'écran de la salle de réunion, afficher un PDF sur le téléphone …).

- **Prise en compte de la QoS**

Sur les approches vues précédemment les seules approches qui prennent en compte la QoS sont [74], [90] et [92].

Les systèmes ubiquitaires sont orientés environnement et utilisateur. Pour qu'un utilisateur soit satisfait, le processus de composition de services doit prendre en considération la qualité de service pour fournir un service adéquat à la requête de ce dernier.Le mécanisme de composition doit donc être en mesure de satisfaire les préférences et répondre aux exigences des utilisateurs.

- **Prise en charge de la sémantique des descriptions**

Dans [71], [72] et [92],la sémantique n'est pas prise en charge alors qu'elle l'est dans [73], [74], [76], [77], [90] et [91].

Les paramètres fonctionnelles peuvent décrire un service mais ne fournissent pas assez de données pour bien identifier les offres et les avantages de chaque service atomique. En effet, deux services peuvent offrir les mêmes fonctionnalités mais l'un est certainement meilleur que l'autre, sans compter le fait que l'utilisateur va décrire sa demande à sa manière. Il serait intéressant de pouvoir l'analyser sémantiquement et cela, pour pouvoir offrir le service adéquat.

- **La tolérance aux pannes**

Parmi les approches étudiées, [71], [73], [76] et [92] n'implémentent pas de mécanisme de tolérance aux pannes, contrairement à [72], [74], [77], [90] et [91].

Le problème est que dans ce type d'environnement, les pannes sont fréquentes.Elles sont dues à des déconnexions, des départs de certains dispositifs ou encore à une surcharge de certains dispositifs.

Le fait de ne pas pouvoir réagir immédiatement à ces pannes peut induire à un manque de satisfaction des clients car si à chaque problème, il faut reformuler sa requête, le mécanisme deviendra inutile alors que ce domaine est très compétitif.

III.7. Conclusion

Dans ce chapitre, nous avons présenté quelques approches de composition de services chacune d'elles utilise une technique différente de compositions.

A partir de là, nous déduisons que pour maximiser les performances et correspondre au mieux aux besoins des utilisateurs et des environnements ubiquitaires, une approche de composition doit être :

- dynamique car les environnements étant mobiles et instables, l'offre doit s'adapter à la disponibilité des services, des utilisateurs et des dispositifs.
- automatique, car 'l'implication de l'utilisateur dans le processus de composition est contraire à la philosophie des environnements ubiquitaires qui est d'offrir un service informatique à l'utilisateur sans qu'il s'en rende compte.
- distribuée sur plusieurs dispositifs pour éviter de surcharger un seul dispositif, en général, ces derniers ayant des capacités réduites en termes de batterie et de capacité de calculs.
- tolérante aux pannes ;en effet, les pannes dues à une indisponibilité du service ou du dispositif contenant le service sont fréquentes à cause de la mobilité de ce type d'environnement ou encore à des changements de contexte.Cela conduit à la nécessité d'inclure des mécanismes de tolérance aux pannes pour assurer le service quoi qu'il arrive.
- sensible au contexte ;en effet, le contexte influe fortement sur la nature du résultat désiré et pour que ce résultat corresponde au mieux aux réels besoins de l'utilisateur, l'approche de compositions doit pouvoir adapter son schéma au contexte courant.
- en mesure de gérer la sémantique car les requêtes des utilisateurs sont formulées par un langage informel ; il faut donc pouvoir interpréter cette requête pour répondre au mieux à sa demande.
- capable de prendre en considération la QoS, et cela pour offrir un service de qualité à l'utilisateur.

D'après ce que nous avons vu dans ce chapitre, la technique de composition qui remplit le plus ces critères est la technique des SMAs. Dans le prochain chapitre, nous proposerons une approche de composition basée sur les SMAs en tentant de satisfaire les principales exigences des environnements ubiquitaires.

Chapitre IV : Composition flexible de services sensible au contexte

IV.1. Introduction

Dans les chapitres précédents, nous avons décrit les environnements ubiquitaires comme étant très dynamiques et intégrant plusieurs technologies hétérogènes et indépendantes les unes des autres, mais pouvant néanmoins communiquer entre elles et interagir dans le but de s'échanger des fonctionnalités permettant une évolution mutuelle. Un environnement ubiquitaire peut donc être perçu comme un environnement multi-agents intégrant de façon transparente des agents intelligents et autonomes. La principale difficulté de ce type d'environnement réside dans la satisfaction des requêtes complexes des utilisateurs car il faut intégrer plusieurs équipements selon leurs disponibilité et les fonctionnalités qu'ils offrent, ils doivent pouvoir réaliser les différentes tâches de la requête et être disponible au moment de l'arrivée de cette dernière, sans oublier que le processus de réalisation de la requête doit prendre en considération certaines contraintes dépendant du contexte d'exécution et de la qualité de service exigée par l'utilisateur.

L'objectif de notre travail est de développer un mécanisme de composition qui fonctionne de manière dynamique et décentralisée, et cela dans un environnement ouvert et mobile. Le résultat recherché par un tel mécanisme est la satisfaction de la requête de l'utilisateur tout en prenant en compte les contraintes spécifiées par l'utilisateur et le contexte dans lequel cette application sera exécutée.Cela signifie que nous devons disposer d'un Framework qui permet de gérer ces données dans le sens où elles doivent être récupérés à partir de l'environnement, modélisés, stockés et fournie aux différents acteurs de l'approche qui sont chargés d'effectuer le processus de composition. Néanmoins, si les contraintes sont trop strictes, nous prenons le risque de ne pas répondre du tout à la requête de l'utilisateur, cela, même si avec une réduction de l'exigence, la requête sera réalisable. C'est pour cela que nous choisissons de nous appuyer sur un principe simple, une réponse même approximative vaut mieux qu'aucune réponse du tout.

Dans ce qui suit, nous détaillerons les différents aspects constituants notre approche.Nous commencerons par fixer les suppositions de départs sur lesquelles nous comptons nous appuyer pour développer notre mécanisme.Par la suite, nous décrirons en détail l'architecture conçue et les différentes phases d'exécution du processus de composition.Nous terminerons par une conclusion.

IV.2. Suppositions de départ

Pour la réalisation de notre système de composition, nous allons partir du principe que :

1. Les services concrets sont déjà découverts, publiés et classés en services abstraits en fonction de leurs paramètres d'entrées/sorties ;
2. Chaque dispositif de l'environnement dispose de la liste des services abstraits qui sont disponibles dans l'environnement, cette liste étant périodiquement mise à jour ;
3. Tous les services sont décrits avec le même langage (OWL-S) ;
4. Le Framework de gestion de contexte est chargé de la collecte, de la modélisation, du stockage et de la dissémination des informations contextuelles aux applications.

IV.3. Modèle de contexte

IV.3.1. Description du contexte

Le contexte est toute information externe à l'application qui influe sur le comportement de celle-ci et ce, en donnant un nouvel usage aux données dont dispose les applications pour leurs exécutions. Par exemple, si un utilisateur reçoit un message pendant qu'il est en réunion, ce message ne sera pas affiché sur le grand écran mais plutôt sur son PDAs. Les données contextuelles sont dynamiques, elles peuvent évoluer entre le début et la fin de l'exécution d'une application qui les utilisent. Par exemple, un utilisateur peut se déplacer donc changer de localisation pendant qu'il utilise un service.

Les services qui participent à la tâche doivent être choisis selon les données contextuelles de l'environnement. Afin que cela soit possible, chaque service doit s'adapter au contexte, ce qui signifie que selon le contexte, la description qu'offre le service est différente.

IV.3.2. Structuration du contexte

Nous allons structurer notre contexte selon les classes suivantes qui correspondent aux catégories de contexte étudié dans le chapitre 2.

- Le contexte utilisateur : il correspond à l'identifiant de l'utilisateur, son profil, sa localisation, ses émotions, son état (connecté ou pas), son activité (en réunion, marche,..), ses préférences, le matériel dont il dispose, ses exigences de qualité de service.

- Le contexte environnemental est relatif à tout ce qui entoure le système niveau de bruit, température humidité…etc.
- Le contexte physique est relatif aux dispositifs de l'environnement leurs localisations, le niveau de batterie, capacité de stockages performances…etc.
- Le contexte de service concerne sa localisation (dans quel dispositif il est stocké), le type de logiciel, la description, le QoS qu'il offre…
- Le contexte temporel correspond aux dates, aux historiques, etc.

IV.3.3. Modélisation du contexte

Parmi les modélisations du contexte possible que nous avons vu dans le chapitre 2, nous avons opté pour une modélisation en UML ; le diagramme de classe suivant est une modélisation de notre contexte :

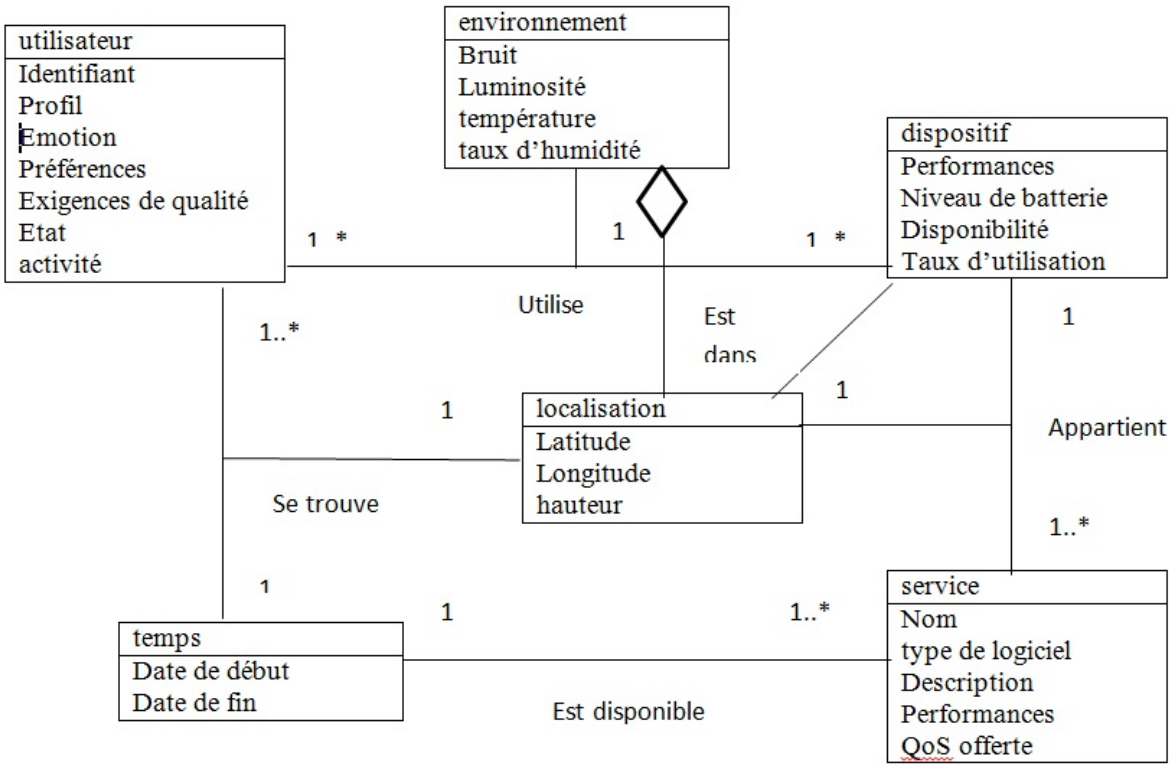

Figure 16 : Modélisation du contexte.

- Un ou plusieurs **utilisateurs***se trouvent* à un **endroit précis** durant un **temps** limité.
- Un ou plusieurs **utilisateurs***utilisent*un ou plusieurs **dispositifs** d'un **environnement.**
- Un ou plusieurs **services** *sont disponibles* pendant un **temps** limité.
- Un ou plusieurs **services** *appartiennent* à un même **dispositif localisé** à un endroit précis.

Nous supposons que nous construisons une maison intelligente dont l'objectif est de permettre aux utilisateurs d'accéder à plusieurs services disponibles (allumage des lumières, ouverture des portes, chauffage central, services de messageries,…etc.).Les utilisateurs ont accès à un certain nombre de dispositifs par lesquels ils peuvent lancer des services qui peuvent être simples ou complexes. Les pièces de la maison sont donc des pièces intelligentes. Par exemple, dès qu'un utilisateur sera à proximité d'une porte, celle-ci s'ouvrira automatiquement. Cela sera possible grâce aux capteurs de présences et aux applications « ouverture de porte » qui se lancent automatiquement dès que les capteurs de présence envoie un signal à l'application.

Par exemple, si le niveau de bruit est faible et que la luminosité est bonne (lumières allumé), alors il faudra que l'application « éteindre les lumières » soit lancé vu qu'il n'y a personne dans la salle.

Si le taux d'humidité est trop haut ou trop bas, il faudra lancer les aérateurs ou alors ouvrir les fenêtres (dans les pièces intelligentes, tout cela est fait automatiquement en lançant simplement des services informatiques).

Pour ce qui est de l'activité, au moment de la vérification, Alice est près du bureau : on en déduit qu'elle travail, nous pouvons donc lancer le PC ; quant à Bob, il est près de la porte, il vient alors d'arriver, nous lancerons donc l'application « ouverture de porte ».

Quant aux préférences, on suppose par exemple qu'Alice a pour habitude de préférer les applications à faible interaction avec l'utilisateur (totalement automatisé) et Bob préfère participer au processus de prise de décision du service.

Finalement, un pc est utilisé par Alice. Seulement, il est à 50% d'utilisation (on peut faire tourner plus d'application dessus sans qu'il soit dépassé).

Nous pouvons conclure que le modèle proposé est assez général et complet pour pouvoir être enrichi et appliqué à des situations particulières.Il est possible d'ajouter ou de supprimer certains attributs dans chaque classe selon le domaine étudié. Cela permet de couvrir un grand nombre de spécifications contextuelles dans le sens où, pour chaque domaine, il est possible d'adapter le modèle et de l'instancier tout en englobant la totalité du contexte correspondant.

IV.4. Architecture du système de composition de services

La figure suivante représente les différents éléments qui constituent l'architecture de composition de services inspiré par [76] et [80]. La première couche est composée de différents dispositifs communicants de l'environnement :ce sont ces objets qui fournissent les différentes fonctionnalités aux utilisateurs. La deuxième couche est une infrastructure de service SOA de gestion de services d'objets communicants : elle permet la publication, la découverte, la distribution et l'interaction entre services hétérogènes. La troisième couche est un Framework de gestion de contexte : il fournit des données de contexte aux différents éléments du système et ce, en récupérant ces données à partir des capteurs disséminés dans l'environnement, puis en les modélisant afin qu'elles soient compréhensibles par tous les dispositifs impliqués dans le processus de composition. La dernière couche est chargée de la création d'applications complexes et adaptables aux besoins des utilisateurs et ce, en composant dynamiquement les services atomiques disponibles dans l'environnement.

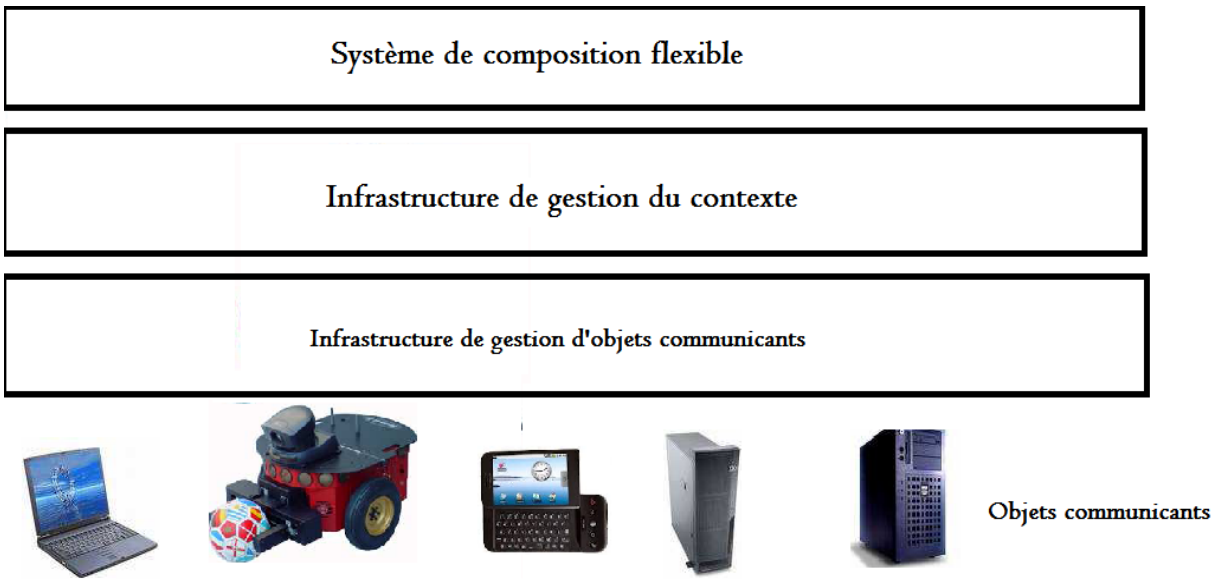

Figure 17 : Architecture générale du système.

IV.4.1. Principe de la composition flexible de services

En règle générale, les applications des environnements ubiquitaires coordonnent des fonctionnalités issues d'objets communicants variés. Une composition de services a pour objectif de définir la manière dont ces fonctionnalités interagissent afin de réaliser le comportement attendu pour l'application. Indépendamment des situations, diverses compositions de services peuvent ainsi être mises en place pour la réalisation d'une

application adaptée. En partant de ce postulat, nous proposons une approche de composition de services fondée sur les principes suivants :nous avons construit notre mécanisme de composition en nous basant sur la coordination multi-agents, chaquephase du processus de composition étant effectuée par un type d'agents choisis parmi les dispositifs de l'environnement selon leurs capacités à prendre en charge la tâche.

Pour générer le plan de composition, nous choisissons la méthode de planification par chaînage arrière, c'est-à-dire que nous partons du but à atteindre (ce que demande l'utilisateur) et nous essayons d'atteindre les paramètres d'entrée (données par l'utilisateur).

Notre mécanisme de sélection de services est fortement dépendant de l'environnement ambiant, en d'autres termes, du contexte d'exécution.Les services sont sélectionnés selon leur degré de correspondance au contexte. Pour plus de précision dans la sélection du service, nous choisissons de donner un poids d'importance aux données de contexte, ces poids représentant en réalité les préférences des utilisateurs données au moment de la première connexion à l'environnement. De la même manière, nous traitons les paramètres de QoS comme des informations de contexte relatives aux préférences de l'utilisateur et aux performances des services qui sont fournies dans leurs descriptions non fonctionnelles.

Le système que nous proposons doit rendre l'utilisation des applications informatiques pour les utilisateurs la plus implicite possible.Pour ce faire, nous établirons une base de règles de déclanchement pour certains services selon le contexte courant.Cette tâche sera assurée par un agent abonné aux mises à jours de contexte.

Dans notre approche, nous considérons deux types de services, à savoir les services concrets et les services abstraits.

IV.4.1.1. Modélisation du service concret

Un service concret est une action effective agissant sur les entrées pour réaliser une fonctionnalité particulière et pour produire des sorties faisant office de résultats. Un service concret noté CS_i est décrit comme suit [74][80] :

- CS_i^{in} : sont les paramètres d'entrées du service.
- CS_i^{out} : sont les paramètres de sortie du service.
- Prec : les *préconditions* du service
- Eff : les effets du service

- Cxt : l'ensemble des attributs du contexte à les quels le service est sensible.

IV.4.1.2. Modélisation du service abstrait

Un service abstrait est un ensemble de services concrets qui ont les mêmes paramètres d'entrées et fournissent les même paramètres de sorties. Un service abstrait noté AS_i est décrit comme suit [74][80]:

- AS_i^{in}: sont les paramètres d'entrées du service abstrait AS_i
- AS_i^{out}:sont les paramètres de sortie du service abstrait AS_i
- CS : la liste des services concrets

IV.4.2. Mode de fonctionnement du système

Le processus de composition est divisé en plusieurs phases :

- formulation de la requête de composition ;
- génération du plan abstrait pour chaque tâche ;
- sélection des services concrets participant et établissement du plan d'exécution de chaque tâche ;
- exécution du service composite.

Toutes ces phases sont exécutées par différents types d'agents et les agents sont choisis parmi les dispositifs de l'environnement selon leurs capacités à prendre en charge le processus.

L'exécution de chacune de ces phases est répétée jusqu'à atteindre le but. Au final, on se retrouve avec plusieurs agents qui interagissent durant tout le processus de composition. La figure suivante représente le processus d'exécution de notre mécanisme de composition :

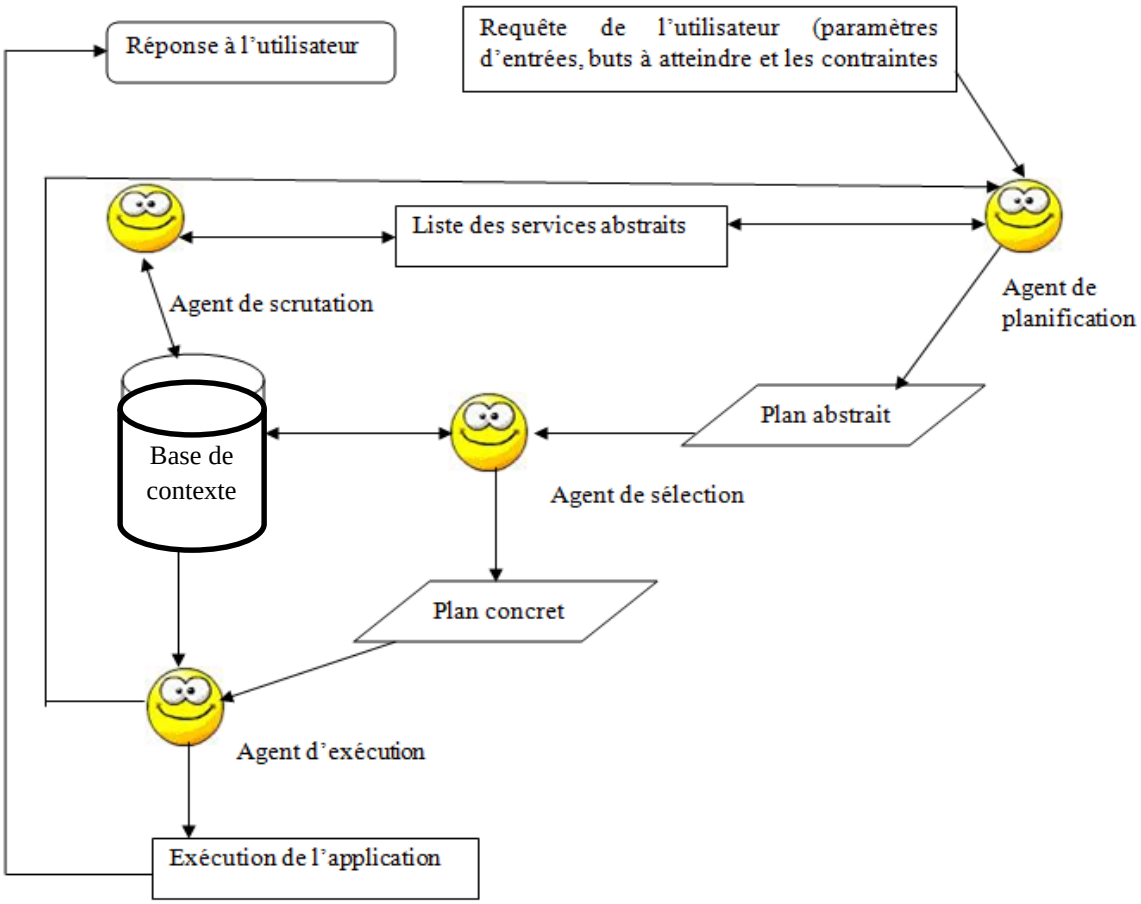

Figure 18 : Processus de composition dynamique.

IV.4.2.1. Formulation de la requête

La première étape est faite à la demande de l'utilisateur.Celui-ci va formuler sa requête via une interface utilisateur. Dans sa requête, il spécifie les paramètres dont il dispose (les entrées) et ce qu'il recherche (les sorties du système ou le but à atteindre). En plus de ces deux types de paramètres, il peut également ajouter quelques contraintes à condition que celles-ci ne soient pas déjà spécifiées dans ses préférences.

Les paramètres non-fonctionnels seront traités comme des données de contexte relatives à l'utilisateur. Ils seront donc directement traités par le Framework de gestion de contexte. L'agent de planification va, quant à lui, récupérer les sorties demandées qui seront le but de la requête et va les utiliser pour générer le plan abstrait d'exécution

IV.4.2.2. Génération du plan abstrait

C'est l'agent de planification qui s'occupe de générer le plan de composition abstrait. Il a à sa disposition les paramètres d'entrées de la requête, le but à atteindre et la liste des services

abstraits disponibles dans l'environnement. Pour construire le plan abstrait, l'agent planificateur utilise un algorithme de planification par chaînage arrière et construit une chaine de services pouvant fournir les sorties nécessaires étant donné les entrées dont il dispose. L'agent va générer le plan comme suit :sélectionner les services abstraits qui ont comme paramètres de sortie le but à atteindre, ce processus s'arrêtant une fois que tous les paramètres du but sont récupérés. Il va ajouter les services abstraits sélectionnés au plan abstrait en leur affectant un niveau d'exécution pour spécifier l'ordre dans lequel ils s'exécutent.

Une fois que le processus est achevé, le plan abstrait est transmis à l'agent de sélection.

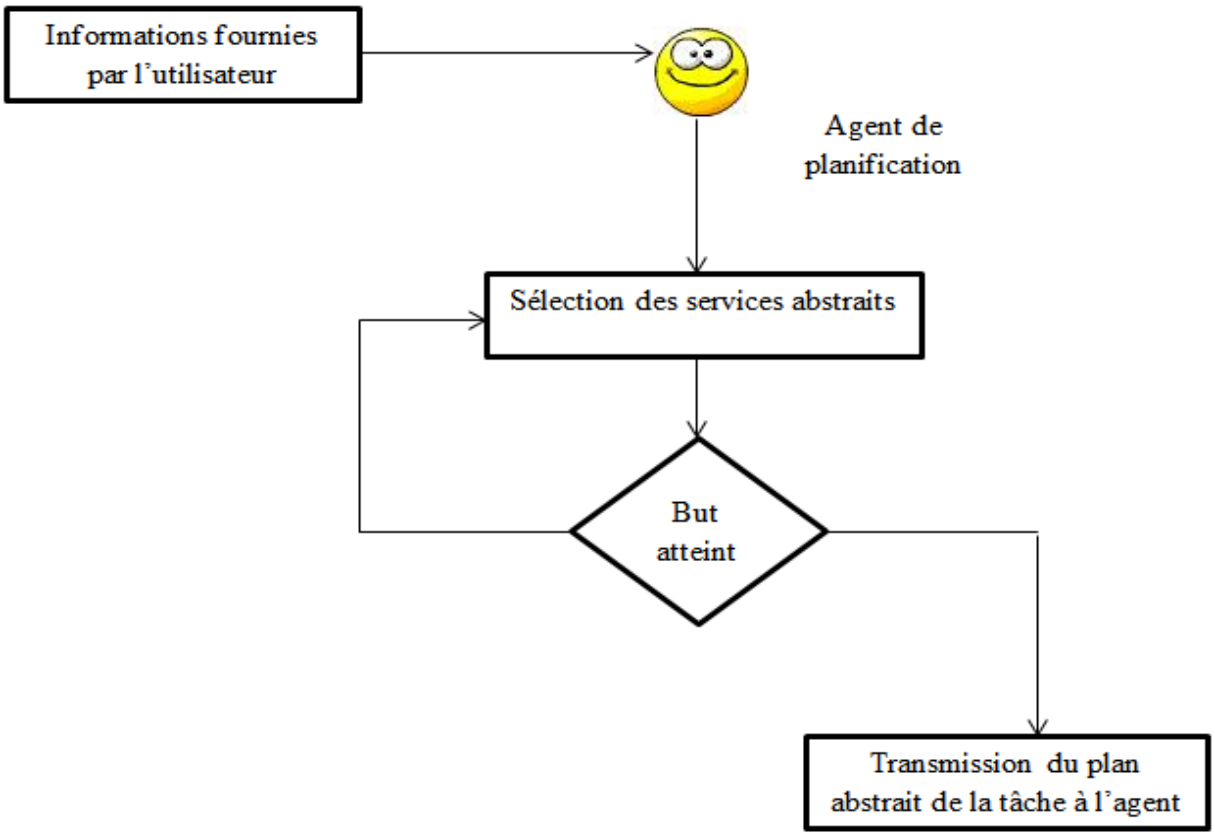

Figure 19 : Génération du plan abstrait d'une tâche.

IV.4.2.3. Sélection des services concrets et établissement du plan d'exécution

C'est l'agent de sélection qui est chargé d'effectuer cette tâche. Les services concrets sont choisis selon leur degré de correspondance au contexte d'exécution de l'application. Pour l'accomplissement de ce processus, l'agent de sélection va procéder comme suit :

- d'abord, l'agent va éliminer tous les services concrets qui ne correspondent pas au contexte de localisation de l'utilisateur. En effet, notre approche est centrée autour de l'utilisateur et de la satisfaction de ses besoins. Si cela implique qu'il ait à se déplacer, ce ne sera plus la peine.
- ensuite, il va calculer pour tous les services concrets du service abstrait le score de correspondance, ce score étant calculé selon le degré de correspondance au contexte de l'application. Les attributs de contexte ont chacun un poids ou un niveau d'importance qui a été affecté par l'utilisateur (dans ses préférences). Le service ayant le score le plus haut sera alors sélectionné.
- dès que le service concret est choisi, son identifiant et son niveau dans le plan est envoyé à l'agent d'exécution ; le niveau est utilisé lors de l'exécution en tant qu'ordre d'exécution du service.
- ce processus est réitéré jusqu'à atteindre le dernier service abstrait du plan de composition abstrait.

Nous savons que le contexte est très dynamique, cela signifie que, pendant le processus de sélection, il est possible que les données utilisées changent. Pour pallier ce problème, l'agent de sélection est abonné aux mises à jour des informations de contexte de l'application et cela, dans le but de réagir de manière réactive et immédiate aux éventuels changements de contexte.

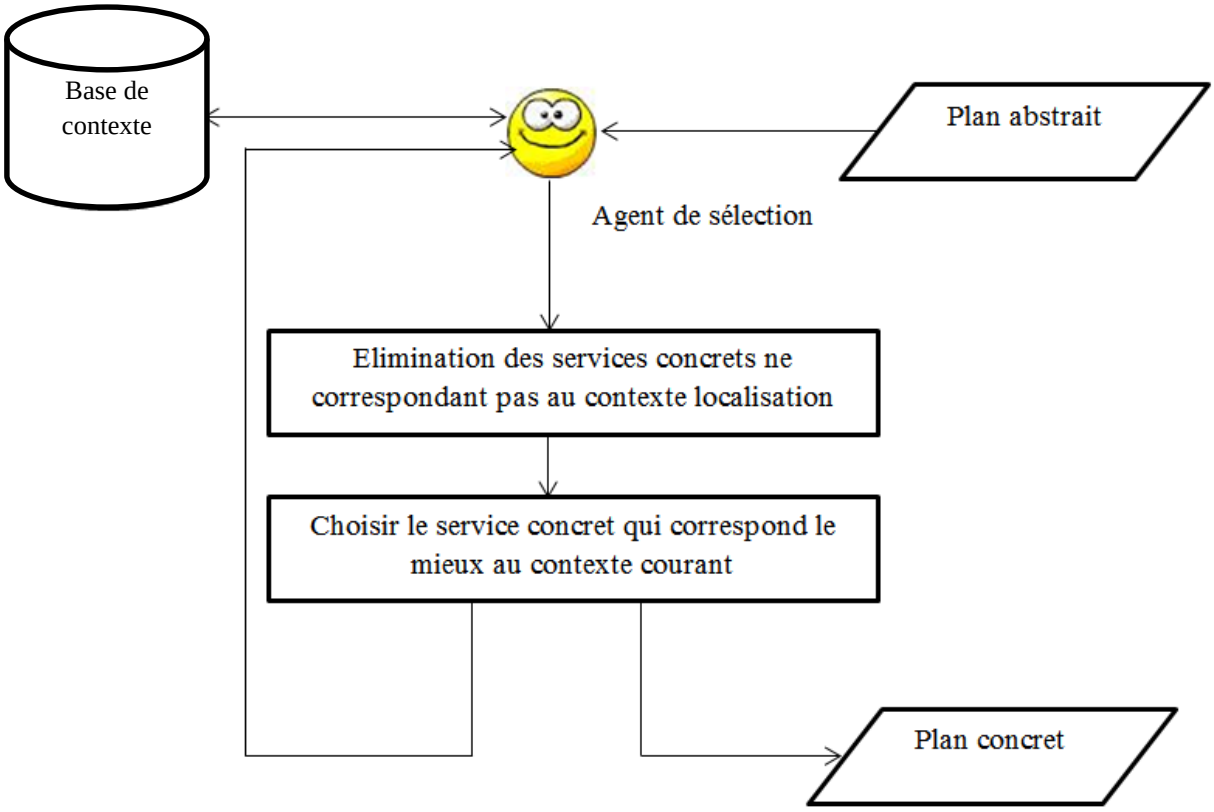

Figure 20 : Génération du plan concret d'une tâche.

IV.4.2.4. Exécution du service composite

C'est l'agent d'exécution qui est chargé de cette tâche. Il reçoit le plan d'exécution à partir de l'agent de sélection et son rôle est tout simplement de l'exécuter. Pour ce faire, il procède comme suit :à chaque fois qu'il reçoit l'identifiant d'un service et son niveau dans l'arbre d'exécution, il vérifie si tous les paramètres d'entrées dont il a besoin pour l'exécution de son plan sont contenus dans les paramètres d'entrées fournis par l'utilisateur. Si c'est le cas, il lance l'exécution de l'application et envoie un signal de terminaison aux agents planificateur et de sélection.Sinon, il envoie les paramètres restants comme nouveau but à atteindre à l'agent planificateur.Cet agent est également abonné aux éventuelles évolutions contextuelles relatives à l'application pour les mêmes raisons que l'agent de sélection.

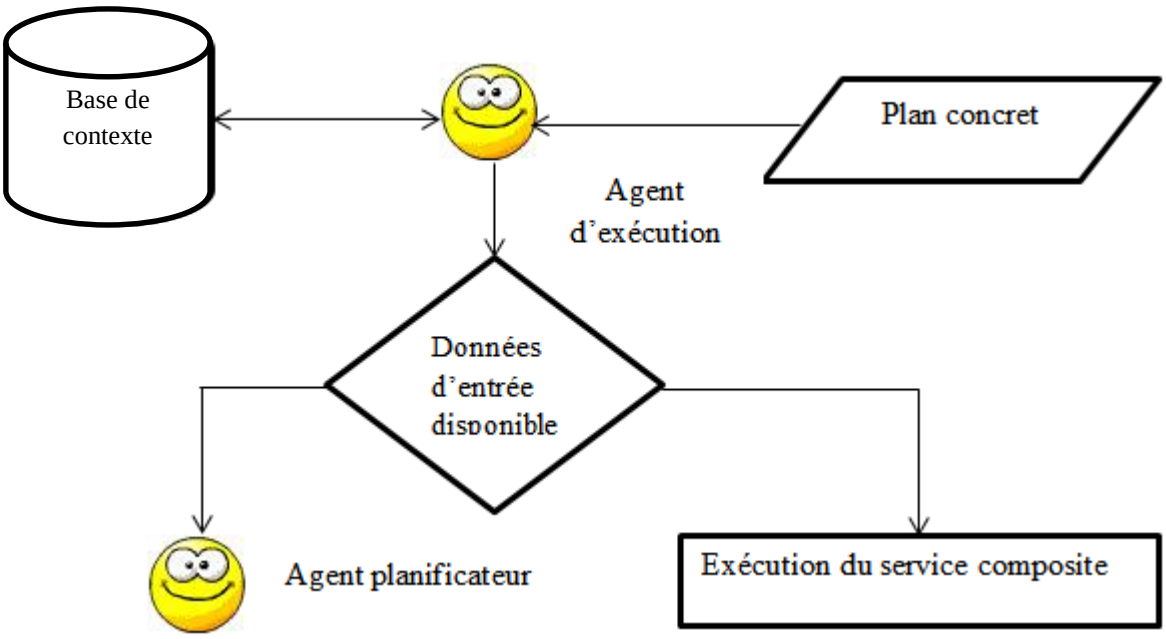

Figure 21: Exécution du service composite.

IV.4.3. Exécution automatique de certains services selon une base de règles

L'un des principaux objectifs de l'informatique ubiquitaire est la non-implication de l'utilisateur dans le processus d'exécution des applications. Nous disposons d'un historique de contexte stocké par notre Framework de gestion de contexte. Celui–ci peut donc être utilisé pour établir des règles de déclanchement de certaines applications qui sont demandées fréquemment en cas d'occurrence d'une certaine information de contexte. Cette tâche est effectuée par l'agent de scrutation. Pour la réaliser, il procède comme suit :

- l'agent de scrutation va parcourir périodiquement l'historique de contexte et vérifier les exécutions des services selon les données de contexte. Il va enregistrer le nombre de fois que l'exécution a lieu avec les données de contexte qui provoquent cette exécution. Il reçoit des notifications à chaque nouvelle demande de service, il parcourt l'historique de contexte.Si ce service a déjà été demandé avec la même situation contextuelle, on incrémente le compteur de répétition.
- à partir d'un certain seuil (défini au départ), l'agent va déclencher une règle de d'exécution automatique ayant pour entrées les valeurs des données de contexte et le résultat est l'invocation des services concernés.
- ces règles sont stockées par le Framework de gestion de contexte (avec l'historique de contexte), de sorte à ce que leur mise à jour soit plus simple.

- si le service déclenché par la règle est désactivé par l'utilisateur (un même nombre de fois), la règle provoquant son déclanchement est supprimée de la base de règles.

IV.4.4. Mécanisme de tolérance aux pannes

Nous savons que le principal problème des environnements ubiquitaires reste les pannes fréquentes qui sont dues à plusieurs raisons possibles(les défaillances matérielles par exemple) qui peuvent provoquer une indisponibilité des services, ou encore plus fréquemment un changement de contexte. En effet, le contexte étant un paramètre très dynamique, il peut changer à tout moment. Or, nous avons vu que si le contexte change, l'application peut ne plus être conforme à la demande de l'utilisateur. Pour pallier ce problème, il n'y a qu'une solution envisageable, qui est la composition dynamique de services, et cela, dans le but d'assurer la continuité du service malgré le comportement imprévisible de l'environnement.

Lorsque nous parlons de composition dynamique, nous partons du principe que la composition doit être dynamiquement réalisée en prenant en compte les changements et les cas d'échecs.

Nous nous intéressons ici aux pannes qui peuvent survenir durant les phases de sélection et d'exécution. En effet, les services concrets sélectionnés peuvent ne plus être disponibles à tout moment, ou alors ne plus correspondre au contexte courant : par exemple, le service sélectionné peut être un affichage sur l'écran de la tv alors qu'entre temps, l'utilisateur a changé de pièce.L'affichage doit donc se faire ailleurs, sinon il ne sert à rien. Afin de simplifier les choses, nous avons inclus la disponibilité d'un service à ses paramètres de contexte, de sorte que ce type de pannes soit considéré comme un changement de contexte.

Pour résoudre le problème de défaillance, nous avons décidé d'abonner les agents de sélection et d'exécution aux mises à jour du contexte de l'application.Ainsi,, s'il y a un quelconque changement, ils seront immédiatement mis au courant et pourront relancer le processus de sélection ou renvoyer une requête à l'agent de planification. De cette façon, si le service concret est défaillant ou qu'il ne correspond plus au contexte d'exécution, on peut directement en sélectionner un autre, sinon une requête est lancée pour remplacer le service abstrait qui aura échoué.

IV.5. Conclusion

L'approche que nous proposons est basée sur une coordination d'agents. Elle est simple et facile à mettre en place. Elle a l'avantage d'être distribuée, ce qui permet une utilisation réduite des équipements et cela est très utile dans un environnement ubiquitaire où la plupart des dispositifs déployés ont des capacités réduites en termes de capacités de calcul et d'énergie.

Les environnements ubiquitaires sont aussi mobiles. Les dispositifs peuvent donc quitter le système à tout moment, et certains services peuvent être indisponibles.La génération du schéma d'exécution se fait donc de manière dynamique et cela permet d'envisager les besoins au cas par cas.

L'apport principal de notre proposition repose sur les exécutions implicites des services selon les données contextuelles. En effet, le concept d'un système ubiquitaire est d'être centré autour de l'utilisateur et de ses besoins sans pour autant que celui-ci se sente écrasé par le poids de l'informatisation. Le fait de prévoir des applications lancées automatiquement selon le contexte permet de réaliser un des principaux objectifs de l'informatique ubiquitaire qui est « l'informatique enfouie» c'est-à-dire que l'utilisateur accède à un service informatisé sans même s'en rendre compte.

Dans le prochain chapitre, nous tenterons de valider notre approche de composition de services en simulant un scénario d'exécution et donnerons les résultats obtenus.

Chapitre V : Aspects techniques et mise en œuvre de l'approche

V.1. Introduction

Dans le chapitre précèdent, nous avons spécifié les concepts de base liés au fonctionnement de notre approche de composition flexible sensible au contexte. Ce chapitre est consacré à la présentation des aspects techniques liés à la mise en œuvre de notre approche ainsi qu'à son évaluation sur des scénarii d'exécution.

La première partie concerne la conception du système de composition ; on y introduit les concepts de base sur les agents et une description plus poussée du comportement des agents de l'application.

La deuxième partie concerne la mise en œuvre effective de l'approche, avec une présentation des outils utilisés. Nous y décrivons également l'environnement de simulation avec trois scénarios chacun illustrant un des aspects de l'approche.

Apres cela, nous discutons les résultats obtenus et terminons par une synthèse générale.

V.2. Conception du système de composition

Dans cette section, nous commençons par introduire les concepts de base sur les agents que nous utilisons lors de la conception de notre schéma de composition. Nous introduirons également les principes de la communication entre agents en nous concentrant sur le langage FIPA-ACL que nous utiliserons.

La deuxième partie est consacrée à une description détaillée du comportement des agents de l'application.

V.2.1. Concepts de base des agents

V.2.1.1. Les agents

Donnons tout d'abord une définition d'un agent. Un agent est, selon [81], une entitéphysique ou virtuelle, autonome, située dans un environnement, capable de le percevoir, capable d'y agir, de communiquer avec d'autres agents, et éventuellement de se reproduire. De plus, un agent possède un objectif individuel (fonction de satisfaction), des ressources, une représentation partielle de l'environnement, des compétences et il offre des services.
Son comportement dépend de ses observations, de ses connaissances, de ses croyances, de ses compétences, et de ses interactions. L'autonomie d'un agent peut se définir par trois capacités :

- les agents ont une existence propre, indépendante de l'existence des autres ;
- les agents sont capables de maintenir leur viabilité dans des environnements dynamiques sans contrôle extérieur ;
- la prise de décision interne des agents quant au comportement à avoir est uniquement dépendante des perceptions, connaissances et représentations du monde propre aux agents.

Un agent peut donc être considéré comme un programme ayant des propriétés particulières ; parmi ces propriétés, signalons celles qui sont les plus significatives, à savoir l'autonomie, la communication avec d'autres programmes par l'intermédiaire d'envois de messages et le raisonnement à partir d'une base de connaissances et d'événements extérieurs. Nous présentons uniquement ici des concepts généraux permettant de distinguer différents types d'agents qui existent dans le monde des systèmes multi-agents.

Les agents peuvent être conçus pour différents types de systèmes caractérisés par les actions qui seront attendues des agents, leur niveau de raisonnement, leur sociabilité, etc. On distingue donc des agents cognitifs, intentionnels, rationnels, réactifs et bien d'autresencore.

De plus un agent peut faire partie de l'une des trois grandes familles suivantes ; il peut êtreautonome, interagissant ou social [81].

- Un agent autonome est un agent qui raisonne et agit/réagit avec l'environnement.
- Un agent interagissant est un agent qui peut interagir avec d'autres agents.
- Un agent social est un agent dont l'action s'inscrit dans un contexte social explicite.

Notons également qu'un grand nombre de types d'agents, du point de vue de l'aspect coordination, peuvent être obtenus à partir du modèle AEIO [82] qui définit 4 facettes intégrées à l'architecture de l'agent :

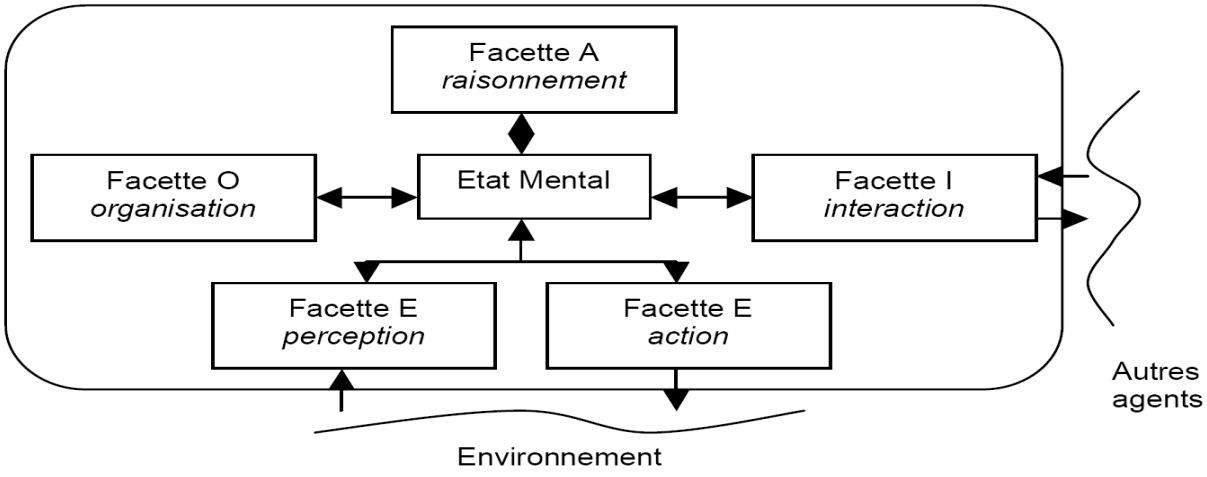

Figure 22 : Facettes d'un agent, du point de vue de la coordination.

Dans la figure ci-dessus, la facette A désigne l'ensemble des fonctionnalités liées auraisonnement de l'agent. La facette E représente les capacités de perception et d'action de l'agent sur l'environnement. La facette I symbolise les capacités d'interaction de l'agent avec les autres agents, et la facette O concerne la gestion des agents entre eux, d'un point de vue organisationnel.

V.2.1.2. La communication entre agents

La communication entre agents est possible grâce à deséchanges de messages, éventuellement englobés dans un protocole de communication quidéfinit une session de dialogue entre deux agents.

Le modèle promu par la FIPA [83] FIPA-ACL offre une base solide pour la représentation du processus de communication. FIPA-ACL est un modèle très largement répandu et succède historiquement à KQML [84], l'unique alternative existante. Plus qu'étendre ce dernier en puissance d'expression, FIPA-ACL le simplifie également. Et, de par la nature de la FIPA dont le but est de promouvoir des standards dans le monde des systèmes à agents hétérogènes et coopérants, il est largement reconnu et supporté par la communauté d'individus et d'organismes manipulant des agents. Il définit, entre autre, le format des messages échangés entre agents, la notion d'annuaire et de pages jaunes.

a) Les actes de langage

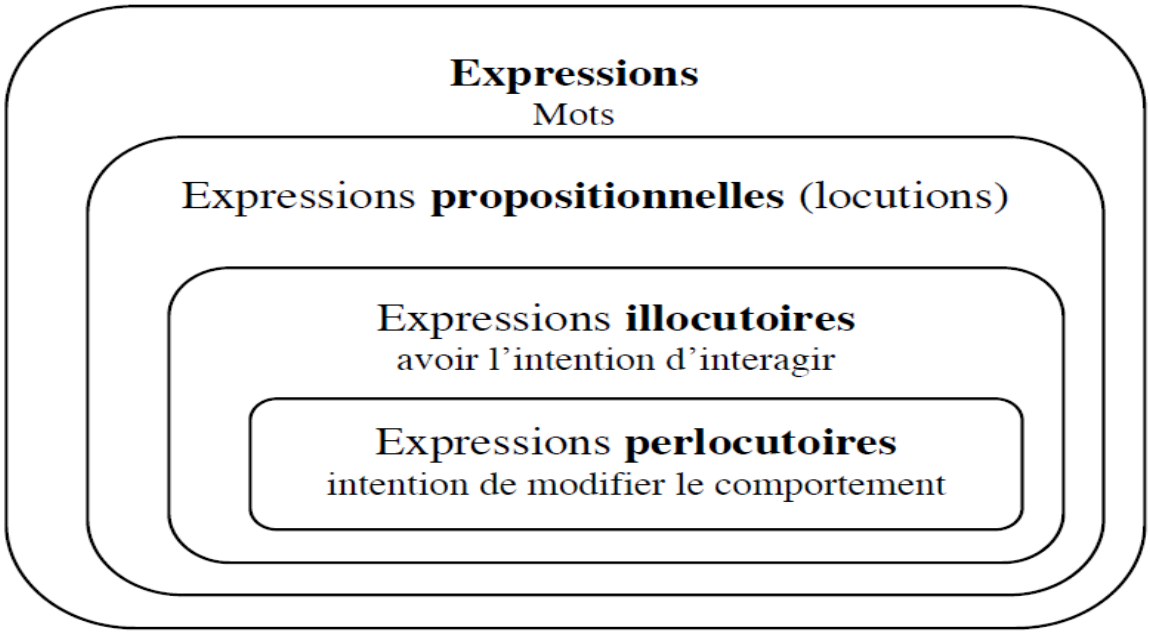

Figure 23 : Les différents niveaux de langage dans la théorie de Searle.

FIPA-ACL défini 22 actes de langage (aussi appelés performatifs), dont 4 actes primitifs qui sont *inform*, *request*, *confirm* et *disconfirm*. Avec les 18 actes composés à partir desactes primitifs, on obtient un ensemble d'actes de langage qui peuvent être découpés en 5 catégories qui traduisent l'intention d'un message. On retrouve ces catégories comme étiquettes des colonnes du tableau suivant.

	Transmission d'information	Demande d'information	Négociation	Action	Gestion d'erreurs
Accept-proposal			x		
Agree				x	
Cancel				x	
Cfp (call for proposal)			x		
Confirm	X				
Disconfirm	X				
Failure					x
Inform	X				
Inform-if	X				
Inform-ref	X				
Not-understood					x
Propagate				x	
Propose			x		
Proxy				x	
Query-if		X			
Query-ref		X			
Refuse				x	
Reject-proposal			x		
Request				x	
Request-when				x	
Request-whenever				x	
Subscribe		X			

Tableau 4 : Actes de communication du modèle FIPA-ACL, classés par catégories.

Un acte de communication est défini par une *précondition* defaisabilité FP (« *Feasibility Precondition* ») et un résultat attendu RE (« *Rational Effect* »).

Le modèle formel sur lequel se base la construction des actes de langage assure donc la stabilité de système. De plus, le modèle mental d'un agent est basé sur les trois attributs primitifs suivants : Croyance (*Belief*), Incertitude (*Uncertainty*) et But (Intention). Ils sont respectivement formalisés par les opérateurs modaux B, U et C (pour *Choice*) :

- Bi(p) signifie que i (implicitement) croit (que) p
- Ui(p) signifie que i est incertain à propos de p mais pense que p est plus probableque *not(p)*
- Ci(p) signifie que i souhaite que p soit vérifiée

On peut ainsi donner la définition de l'acte *confirm* :

```
<i, confirm (j, P) >
FP: Bi (p) and Bi (Uj (p))
RE : Bj(p)
```

FP : l'agent i croit que (p) est Vraie et i croit que Agent-j n'est pas sûr que (p) soit Vraie.

RE : l'agent j croit que (p) est Vraie.

b) Structure des messages

Un message FIPA-ACL définit un certain nombre de champs dont certains sont obligatoires et d'autres facultatifs. Les champs les plus courants sont donnés dans le tableau ci-dessous.

Champ	Catégorie d'élément
Performative	Type d'acte de communication : représente la valeur illocutoire d'un acte de langage.
Sender	Participant : Agent émetteur
Receiver	Participant : Agent récepteur
Reply-to	Participant
Content	Contenu du message
Language	Description du contenu : langage utilisé pour exprimer le contenu du message
Encoding	Description du contenu : codage du contenu.
Ontology	Description du contenu : ontologie utilisée pour l'expression du contenu du message.

Protocol	Contrôle de la conversation
Conversation-id	Identifiant de la conversation
Reply-with	Contrôle de la conversation : Demande au destinataire de répondre au présent message avec cet identifiant.
In-reply-to	Contrôle de la conversation
Reply-by	Contrôle de la conversation

Tableau 5 : Eléments d'un message FIPA-ACL.

c) Le protocole

Les agents cognitifs et communicants ont besoin d'un langage élaboré pour pouvoir échanger des messages. Toutefois, un modèle structurant uniquement le format d'un message ne suffit pas en lui-même pour pouvoir structurer des conversations entre agents. Nous avons donc besoin de la notion de protocole pour supporter des conversations valides, ce qui permet aux actes de langage de prendre tout leur sens.

Un tel protocole permet d'inclure le comportement d'un agent dans un contexte particulierauquel il doit se conformer en envoyant des messages cohérents grâce à l'utilisation d'actes de langage adaptés au discours (c'est-à-dire valides à n'importe quel moment de l'exécution du protocole) : le protocole définit les types de messages possibles à chaque instant.

FIPA-ACL supporte par défaut un certain nombre de protocoles comme, par exemple, des protocoles d'enchères, de requêtes ou de propositions. Ces protocoles sont décrits en utilisant la notation AUML qui est une extension de UML (*Unified Modeling Language*) aux agents [87].

d) Le langage de contenue

Le langage de contenu se réfère à la valeur locutoire d'un message envoyé d'un agent à un autre, c'est-à-dire au contenu « exprimé » de ce message, et à la façon de présenter ce contenu. Un bon langage de contenu doit être capable d'exprimer des expressions suffisamment riches, doit pouvoir être traité efficacement et doit bien s'adapter à l'ensemble du système au niveau du choix des technologies utilisées.

Un message FIPA-ACL, supporte par défaut les langages FIPA-SL, KIF (Knowledge Interchange Format), XML et RDF ; FIPA-SL ayant été spécifiquement défini par la FIPA.
De plus, la FIPA a défini une bibliothèque de langages pouvant être utilisés par les agents.

L'ajout d'un langage à cette bibliothèque est contraint par quelques règles de base, comme le respect d'une nomenclature, d'un développement syntaxique satisfaisant et la livraison d'exemples d'utilisation ainsi que d'une documentation claire.

Signalons que l'emploi d'un langage de contenu n'est en aucun cas recommandé par rapport à un autre. La seule contrainte pour l'emploi d'un langage de contenu, dans le modèle FIPA-ACL, concerne la gestion des agents qui impose l'utilisation du langage FIPA-SL0.

e) La découverte de services : les DF

La notion de *Directory Facilitator* [88] s'apparente à la technologie « UniversalDescription, Discovery and Integration » [20] utilisée pour l'enregistrement et la publication de services Web.

A l'instar de celui-ci, un DF permet à un agent d'enregistrer (de retirer, et de rechercher) un service en fournissant le type et le propriétaire de ce service. La description peut, et de préférence doit, contenir des détails sur le service. Il s'agit donc d'un service de type « pages jaunes ».

V.2.2. Comportement des agents de l'application

Comme nous l'avons expliqué dans le chapitre précédent, notre schéma de composition est centré autour d'une coordination entre plusieurs agents cognitifs qui s'échangent des informations pour construire le service composite.

Le choix des agents s'effectue selon leur capacité de calcul et d'énergie. L'utilisateur formule sa requête dans un dispositif.Il fera office d'agent de planification pour la première tâche. Par la suite, cet agent sélectionnera celui qui offre les meilleurs performances pour être agent de sélection et ainsi de suite jusqu'à atteindre le but. Le seul agent qui sera chargé de la tâche du début à la fin du processus est l'agent d'exécution.En conséquence, il doit être choisi de manière à assurer sa tâche du début à la fin. Pour ce faire, il doit être suffisamment puissant et disposer de la ressource énergétique nécessaire.

Dans ce qui suit, nous détaillons le comportement de chaque type d'agent de l'approche.

V.2.2.1. Agent de planification

L'agent de planification est chargé de l'élaboration des plans abstraits pour chaque tâche. Il reçoit en paramètres d'entrées le but à atteindre qui est constitué des paramètres requis par l'utilisateur ; chaque agent de planification dispose de la liste des services abstraits présents

dans l'environnement et va se servir de celle-ci ainsi que du but à atteindre pour générer le plan abstrait de la tâche en cour. La figure suivante représente l'algorithme de génération du plan :

RP : le but à atteindre ;

ASL : liste des services abstraits ;

ASP : plan des servies abstraits.

Entrée :RP, ASL

Sortie :ASP

Début

Répéter

Choisir $AS_i \in$tel que : AS_i^{out}=ArgMax(RP) ;

RP=RP\AS_i^{out}

$S_i=AS_i^{in}$

Fils(s)=fils(s)+S_i ;

Jusqu'à RP=∅

Fin.

Figure 24 : Algorithme de génération du plan abstrait.

V.2.2.2. Agent de sélection

Après que l'agent de planification élabore un plan abstrait, il transmet les identités des services abstraits choisies à l'agent de sélection, celui-ci est chargé de choisir les services concrets qui correspondent le mieux au contexte de l'application, et cela, en calculant le taux de correspondance de chacun des services concerts au contexte. Ce taux est évalué en fonction de l'importance des données contextuelles.

A chaque attribut de contexte est affecté un poids, ce poids détermine l'importance de ce paramètre du contexte pour l'utilisateur et il est affecté par ce dernier à sa première connexion au réseau. Ces données sont considérées comme les « préférences » de chaque utilisateur.De

cette façon, chacun peut accorder plus ou moins d'importance à certains paramètres. Le poids varie entre 1 et 10 ,10 étant le poids maximal qu'on peut affecter à un attribut de contexte. Le taux de correspondance d'un service concret au contexte est calculé par la formule suivante :

$$T(SC_i) = \left(\sum_{j \in cxt} P_i\right) / (10 * V_{cxt})$$

$P_j \in [1,10]$: poids de l'attribut de contexte j.

V_{cxt} : le nombre total d'attributs de contexte auxquels le service concret i est sensible.

Si nous prenons l'exemple d'une application composite sensible à quatre paramètres de contexte respectivement A, B, C. et D,trois services concrets prennent en charge les fonctionnalités requises par l'utilisateur. Les poids sont affectés comme suit : $P_A=3$, $P_B=2$, $P_C=9$ et $P_D=5$. Trois services concrets correspondent au service abstrait concerné ; Le premier service concret correspond aux paramètres contextuels A, B et D ; Le deuxième correspond aux paramètres A et C ; Enfin, le troisième correspond aux paramètres B et C. Les taux de correspondance de chaque service concret sont alors : $T(CS_1)=(10/40)=0.25$, $T(CS_2)=(12/40)=0.37$ et $T(CS_3)=(7/40)=0.17$. Le service concret sélectionné sera donc CS_2.

L'algorithme décrivant le comportement de l'agent de sélection est représenté dans la figure ci-dessous :

Entrée : AS_i

Sortie : CS

Début

Pour chaque CS_i dans AS_i

Calculer $T(CS_i)$

Finpour

$CS=ArgMax(T(CS_i))$

Fin.

Figure 25 : Algorithme de sélection du meilleur service concret.

V.2.2.3. Agent d'exécution

L'agent reçoit au fur et à mesure les services concrets sélectionnés par l'agent de sélection.Il l'ajoute à la pile d'exécution du service composite et compare les paramètres nécessaires à l'exécution à ceux qui sont disponibles. Si tous les paramètres nécessaires sont disponibles, alors l'exécution est lancée.Sinon, une nouvelle requête est transmise à l'agent de planification.

Pour évaluer la correspondance entre les concepts disponibles et ceux nécessaires, nous utilisons l'algorithme de matching proposé par Kouicem [80]. Cet algorithme permet de faire correspondre deux concepts. Nous n'allons utiliser que la fonction numérique « subsomption » pour évaluer le taux de correspondance entre les entrées nécessaire à l'exécution du service composite et ceux disponible pour l'application. Il y a quatre niveaux de correspondance entre deux concepts d'ontologie. Nous nous intéressons aux niveaux exacts et plug_in. Le premier signifie que les deux concepts sont exactement les mêmes et le deuxième que celui disponible subsume le nécessaire.

- Exact si $R^{in} = CS^{in}$ où CS^{in} est une sous classe direct de R^{in}.
- Plug_in Si R^{in} subsume CS^{in} c'est à dire que R^{in} peut être utilisé à la place de CS^{in}.

La fonction numérique Subsomption(x,y) retourne 0 si il n'y a aucune relation entre x et y, 1 si x subsume y et 2 si les deux concepts sont les mêmes. Le comportement de l'agent d'exécution devient alors :

GP : paramètres d'entrées disponibles ;

EP : plan à exécuter.

Entrée :GP, CS_i

Sortie :EP

Début

Si (Subsomption (GP,CS^{in})=1) **ou** (Subsomption(GP,CS^{in})=2)**Alors**

Empiler SC et aller a '1'

Sinon envoyer une requete à l'agent de planification avec

RP=CS^{in}\GP

'1' :

Tanque pile non vide

Dépiler et invoquer CS_i

FinTanque ;

Fin.

Figure 26 : Algorithme de fonctionnement de l'agent d'exécution.

V.2.2.4. Agent de scrutation

Cet agent est en marge de l'application.Son comportement est indépendant du schéma de composition. Son rôle est simple : il stocke la base de règles et il est abonné au Framework de gestion de contexte. A chaque changement de contexte, il vérifie si cette situation contextuelle provoque l'exécution d'une des règles.

D'un autre côté, cet agent peut également générer de nouvelles règles. Il garde un historique des requêtes des utilisateurs selon une certaine situation contextuelle.Au bout d'un certain nombre de fois, une nouvelle règle est générée.

Déclencher une règle signifie pour l'agent de scrutation « envoyer une requête de composition à l'agent de planification ».

V.3. Mise en œuvre de l'approche

V.3.1. Framework de gestion de contexte

Pour mettre en œuvre notre approche, nous avons besoin d'un Framework sous-jacent de gestion de contexte qui se charge de récupérer les données, les modéliser, les stocker et les transmettre aux agents de l'application. Afin de pouvoir communiquer avec notre application, il doit le faire en format de messages ACL. Nous supposons alors que nous disposons d'une plate-forme multi-agent de gestion du contexte du type de celle proposé par Sebaak [89] avec un agent chargé de transmettre les données de contexte aux agents impliqués dans notre approche.

Nous ne disposons pas des moyens matériels pour mettre en œuvre cela.Donc, nous nous contenterons de passer les données de contexte à travers un agent de contexte qui va les récupérer directement à partir d'un base de connaissance que nous définirions au préalable.

V.3.2. JADE

JADE (Java Agent DEvelopement framework) est une plate-forme multi-agent créé par le laboratoire TILAB et décrite par Bellifemine et al. JADE permet le développement de systèmes multi-agents et d'applications conformes aux normes FIPA. Elle est implémentée en JAVA et fourni des classes qui implémentent « JESS » pour la définition du comportement des agents. JADE possède trois modules principaux :

- **Directory Facilitor « DF »** (facilitateur d'annuaire) : c'est l'agent fournissant un service de pages jaunes à la plateforme. Grâce à ce service, un agent peut connaître les agents capables de lui fournir les services qu'il requiert pour atteindre son but.
  L'interface du DF peut être lancée à partir du menu du RMA, cette action est en fait implémentée par l'envoi d'un message ACL au DF lui demandant de charger son interface graphique. L'interface peut être juste vue sur l'hôte où la plateforme est exécutée. En utilisant cette interface, l'utilisateur peut interagir avec le DF : voirla description des agents enregistrés, ajouter ou supprimer des agents, modifier la description sur les agents enregistrés ou chercher la description d'un agent.
- **ACC «Agent Communication Channel »** (canal de communication) : c'est le composant logiciel qui contrôle tous les échanges de messages dans la plateforme incluant également les messages de/vers des plateformes distantes.

- **AMS « Agent Management System »** supervise l'enregistrement des agents, leur authentification, leur accès et l'utilisation du système.

Ces trois modules sont activés à chaque démarrage de la plate-forme.

V.3.2.1. Architecture logicielle de la plate-forme JADE

Dans la plate-forme JADE, deux méthodes sont fournies par la classe 'Agent' afin d'obtenir l'identifiant de l'agent DF par défaut et celui de l'agent AMS : getDefaultDF() et respectivement getAMS(). Ces deux agents permettent de maintenir une liste des services et des adresses de tous les autres agents de la plate-forme. Le service DF propose quatre méthodes afin de pouvoir :

- enregistrer un agent dans les pages jaunes (register).
- supprimer un agent des pages jaunes (deregister).
- modifier le nom d'un service fourni par un agent (modify).
- rechercher un service (search).

Le service AMS s'utilise généralement de manière transparente (chaque agent créé est automatiquement enregistré auprès de l'AMS et se voit attribué une adresse unique). Ces deux services fournissent donc les annuaires qui permettent à n'importe quel agent de trouver un service ou un autre agent de la plate-forme.

V.3.2.2. Comportement des agents dans la plate-forme JADE

Un agent doit être capable de gérer plusieurs tâches de manière concurrente en réponse à différents évènements extérieurs.

Afin de rendre efficace cette gestion, chaque agent de JADE est composé d'un seul thread et chaque comportement qui le compose est en fait un objet de type Behaviour. Des agents multithreads peuvent être créés mais il n'existe actuellement aucun support fourni par laplate-forme.

Pour implémenter un comportement, le développeur doit définir un ou plusieurs objets de la classe Behaviour, les instancier et les ajouter à la file des tâches « ready »de l'agent. Il est à noter qu'il est possible d'ajouter des comportements et sous-comportements à un agent ailleurs que dans la méthode setup ().

La plate-forme JADE fournit sous forme de classes un ensemble de comportements ainsi que des sous-comportements prêt à l'emploi. Elle peut les exécuter selon un schéma prédéfini, par exemple la classe SequentialBehaviour est supportée et exécute des sous-comportements de manière séquentielle. Toutes les classes prédéfinies dans JADE héritent de la classe Abstraite Behaviour. On peut les citer :

- Classe SimpleBehaviour (abstraite): modélise un comportement simple. Sa méthode reset() n'effectue aucune opération.
- Classe CompositeBehaviour (abstraite) : modélise un comportement composé. Les actions effectuées par cette classe sont définies dans les comportements enfants.
- Classe FSMBehaviour : Cette classe hérite de CompositeBehaviour et exécute des comportements enfants suivant un automate à états finis défini par l'utilisateur. Lorsqu'un comportement enfant se termine, sa valeur de fin retournée par la fonction onEnd() indique le prochain état à atteindre. Le comportement correspondant à cet état sera exécuté à la prochaine exécution de la classe. Elle se termine lorsque qu'un comportement associé à un état final a été exécuté.
- Classe SenderBehaviour : elle étend la classe OneShotBehaviour et encapsule une unité atomique qui effectue une opération d'envoi de message.
- Classe ReceiverBehaviour : elle encapsule une unité atomique qui effectue une opération de réception de message. Ce comportement s'arrête dès qu'un message a été reçu. S'il n'y a pas de message dans la file d'attente de l'agent ou que le message ne correspond pas au MessageTemplate du constructeur de cette classe, alors il se met en attente.
- On peut aussi citer d'autres classes par exemples : Classe WakerBehaviour (abstraite), ParrallelBehaviour, SequentialBehaviour, CyclicBehaviour. (abstraite), OneShotBehaviour (abstraite).

V.3.3. Description de l'environnement de simulation

Afin de démontrer la validité de notre approche, nous allons l'appliquer à des scénarii envisageables dans le monde réel. Notre choix s'est directement porté sur la domotique et l'aide à la personne.

Notre environnement est donc une maison intelligente où sont disséminés plusieurs dispositifs. Chaque dispositif offre un certain nombre de services aux utilisateurs qui sont en

l'occurrence les habitants de la maison. Chaque utilisateur dispose d'une étiquette RFID qui permet de le localiser, Ce dispositif contient également les préférences de l'utilisateur, ces dernières étant utilisées lors du processus de sélection des services concrets ainsi que certaines règles d'invocation de services pour l'utilisateur.

Dans ce qui suit, nous allons brièvement décrire les différents dispositifs disponibles dans l'environnement ainsi que les services qu'ils offrent. Apres cela, nous décrirons trois scénarios, le premier vise à expliquer le fonctionnement classique de notre approche, le deuxième illustre la manière dont fonctionne notre mécanisme de tolérance aux pannes et le dernier vise à détailler le fonctionnement de l'agent de scrutation dans le déclanchement des règles selon les situations de contexte ainsi que la génération de nouvelles règles.

V.3.3.1. Les dispositifs disponibles

Une maison « intelligente » doit contenir un certain nombre d'équipements électroniques déployés dans chaque pièce. Parmi ces équipements :

- des robots qui sont les dispositifs principaux de la simulation : ils peuvent se déplacer dans l'environnement et déplacer des objets, ils agissent directement sur l'environnement et peuvent le modifier. Nous partons du principe qu'ils sont équipés d'un bras mobile qui permet de prendre des objets, ouvrir des portes, fermer des fenêtres…etc.
- des RFIDs qui permettent de localiser les utilisateurs et de récupérer leurs profils afin d'adapter l'offre de services à leurs préférences.
- différents types de capteurs déployés dans toutes les pièces de la maison. Des capteurs de présence pour localiser l'arrivée ou le départ des utilisateurs, des capteurs de bruit pour évaluer le niveau sonore, des capteurs de température…etc.
- des caméras sur les robots et dans des endroits stratégiques de la maison (en face des portes, et aux différents accès de la maison) pour localiser les éventuels intrus.

Chacun de ces équipements offre un certain nombre de services.Dans ce qui suit, nous donnons la liste des services disponibles.

V.3.3.2. les services disponibles

Nous considérons que chaque dispositif offre un certain nombre de fonctionnalités qui seront nécessaires pour répondre aux requêtes des utilisateurs parmi les services que nous considérons dans notre environnement de simulation :

- ✓ service de déplacement du robot : pour pouvoir offrir ses services aux utilisateurs dans toute la maison, il doit être en mesure de se déplacer vers des coordonnées précises. Il peut également bouger à gauche à droite en avant ou en arrière.
- ✓ service de prise d'un objet par le robot : le bras du robot sert entre autre à prendre des objets pour pouvoir les déplacer.Il reçoit en paramètre la position de l'objet et le robot actionne son bras pour prendre l'objet en question.
- ✓ donner un objet a l'utilisateur : une fois que le robot aura l'objet en main, il pourra le donner à l'utilisateur.Il aura en paramètres les coordonnées de l'utilisateur.Il en résulte que le bras est tendu vers la position de l'utilisateur.
- ✓ service localisation du robot : ce service permet d'avoir l'emplacement exact du robot.Ce service peut être utile dans le cas où il y a le choix du robot : autant choisir celui qui est le plus prêt. Un autre cas ou ce service peut être utilisé est lorsque l'utilisateur souhaite savoir où sont tous ses équipements.
- ✓ sélection des mouvements des utilisateurs : permet de repérer les changements de positions des utilisateurs dans l'environnement grâce entre autres aux caméras, capteurs de présences. C'est surtout utile pour les exécutions automatiques de certains services. Par exemple, si on a une règle « allumer la lumière si un utilisateur arrive dans cette pièce » dès qu'il sera détecté là, l'agent de scrutation va exécuter la règle.
- ✓ service de mouvement de la caméra :elle peut pivoter dans tous les sens et cela permet de récupérer des images de tout l'environnement et évaluer les éventuels changements et ces images peuvent être utilisées pour déduire le contexte courant.
- ✓ détection des voix grâce aux capteurs : il est possible d'évaluer le volume dans une pièce et ainsi déduire si une activité particulière est en cours, pour adapter la luminosité et la température à l'activité en question.

- ✓ détection des utilisateurs via les RFIDs : les utilisateurs sont équipés d'étiquettes grâce auxquels ils sont identifiés par les RFIDs lorsqu'ils sont à leurs portée. Cela permet d'identifier l'utilisateur à son arrivée et de récupérer son profil. Ainsi, s'il demande un service, il sera adapté à ses préférences

V.3.4. Scénario 1 (cas normal)

L'objectif de ce premier scénario est d'illustrer le fonctionnement classique de notre approche. Il met en évidence l'interaction et la coordination entre les différents agents du système.

Il s'agit de mettre en scène un ensemble d'équipements dans un environnement domotique, et montrer comment l'approche réagit à une requête de l'utilisateur et la manière dont elle utilise les informations contextuelles pour adapter la réponse à la situation contextuelle.

Nous supposons que notre environnement est une maison truffée d'équipements informatiques offrants chacun un certain nombre de services.

Des capteurs de mouvement et de présence sont disposés un peu partout dans chaque pièce de la maison ; il y a un robot muni d'une caméra, d'un bras qui peut bouger et prendre des objets. L'utilisateur est muni d'une étiquette RFID qui peut être utilisée pour le localiser.

- **Besoin de l'utilisateur**

L'utilisateur « Homer » qui se trouve dans le salon souhaiterait manger des donuts. Il demande donc au robot de lui apporter le plat qui est dans la cuisine.

- **Description du scénario d'exécution**

Vu que c'est le robot qui a reçu la requête de l'utilisateur, il sera l'agent de planification de la première tâche.Il parcourt la liste des services abstraits avec le but à atteindre « avoir le plat de donuts ». Le robot va donc se déplacer vers la cuisine, prendre le plat de donuts, retourner vers Homer et lui donner le plat de donuts.

Le premier service abstrait choisit par l'agent de planification est « avoir le plat de donuts ».Il le transmet à l'agent de sélection qui va choisir le meilleur service concret selon le contexte.Le résultat sera « donner le plat à Homer », ceci est transmis à l'agent d'exécution, il l'empile et vérifie si toutes les entrées sont disponibles. Si ce n'est pas le cas, il envoie une

requête à l'agent de sélection (le dispositif n'est pas forcément le robot mais c'est selon la capacité à assumer la charge) avec le but à atteindre « être avec l'utilisateur ».

La nouvelle requête est donc « être avec l'utilisateur » .L'agent de planification va choisir le service abstrait « déplacement du robot vers la position de l'utilisateur ».Il transmet cela à l'agent de sélection qui va sélectionner « aller au salon »,Il transmet cela à l'agent d'exécution. Ce plan est empilé dans la pile de l'application et l'agent d'exécution vérifie la correspondance des entrées nécessaires avec les entrés disponibles.Le nouveau but à atteindre est « avoir le plat de donuts ».

L'agent de planification va choisir le service abstrait « récupérer l'objet ».L'agent de sélection choisira le service concret « actionner le bras du robot et lui faire prendre le plat de donuts ».L'agent d'exécution empile et transmet la nouvelle requête à l'agent de planification qui est « rejoindre l'objet ».

Enfin, l'agent de planification va choisir le service « aller vers le plat ».L'agent de sélection va choisir le service « déplacer le robot vers la position cuisine » et l'agent d'exécution empile et vérifie si toutes les entrées sont disponibles.Si c'est le cas, il déclenche l'exécution des services en dépilant la pile d'exécution de l'application et en invoquant les services au fur et à mesure.

Le schéma suivant montre les différentes interactions qui ont eu lieu entre les agents lors de la construction du plan d'exécution. L'ordre d'interaction est représenté par les numéros des messages :

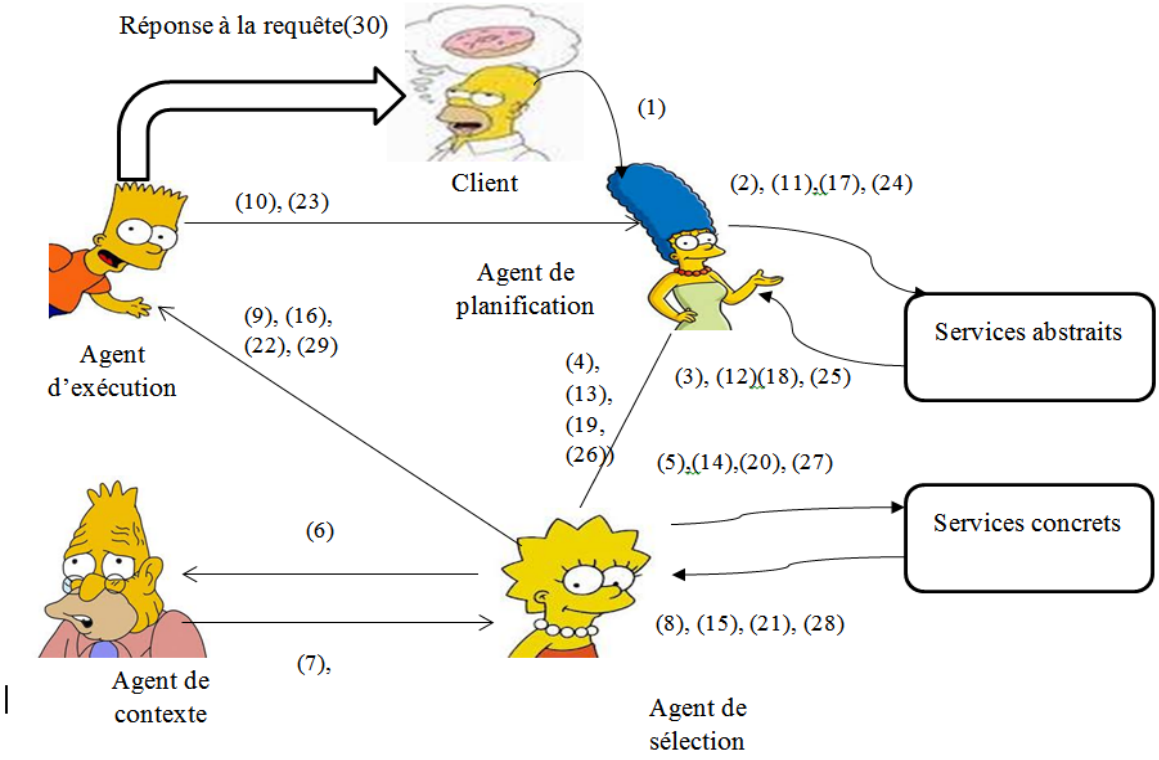

Figure 27 : Interactions entre les agents durant l'exécution du premier scénario.

- **Analyse**

Ce scénario démontre la capacité de l'approche proposée à composer plusieurs services en faisant interagir plusieurs agents et ce, pour satisfaire l'utilisateur. Cette composition étant entièrement centré autour de la correspondance au contexte, la réponse est plus adaptée aux conditions d'exécution du service composite. Enfin, le plan d'exécution est généré dynamiquement sans s'appuyer ni sur des règles prédéfinis ni sur des Workflows préenregistrés dans une base de données.

Les agents communiquent en s'échangeant des messages, la figure 28 représente le schéma de communication et d'interaction entre les agents sniffés par le sniffer de la plateforme JADE.

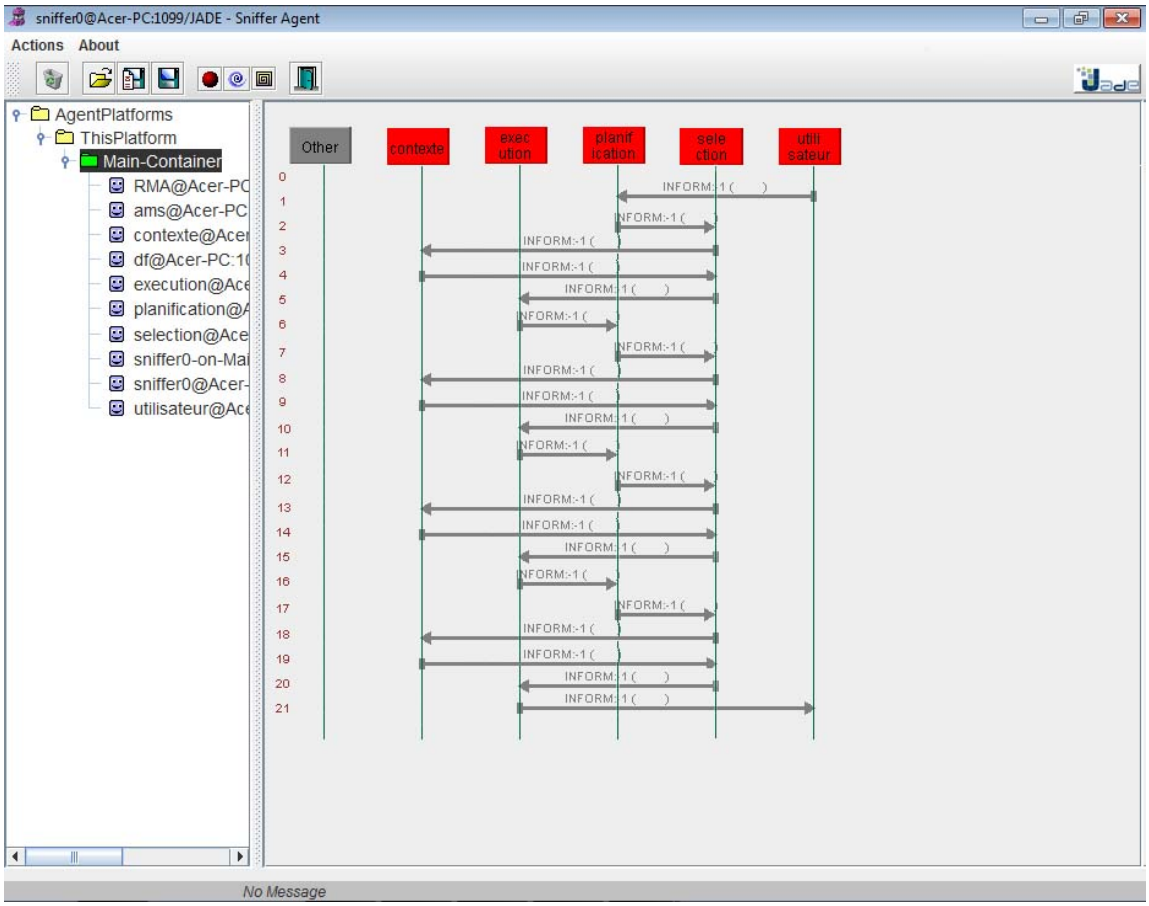

Figure 28 : Schéma d'interaction entre les agents durant le déroulement du scénario 1 .

V.3.5. Scénario 2 (illustrer la tolérance aux pannes)

L'objectif de ce scénario est de mettre en lumière la manière dont notre approche gère les situations de pannes. Comme nous l'avons dit dans le chapitre précèdent, nous nous intéressons aux pannes qui peuvent survenir au niveau de la sélection du service concret et de l'exécution de l'application. Pour détecter les pannes, nous utilisons les changements de contexte comme référence. Si un service n'est plus disponible, son attribut « disponibilité » devient « non » ; si un dispositif est presque en panne d'énergie, son attribut « capacité » baissera...etc.

On suppose ici qu'au lieu d'avoir un seul robot, on en a deux. Le premier est au salon et c'est lui qui reçoit la requête de l'utilisateur, le deuxième est dans la cuisine.

- **Besoin de l'utilisateur**

La requête reste la même que dans le scénario précédent, c'est-à-dire que l'utilisateur demande à avoir le plat de donuts qui est dans la cuisine.

- **Description du scénario d'exécution**

Au début, le premier se charge de la requête mais, pendant le processus, son niveau d'énergie devient très faible. Alors au lieu de sélectionner le service concret « déplacer robot 1 vers la cuisine », il va choisir le deuxième qui est déjà dans la cuisine.

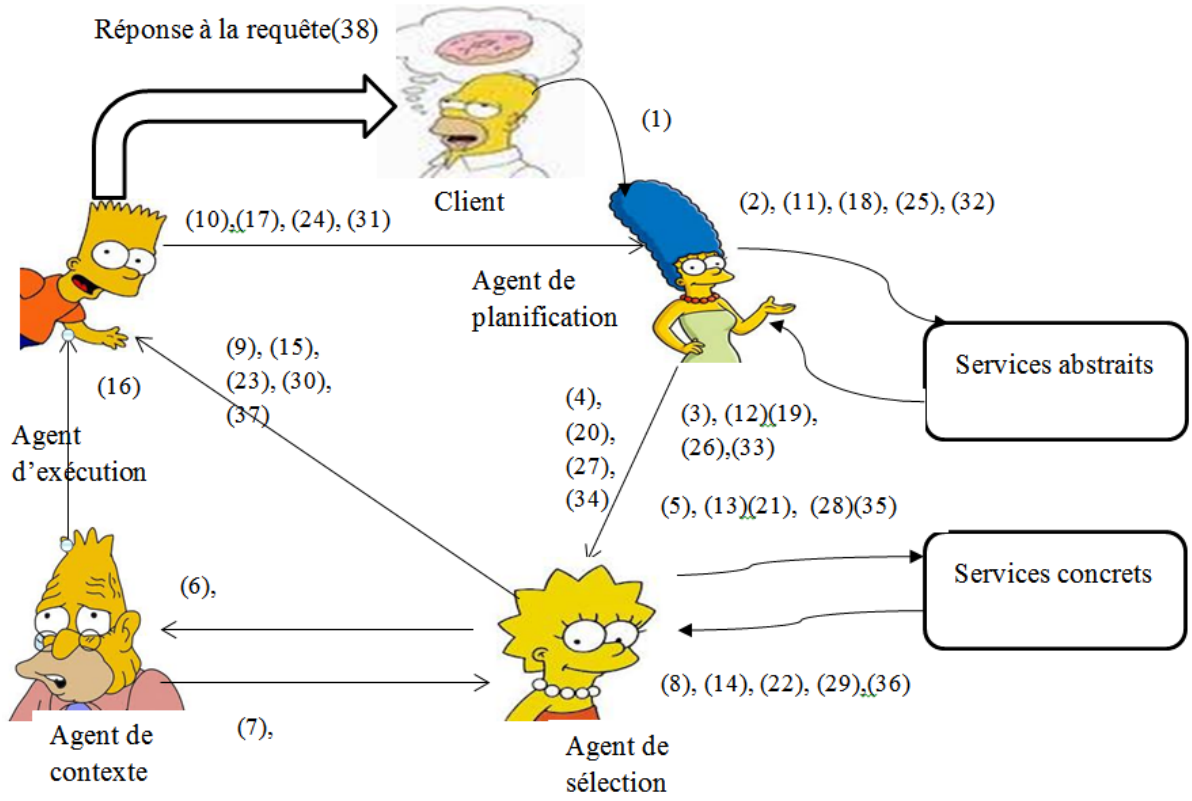

Figure 29 : Interactions entre les agents durant l'exécution du deuxième scénario.

- **Analyse**

A travers cet exemple, on constate que l'abonnement des agents aux misesàjour du contexte de l'application est très utile : d'une part, pour la réactivité du système (cela permet de réagir immédiatement aux changements qui peuvent survenir durant la construction du schéma de composition ou encore de l'exécution de l'application). D'autre part, cela évite des opérations inutiles de parcours de la base de données de contexte à chaque opération de sélection de service.

La figure 30 représente le schéma de communication et d'interaction entre les agents sniffés par le sniffer de la plateforme JADE.

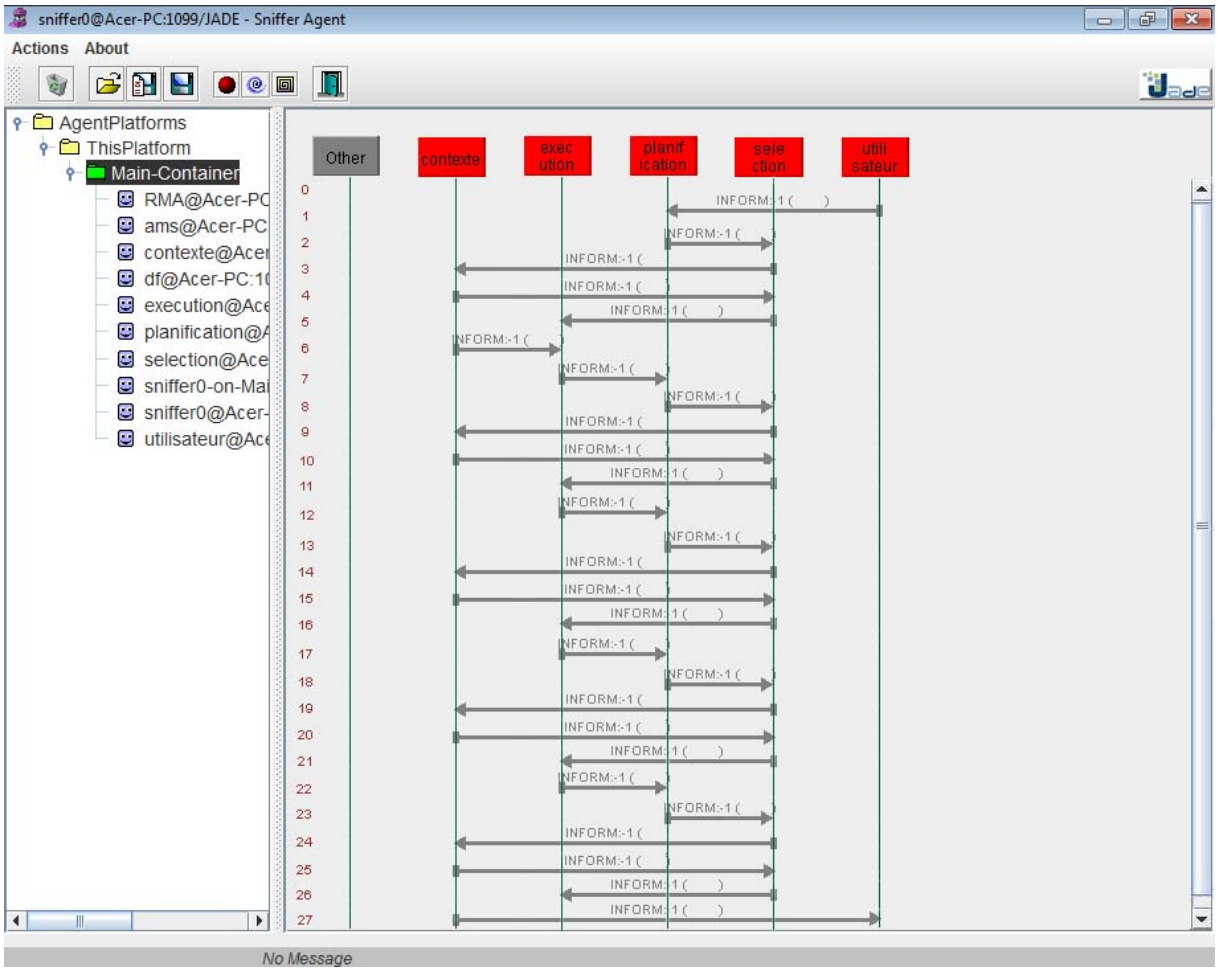

Figure 30 : Schéma d'interaction entre les agents durant le déroulement du scénario 2.

V.3.6. Scénario 3 (illustrer le fonctionnement de l'agent de scrutation)

Ce dernier exemple permet de voir le fonctionnement de l'agent de scrutation, cet agent a en réalité deux rôles : c'est lui qui est chargé d'émettre les requêtes d'exécution automatiques qui sont le résultat de déclanchement de règles ; il est aussi chargé de générer de nouvelles règles et de supprimer des règles inutiles.

Des règles prédéfinies sont stockées parmi les préférences des utilisateurs. On suppose qu'à chaque fois que l'utilisateur s'installe sur le canapé du salon, c'est pour regarder la TV.

- **Description du scénario d'exécution**

Quand la position de l'utilisateur devient « sur le canapé », l'agent de scrutation reçoit une notification de la base de contexte indiquant que la règle :

« Utilisateur 'Homer 'sur le canapé→éteindre la lumière du salon, donner la télécommande de la tv à l'utilisateur » doit être déclenchée. L'agent de scrutation va donc vérifier que toutes les entrées sont disponibles.Il peut actionner directement l'extinction de la lumière mais pas la récupération de la télécommande. Il envoie alors une requête à l'agent de planification qui sera : « donner la télécommande a Homer » et la construction du schéma de composition peut démarrer. La figure suivante montre le fonctionnement standard de l'agent de scrutation :

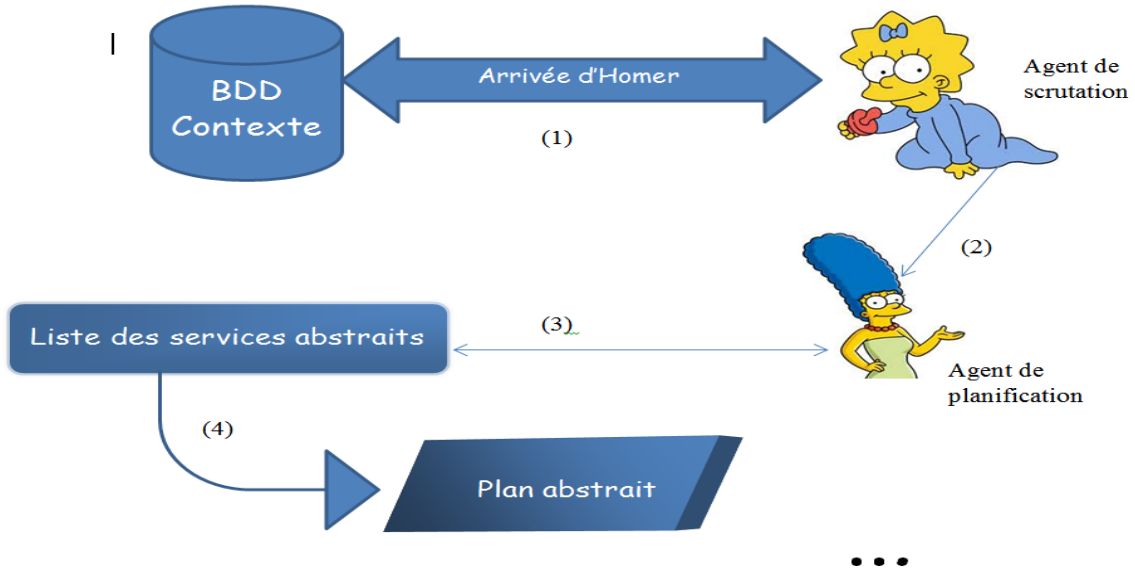

Figure 31 : Fonctionnement standard de l'agent de scrutation.

- **Analyse**

L'apport principal de notre approche est justement cet agent car le concept fondamental de l'informatique ubiquitaire est d'offrir à l'utilisateur l'accès à des services informatiques sans qu'il ait à les demander et parfois même sans qu'il en soit conscient. Et le fait de profiter de l'historique des exécutions pour pouvoir générer des règles d'exécution permet de rendre le système intelligent car il peut en quelque sorte raisonner sur les préférences et les habitudes de ses utilisateurs, de cette façon il offre un service personnalisé à chacun.

La figure 32 représente le schéma de communication et d'interaction entre les agents sniffés par le sniffer de la plateforme JADE.

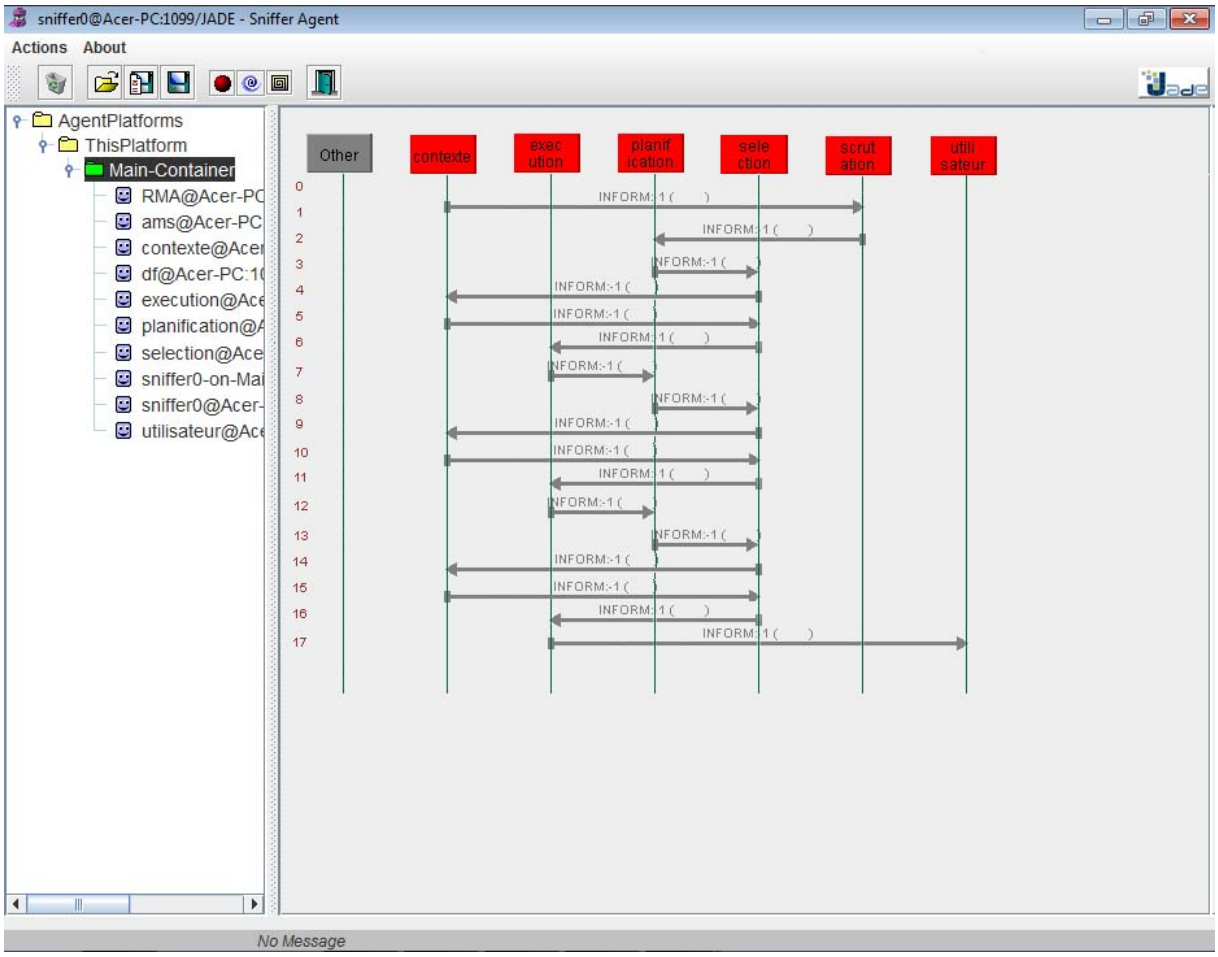

Figure 32 : Schéma d'interaction entre les agents durant le déroulement du scénario 3.

V.4. Synthèse

Dans ce chapitre nous avons présenté la conception technique de l'approche de composition flexible sensible au contexte. Nous avons d'abord commencé par décrire les aspects techniques relatifs à sa mise en œuvre. Puis, nous avons défini l'environnement de simulation avec ses équipements et les fonctionnalités qu'ils offrent. Enfin, nous avons détaillé la manière dont fonctionne l'approche en l'appliquant à trois scénarii, chacun mettant en avant l'un des aspects principaux de l'approche qui sont la dynamique, la disponibilité et l'exécution automatique des services grâce à l'agent de scrutation.

Conclusion générale et perspectives.

Conclusion générale et perspectives

Le travail de recherche présenté dans ce mémoire fait l'objet d'une contribution qui concerne la proposition d'une approche de composition dynamique de services sensible au contexte basée sur les systèmes multi-agents, ce qui garantit une décentralisation complète. Cette approche facilite ainsi le développement et l'intégration de nouveaux services à l'environnement sans avoir à réinitialiser le système.

La gestion du contexte est confiée à un Framework dédié à cela, de la capture à la dissémination en passant par la modélisation et le stockage. Le fait que les agents de composition soient indépendants vis-à-vis du Framework de gestion de contexte permet une plus grande flexibilité du processus de construction des workflows de composition, dans le sens où le contexte devient un paramètre à prendre en compte lors du processus de composition et cela fait que des plans générés sont plus souples et plus adaptables selon les paramètres du contexte pris en compte.

Enfin, la génération automatique de plans de composition selon une base de règles qui est elle-même dynamique rend l'approche encore plus flexible et auto-adaptative à l'environnement tout en limitant au maximum les interventions humaines.

Ce travail fut pour nous très bénéfique.Il nous a permis d'appréhender de nouveaux aspects de l'informatique, notamment dans le domaine de la recherche et de l'innovation. Grâce à ce mémoire, nous avons abordé l'informatique ubiquitaire sous un angle intéressant qui est celui des réseaux domestiques et plus particulièrement de l'aide à la personne. La domotique est un domaine de recherche très populaire de nos jours. Il fait partie d'un secteur économique stratégique qui est celui des maisons intelligentes. D'ailleurs, plusieurs multinationales spécialisées dans les nouvelles technologies se sont lancées dans le concept des « smart homes ». Cela indique que l'informatique ubiquitaire et plus particulièrement les maisons intelligentes vont encore connaitre un grand intérêt de la part des chercheurs et des entreprises durant les prochaines années.

De ce travail, nous avons pu dégager quelques perspectives:

- d’abord, il faut simuler notre approche dans un vrai simulateur de sorte à l’évaluer d’une manière formelle, en introduisant des situations de pannes afin de tester si le mécanisme de tolérance aux pannes est en mesure de gérer différents types de situations d’échecs.
- ensuite, il faudrait comparer les performances de l’approche à d’autres approches de composition pour l’évaluer par rapport à ce qui se fait dans le domaine.

Pour ce qui est du moyen et long terme, nous pouvons envisager de :

- Sécuriser l’approche, avec des mécanismes de contrôle d’accès avec limitation des droits selon les utilisateurs ou encore en utilisant la cryptographie sur les étiquettes RFID.
- Finalement, nous pouvons envisager une optimisation du fonctionnement de l’agent de scrutation par l’utilisation de techniques du datamining, cela permettrait de booster les résultats que fournira cet agent et la base de règles résultera d’algorithmes plus poussés.

Bibliographie

Bibliographie

[1] Mark Weiser, "The computer for the 21st century", Scientific American, 265(3):66-75, September 1991

[2] Mahadev Satyanarayanan, IEEE Personal Communications, Volume 8, Issue 4,pp. 10-17, August 2001

[3] Sumi Helal, "Standards for service discovery and delivery", IEEE Pervasive Computing Journal, Volume 1, Issue 3, pp. 95-100, July-Sept. 2002

[4] Dimitar Valtchev, Ivailo Frankov, "Service gateway architecture for a smart home", IEEE Communications Magazine, Volume 40, Issue 4, pp. 126-132, April 2002

[5] Michael P. Papazoglou, "Service-oriented computing: concepts, characteristics and directions", 4th International Conference on Web Information Systems Engineering (WISE'03), Rome, Italy December 2003

[6] André Bottaro, Eric Simon, Stéphane Seyvoz, Anne Gérodolle, "Dynamic Web Services on a Home Service Platform", 22nd IEEE International Conference on Advanced Information Networking and Applications (AINA-08), Ginowan, Okinawa, Japan, March 2008

[7] U. Saif, H. Pham, J.M Paluska, J. Waterman, C. Terman, and S. Ward. "A case for goaloriented programming semantics". In Workshop on System Support for Ubiquitous Computing (UbiSys'03),5th International Conference on Ubiquitous Computing (UbiComp 2003), Seattle, WA, USA, October 12, 2003.

[8] J. Rao, X. Su. "A Survey of Automated Web Service Composition Methods". Semantic web Services and web Process Composition (SWSWPC), First International Workshop, July 6, 2004, San Diego, CA, USA, pp. 43-54.

[9] J.M Geib, C. Gransart, and P. Merle. "Corba : des concepts à la pratique". Dunod Informatique, September 1999.

[10] A. Woolrath, R. Riggs, and J. Waldo. "A distributed object model for the Java system." Computing Systems, 9(4) :291312, 1996.

[11] E. Frank."DCOM : Microsoft Distributed Component Object Model." Hungry Minds, Inc, 1997.

[12] R. Dale. "Inside COM, Microsoft Component Object Model". Microsoft Press, 1996.

[13]http://www.service-architecture.comconsulté le 30 juin 2012.

[14] C. Bettstetter and C. Renner. “A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol”. Proceedings of the 6th EUNICE Open European Summer School: Innovative Internet Applications, 2000.

[15] E. Guttman, C. Perkins, J. Veizades, and M. Day. “Service Location Protocol”, Version 2. IETF RFC 2608, Network Working Group,1999

[16] http://www.ariadne.ac.uk/issue29/gardner/intro.html consulté le 30 juin 2012.

[17] http://www.w3.org/ consulté le 30 juin 2012.

[18] L. Cabral, J. Domingue, E. Motta, T. Payne, and F. Hakimpour. Approaches to Semantic Web Services: an Overview and Comparisons. The Semantic Web: Research and Applications. LNCS, vol. 3053, pp. 225-239, Springer Berlin / Heidelberg, 2004.

[19] G. Alonso, F. Casati, H. Kuno, and V. Machiraiu. Web Services: Concepts, Architectures and Applications. Springer-Verlag, 480 pages, New York, 2004.

[20] http://www.uddi.org/ consulté le 30 juin 2012.

[21] https://www.ibm.com/developerworks/webservices/library/ws-bpelcol1/ consulté le 30 juin 2012.

[22] http://www.editions-eyrolles.com/Livre/9782212110470/services-web-avec-soap-wsdl-uddi-ebxml consulté le 30 juin 2012.

[23] F. Leymann. Web Services Flow Language (WSFL 1.0), IBM Software Group, 108 pages, 2001.

[24] B. Medjahed, A. Bouguettaya, and A. Elmagarmid. Composing Web Services on the Semantic Web. The International Journal on Very Large Data Bases (VLDB). Springer-Verlag, Inc, vol. 12, pp. 333-351, New York, 2003.

[25] T. Gruber. Toward Principles for the Design of Ontologies used for Knowledge Sharing. The International Journal of Human-Computer Studies, Kluwer Academic Publishers, vol. 43, pp. 907-928, 1995.

[26] http://www.daml.org/ consulté le 30 juin 2012.

[27] E. Motta, J. Domingue, L. Cabral, and M. Gaspari. IRS–II: a Framework and Infrastructure for Semantic Web Services. The SemanticWeb - ISWC 2003. LNCS, vol. 2870, pp. 306-318, Springer Berlin / Heidelberg, 2003.

[28] D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications. Elsevier Science, vol. 1, pp. 113-137, 2002.

[29] E. Motta. Reusable Components for Knowledge Modelling: Case Studies in Parametric Design Problem Solving. IOS Press publishers, 255 pages, Amsterdam, The Netherlands, 1999.

[30] B.N. Schilit, M. Theimer. Disseminating Active Map Information to Mobile Hosts. Network, IEEE , September/October 1994, Vol.8, N°5, pp. 22-32.

[31] P.J. Brown, J.D. Bovey, X. Chen. Context-Aware applications : From the Laboratory to the Marketplace. IEEE Personal Communications, 1997, Vol. 4, N°5, pp. 58-64

[32] N. S. Ryan, J. Pascoe,and D.R. Morse. Enhanced reality fieldwork : the context-aware archaeological assistant. InV.Gaffney, M. van Leusen, andS. Exxon, editors, Computer Applications in Archaeology 1997. Oxford :Tempus Reparatum, , Oct. 1998. (British Archaeological Reports)

[33] A. K. Dey and G. D. Abowd. Towards a Better Understanding of Context and Context Awareness. In Proceedings of the Workshop on the What, Who,Where, When and How of Context-Awareness, affiliated with the CHI 2000Conference on Human Factors in Computer Systems, The Hague,Netherlands. New York, NY : ACM Press, 2000.

[34] B.N. Schilit, N.I. Adams and R. Want. Context-Aware Computing Applications. InProceedings of the IEEE Workshop on Mobile ComputingSystems and Applications (WMCSA94), Santa Cruz CA, December 1994.Los Alamitos, CA : IEEE computer society press, 1995, pp 85-90

[35] P.J. Brown. The Stick-e Document: a framework for creating Context aware applications. In Electronic Publishing 96, (1996), Document Manipulation and Typography, 24-26 September 1996, Pala-Alto, California, USA, pp. 259-272

[37] Ward, A. Jones, A. Hopper. A New Location Technique for the Active Office. IEEE personnal Communications, 1997, Vol. 4, N°5, pp. 42-47

[38] J. Pascoe. Adding generic contextual capabilities to wearable computers. InProceedings of 2 nd International Symposium on Wearable Computers,October 1998, pp. 92-99

[39] K. Dey, Daniel Salber, Masayasu Futakawa and Gregory D.Abowd. An Architecture To Support Context-Aware Applications. GVU TechnicalReport GIT-GVU-99-23. Submitted to the 12th Annual ACM Symposiumon User Interface Software and Technology (UIST 99), June 1999

[40] Winograd, Terry. Architectures for Context. Human-Computer Interaction Journal, 2001, Vol. 6, N°2-3, pp. 401-419.

[41] Schilit B., Adams, N. & Want, R. Context-aware computing applications. InProceedings of the 1st International Workshop on Mobile ComputingSystems and Applications. Los Alamitos, CA : IEEE. 1994, pp. 85-90

[42] F. Bennett, T. Richardson and A. Harter. Teleporting - Making Applications Mobile. Proc. InIEEE Workshop on Mobile Computing Systems andApplications, December 1994, Santa Cruz, California, pp. 82-84

[43] Indulska, J., Robinson, R., Rakotonirainy, A., and Henricksen, K. 2003. Experiences in Using CC/PP in Context-Aware Systems. InProceedings ofthe 4th international Conference on Mobile Data Management, January 21 - 24, 2003, Melbourne, Australia. (Lecture Notes In Computer Science, 2003, Vol. 2574, pp.247-261)

[44] Bauer M., Heiber, T., Kortuem, G. & Segall, Z. A collaborative wearable system with remote sensing. Proceedings of the 2nd InternationalSymposium on Wearable Computers (ISWC98), CA : IEEE, 1998, LosAlamitos, pp. 10-17

[45] Dey A. K., Abowd, G. D. & Wood, A. CyberDesk : A framework for providing self–integrating context–aware services. Knowledge Based Systems, 1998, Vol.11, N°1, pp. 3-13

[46] Kiciman E. & Fox, A. Using dynamic mediation to integrate COTS entities in a ubiquitous computing environment. InProceedings of the 2nd International Symposium on Handheld and Ubiquitious Computing(HUC2K). Heidelberg, Germany : Springer Verlag, 2000.

[47] Strang and Claudia Linnhoff-Popien: A context modeling survey. **In** UbiComp 1st International Workshop on Advanced Context Modelling,Reasoning and Management, 2004, Nottingham, pp. 34-41

[48] Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0 W3C Recommendation 15 January 2004 **http://www.w3.org/TR/CCPP-struct-vocab/**consulté le 30 juin 2012.

[49] Open Mobile Alliance http://www.wapforum.orgconsulté le 30 juin 2012

[50] Chen, H., et al.. An Ontology for Context-Aware Pervasive Computing Environments. In Workshop on Ontologies and Distributed Systems (IJCAI2003), August 2003, Mexico.

[51] Belhanafi N., Taconet C., and Bernard G. “Camido, A Context-Aware Middleware based on Ontology metamodel”. In CAPS 2005, Workshop on Context Awareness for Proactive Systems, Helsinki, Finland

[52] Anind K. Dey, Daniel Salber and Gregory D. Abowd, A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context- Aware Applications. special issue on context-aware computing in theHuman-Computer Interaction (HCI), 2001, Vol. 16, N° 2-4, pp. 97-166

[53] Manuel Román, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. Gaia: A Middleware Infrastructure to Enable Active Spaces.In IEEE Pervasive Computing, pp. 74-83, Oct-Dec 2002.

[54] Gu T., Pung H. K., and Zhang D. Q. A middleware for building contextaware mobile services. InProceedings of IEEE Vehicular TechnologyConference (VTC), 2004, Milan, Italy

[55] H. Chen, T. Finin, and A. Joshi. An Ontology for Context-Aware Pervasive Computing Environments. Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review, 18(3) :197–207, May 2004.

[56] H. Chen, F. Perich, D. Chakraborty, T. Finin, and A. Joshi. Intelligent Agents Meet Semantic Web in a Smart Meeting Room. In Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2004), New York City, NY, July 2004.

[57]F. Gandon, and N. Sadeh. A Semantic eWallet to Reconcile Privacy and Context Awareness, Second International Semantic Web Conference, Florida, October 2003.

[58] J. Rao and X. Su. “A survey of Automated Web Service Composition Methods”. In Proc. of 1st International Workshop on Semantic Web Services and Web Process Composition, 2004.

[59] Jinghai Rao and Xiaomeng Sub Norwegian. “ A Survey of Automated Web Service Composition Methods”. University of Science and Technology .Department of Computer and Information Science N-7491, Trondheim, Norway. 2005.

[60] Frédéric Pourraz, Diapason une approche formelle et centrée architecture pour la composition évolutive de services Web, Thèse de Doctorat, LISTIC : Laboratoire d’Informatique, Systèmes,Traitement de l’Information et de la Connaissance, le 10 Décembre 2007.

[61]F. CASATI, S.Ilncki, L. Jin, V. Krishnamoorthy, and M. Shan. Adaptive and Dynamic Service Composition in eFlow. Advanced InformationSystems Engineering. LNCS, vol. 1789,pp.13-31, Springer Berlin/Heidelberg, 2000.

[62] David Hollingsworth, The Workflow Reference Model 10 Years, édition Workflow Handbook, 2004.

[63] Julien Bourdon, Philippe Beaune & Humbert Fiorino. « Architecture multi-agents pour la composition automatique de web services ». Actes de l'atelier Intelligence Artificielle et Web, 2007.

[64] Issam Chebbi, CoopFlow une approche pour la coopération ascendante de workflows dans le cadre des entreprises virtuelles, Thèse de Doctorat, Institut National des

Télécommunications dansle cadre de l'école doctorale SITEVRY en co-accréditation avec l'Université d'Évry Val d'Essonne,le 17 Avril 2007.

[65] F. Casati, M. Sayal, et M.-C Shan, Developping e-services for composing e-services, In Proceedings of 13th International Conference on Advanced Information Systems Engineering (CAiSE), Interlaken, Switzerland, June 2001.

[66] P. Regnier, Algorithmique de la Planification en Intelligence Artificielle, édition Cepadues, 2004.

[67] M. Ghallab, D. Nau, et P. Traverso, Automated Planning : Theory and Practice, édition Morgan Kaufmann, 2004.

[68] S. McIlraith, et T. Son, Adapting Golog for composition of semantic Web services, In Proceedings of the 8th International Conference on Knowledge Representation and Reasoning, Morgan Kaufmann Publishers, pages 482-493, Toulouse, France, April 2002.

[69] P. Albers, Extension de la représentation pour la prise en des effets dépendant du contexte et des axiomes du domaine, Thèse de Doctorat, Université de Paul Sebatier de Toulouse, 1997.

[70] Joachim Peer. "A PDDL Based Tool for Automatic Web Service Composition". in: Proceedings of the Second Workshop on Principles and Practice of SemanticWeb Reasoning (PPSWR 2004) at the 20th International Conference on Logic programming, 2004

[71] Maja Vukovic_, Peter Robinson. ADAPTIVE, PLANNING BASED, WEB SERVICE COMPOSITION FOR CONTEXT AWARENESS. Advances in Pervasive Computing, 2004

[72] Dipanjan Chakraborty Yelena Yesha, and Anupam Joshi. "A Distributed Service Composition Protocol for Pervasive Environments". WCNC 2004 - IEEE Wireless Communications and Networking Conference, vol. 5, no. 1, March (2004) pp. 2579-2584

[73] Mathieu Vallée,Laurent Vercouter, Fano Ramparany. « Composition flexible de services d'objets communicants». In Proc. bimob'05, volume 120, pages 85§192, 2005.

[74] K. Tari Y. Amirat . Rule-based Approach for Automatic Service Composition in Ubiquitous Environment. The 6th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI 2009)

[75] Y. Charif, N. Sabouret. Coordination in introspective multi-agent systems . In proc. of the International Conference on Intelligent Agent Technology (IAT'07), pp. 412-415. Silicon Valley, California, USA, 2007

[76] Mathieu Vallée, Laurent Vercouter, Fano Ramparany: Composition flexible de services d'objets communicants pour la communication ambiante. JFSMA 2006: 267-270

[77] Jiehan Zhou , Jukka Riekki, Context-Aware Pervasive Service Composition and its implementation,Personal and Ubiquitous Computing Volume 15 Issur 3, March 2011

[78] Taylor, I., Shields, M., Wang, I., and Philp, R. “Grid Enabling Applications Using Triana”, Workshop on Grid Applications and Programming Tools, Seattle. In conjunction with GGF8 June 25 ,2003.

[79] Stevens, R.D., Robinson, A.J., and Goble, C.A. “myGrid: Personalised Bioinformatics on the Information Grid”. Bioinformatics Vol. 19 Suppl. 1 2003, (Eleventh International Conference on Intelligent Systems for Molecular Biology)

[80] A.Kouicem. Composition dynamique de services en environnements ubiquitaires. Mémoire de magistère en informatique. Université Abderrahmane Mira, Bejaia, Algérie, 2009.

[81] J. Ferber, Les Systèmes Multi agents. Vers une intelligence collective. Interéditions, 1995.

[82] Yves Demazeau. From interactions to collective behaviour in agentbased systems. European Conference on Cognitive Science, Saint-Malo – France, Avril 1995.

[83] Foundation for Intelligent Physical Agents. http://www.fipa.org consulté le 30 juin 2012.

[84] Tim Finin, Yannick Labrou, and J. Mayfield. KQML (Knowledge Query and Manipulation Language) as an agent communication language. In J. M. Bradshaw, editor, Software Agents. MIT Press, 1997.

[85] J.L. Austin. How to do things with words. Clarendon Press, Oxford, 1962.

[86] J.R. Searle. Speech Acts. Cambridge University Press, 1969

[87] FIPA Modeling : Interaction Diagrams. http://www.fipa.org consulté le 30 juin 2012.

[88] FIPA Directory Facilitator. http://www.fipa.org consulté le 30 juin 2012.

[89] F.Sebaak. Architecture intergicielle sémantique basé sur le standard OSGI pour les services robotiques. Mémoire de magistère en informatique. Université Abderrahmane Mira, Bejaia, Algérie, 2008.

[90] A.Yachir, K.Tari, Y.Amirat, A.Chibani, N.Badache: QoS based framework for ubiquitous robotic services composition. IROS 2009: 2019-2026

[91] Jinchun Xia, Carl K. Chang, Tae-Hyung Kim, Hen-I Yang, Raja Bose and Sumi Helal . “Faultresilient Ubiquitous Service Composition”. Proceedings of IE'07, Ulm, Germany, pp. 108 -115, Sept. 2007.

[92] Mikko Perttunen, Marko Jurmu, Jukka Riekki. “A QoS Model for Task-Based Service Composition”. Managing Ubiquitous Communications and Services, 2007 .

Résumé

Les équipements électroniques envahissent progressivement notre quotidien, et la démocratisation des réseaux sans fil transforme la vision de l'informatique ubiquitaire en réalité. La composition de services qui consiste à faire interagir plusieurs dispositifs pour répondre aux requêtes complexes des utilisateurs est l'un des défis majeur de l'informatique ubiquitaire en vue des contraintes d'ouverture, de dynamique, de distribution et d'hétérogénéité de ce type d'environnement.

Nous proposons ici un mécanisme de composition de services flexible et automatique. Il permet une génération dynamique des schémas de composition de manière distribuée, et cela grâce à la coordination multi-agents. L'approche est centrée autour de la sensibilité au contexte, quant à la QoS, elle est relative aux exigences de l'utilisateur et aux performances des services et dispositifs disponibles et elle est traitée comme une donnée contextuelle ce qui permet d'optimiser le résultat fournie à l'utilisateur.

Nous avons testé l'approche sur différents scénarii pour illustrer chaque aspect. Elle est implémentée en JADE qui est une plate-forme multi-agents.

Mots clés : informatique ubiquitaire, composition dynamique de services, sensibilité au contexte, système distribué, système multi-agents, JADE.

Abstract

Nowadays, electronic equipments gradually invading our lives, and the democratization of wireless networks transforms the vision of ubiquitous computing is becoming more and more real. Service composition which consists of multiple interacting devices to meet complex user queries is one of the major challenges of ubiquitous computing to stress openness, dynamics, distribution and heterogeneity of these types of environments.

We propose a mechanism for flexible and automatic service composition. It allows dynamic generation of compositional schemes in a distributed fashion, thanks to the multi-agent coordination. This approach is centered on context awareness. Concerning the QoS, it is up to the user's requirements and performance of available services and devices and is treated as a given context which optimizes the results provided to the user.

We tested the approach on various scenarios to illustrate each aspect. It is implemented in Jade which is a multi-agents Framework.

Key words: ubiquitous computing, dynamic service composition, context awareness, distributed system, multi-agents systems, JADE.