REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderrahmane Mira de Béjaïa
Faculté des Sciences et Sciences de l'Ingénieur

Département d'Informatique
Ecole Doctorale Réseaux et Systèmes Distribués

Mémoire de Magistère en Informatique

Option
Réseaux et Systèmes Distribués

Thème

ARCHITECTURE INTERGICIELLE SÉMANTIQUE BASÉE SUR LE STANDARD OSGi POUR LES SERVICES ROBOTIQUES

Présenté par

Faouzi SEBBAK

Directeur du mémoire : M. Yacine AMIRAT, Professeur, Université Paris 12, France.

Devant le jury composé de :

Président : Mme. Saadia TAS, Maître de Conférences, Université de Béjaïa, Algérie.

Rapporteur : M. Yacine AMIRAT, Professeur, Université Paris 12, Val-de-Marne, France.

Examinateurs : M. Abdallah BOUKERRAM, Maître de Conférences, Université de Sétif, Algérie.

M. Ali MELLIT, Maître de Conférences, Université de Jijel, Algérie.

Promotion 2006 - 2007

A mes très chèrs parents,

A mes frères et sœurs,

A tout mes proches,

A tout mes amis,

Remerciements

J'exprime mes grandes reconnaissances et mes vifs remerciements au professeur Yacine AMIRAT, pour m'avoir encadré et accordé son aide précieuse tout au long de cette année et pour le temps et l'attention qu'il a bien voulu consacrer au bon déroulement de ce projet.

Je remercie également Monsieurs Karim TARI et Abdelghani CHIBANI, pour leurs disponibilités et leurs bons conseils tout au long du projet.

J'adresse mes sincères remerciements à Madame Saadia TAS, Maître de Conférences à l'Université de Béjaia, qui m'a fait l'honneur de présider le jury de thèse.

J'exprime ma profonde reconnaissance à Monsieur Ali MELLIT, Maître de Conférences à l'université de Jijel, et à Monsieur Abdallah BOUKERRAM, Maître de Conférences à l'Université de Sétif, pour l'intérêt qu'ils ont bien voulu porter à ce travail en acceptant d'en être examinateurs.

Mes très vifs remerciements vont également à Monsieur Kamel TARI, chef du département informatique de l'Université de Béjaia pour son aide, sa disponibilité et ses contributions pour l'école doctorale ReSyD.

Je désire remercier tout particulièrement mes parents, mes frères, mes sœurs, et tous mes proches leurs soutiens et leurs encouragements.

Résumé

La *robotique ubiquitaire* et de service est actuellement confrontée à des verrous technologiques qui apparaissent aux niveaux de l'interopérabilité des services et des communications, liée à l'hétérogénéité des équipements et au caractère dynamique des espaces ubiquitaires. Dans ce projet de recherche, nous proposons une infrastructure intergicielle s'appuyant sur deux couches logicielles. La première repose sur une architecture intergicielle *orientée services* basée sur le standard *OSGi* (*Open Service Gateway Initiative*) pour intégrer de façon transparente des objets communicants hétérogènes (*capteurs/actionnaires*, *robot*, etc.). La deuxième couche est une couche sémantique à base d'agents, construite au dessus de la couche *OSGi*, pour fournir des services dits *sensibles au contexte* d'utilisation. Cette infrastructure intergicielle est mise en œuvre en simulation sur le simulateur *USARSim* dans quelques scénarios de services robotiques d'exploration d'un environnement de désastre à la recherche des victimes.

Mots clés: *Informatique ubiquitaire*, *middlewares robotiques*, *OSGi*, *système multi agents*, *USARSim.*

Abstract

The *ubiquitous service robots* is currently facing technological appearing at the interoperability of services and communications related to the heterogeneity of equipment and dynamic ubiquitous spaces. In this project, we propose a middleware based on two software layers. The first is based on a *service-oriented* middleware based on the open standard *OSGi* (*Open Service Gateway Initiative*) to seamlessly integrate heterogeneous communicating objects (*sensors / actuators*, *robots*, …). The second layer is an agent-based semantic layer, built above the *OSGi* layer to provide context aware services. This middleware is being implemented in the *USARSim* simulator in service robots scenario of exploration of an environmental disaster in research of victims.

Key words: *Ubiquitous computing, robot middleware, OSGi, multiagent systems, USARSim.*

ملخص

إن ميدان المستخدم الآلي المتنقل يواجه حاليا تحديات تكنولوجية والتي تظهر في التشغيل البيني للخدمات والإتصالات ،وذلك راجع لعدم تجانس المعدات وديناميكية مجالات الإستعمال. في هذا المشروع البحثي، نقترح هندسة برنامجية ترتكز على طبقتين من البرمجيات. الطبقة الأولى والتي تتمثل في هندسة البرمجيات الموجهة نحو الخدمات والتي ترتكز على المعيار OSGi لدمج أجهزة إتصال غير متجانسة (أجهزة الإستشعار / أجهزة التحكم ، الإنسان الآلي ، إلى غير ذلك). الطبقة الثانية هي طبقة الدلالة المعلوماتية وترتكز على الوكلاء (agent) ، لتوفير الخدمات الملائمة لسياق الإستعمال. هذه الهندسة البرنامجية تمت محاكاتها عن طريق برنامج المحاكات USARSim في عدة سيناريوهات للخدمات الروبوتية للإستكشاف والبحث عن الضحايا في مكان وقوع كارثة.

المفاتيح : الاعلام الآلي المتنقل ، برمجيات المستخدم الآلي، OSGi ، USARSim.

Table des matières

Liste des figures

Introduction Générale

De nos jours, la plupart des dispositifs qui nous entourent sont équipés de microprocesseurs. Cela va de la voiture à la machine à laver en passant par le téléphone et autre assistant numérique. Par ailleurs, grâce à l'avènement des réseaux sans-fil, ces dispositifs sont de plus en plus dotés d'équipement de communication. Dès lors, ces différents équipements sont en mesure de dialoguer et d'interagir entre eux et avec le monde environnant. Cette tendance semble devoir s'accentuer dans le futur, notamment avec les nanotechnologies pour déboucher sur un monde où l'informatique serait omniprésente et invisible, c'est l'informatique ubiquitaire.

La *robotique ambiante* qui constitue une classe importante de la *robotique de service*, se place dans la continuité du domaine plus restreint de *l'informatique ubiquitaire*. En effet, un robot de service peut être vu comme un objet intelligent, communicant avec d'autres objets et offrant des services aux profils des utilisateurs.

La *robotique de service* est appelée à se développer de manière considérable dans les prochaines années. Microsoft, IBM, Fujitsu, Intel et Sony sont d'ores et déjà positionnés pour tirer profit de cette croissance. En effet, au sein d'un robot de dernière génération (que l'on qualifie parfois de "*symbiotique*") peuvent se croiser une multitude de technologies : Wi-Fi, RFID, reconnaissances vocale, spatiale (3D) et biométrique, capteurs, protocoles réseaux, intelligence artificielle, échanges inter-robots, etc.

Selon une étude récente, le marché émergeant de la robotique de service est estimé à 9 milliards de dollars en 2005 avec une perspective à 19 milliards de dollars en 2010 avant d'atteindre 50 milliards de dollars en 2025.

Problématique

Ce secteur de la robotique de service se trouve à l'heure actuelle confronté à une limite due à son jeune âge : le manque actuel de standards. Des verrous technologiques apparaissent aux niveaux de l'interopérabilité des services et des communications, liés à l'hétérogénéité des équipements.

Contribution

Dans ce cadre de mémoire, nous visons plusieurs objectifs. Le premier est de dresser un état de l'art sur les travaux de recherche dans le domaine des *middlewares* pour robots. D'une manière générale, il s'agit d'établir une classification des travaux et initiatives de standardisation dans le domaine des architectures logicielles distribuées (architectures orientées services, à base de services web, à composants logiciels, etc.) destinées à la robotique de service en général et à la robotique ubiquitaire en particulier. Ce travail sera complété par une étude des modèles de représentation et de traitement des connaissances.

Le deuxième objectif vise à développer une architecture intergicielle sémantique orientée services basée sur le standard OSGi qui permet de faciliter la gestion du contexte. Les agents ou composants constituant l'architecture de l'intergiciel doivent reposer sur un modèle de représentation et de gestion des connaissances contextuelles à base d'ontologies. Ce modèle permettra de décrire d'une part, la sémantique des connaissances contextuelles et d'autre part, la

manière de partager et d'interpréter ces dernières. Cette infrastructure sera mise en œuvre en simulation dans un scénario de services robotiques d'exploration d'un environnement de désastre à la recherche des victimes.

Organisation du Mémoire

Ce mémoire est organisé en cinq chapitres :

Dans le chapitre I, ***Middlewares pour Robot***, nous dressons un état de l'art sur les travaux de recherche dans le domaine des middlewares pour robots, en établissant une classification des travaux et initiatives de standardisation.

Le chapitre II présente un état de l'art sur ***Les Intergiciels Sensibles au Contexte***. Nous présentons tout d'abord, les notions de contexte, la sensibilité au contexte, et la modélisation du contexte. La deuxième partie du chapitre est consacrée quant à elle à l'étude des principales infrastructures logicielles de gestion du contexte. Ces infrastructures sont analysées puis comparées selon plusieurs aspects : modèle d'architecture, modèle de représentation du contexte, modèles de perception et de traitement du contexte, le modèle de sensibilité au contexte, la validité du contexte, la découverte dynamique, la gestion de l'historique du contexte, la sécurité et la confidentialité.

Dans le chapitre III, ***La Représentation des Connaissances,*** nous présentons les modèles de traitement et de représentation des connaissances contextuelles, et nous analysons les avantages que présente le web sémantique pour la représentation et la gestion des connaissances en informatique ubiquitaire.

Le chapitre IV quand à lui, présente la proposition d'une ***Architecture intergicielle sémantique basée sur le standard OSGi*** ; dans ce chapitre, nous présentons le modèle de l'intergiciel que nous proposons pour la mise en oeuvre de services robotiques. Nous présentons tout d'abord le paradigme des systèmes multi agents et montrons son intérêt pour la conception de systèmes complexes en général et des environnements ubiquitaires intelligents en particulier. Dans notre approche, un environnement ubiquitaire est perçu comme un environnement multi agents, intégrant de façon transparente des agents intelligents, communicants, et autonomes, offrant aux utilisateurs de cet environnement des services sensibles au contexte. Dans la suite du chapitre, nous présentons le modèle de notre intergiciel de service robotiques en développant en particulier le modèle de son architecture multi agents. Dans cette architecture, la gestion du contexte est répartie entre des agents de contexte et des services *OSGi*, les *services OSGi* ont pour rôle de cacher les détails d'implémentation de bas niveau de différents capteurs et actionnaires disséminés dans l'environnement ubiquitaire.

Nous terminons avec le chapitre V, ***Mise en œuvre et validation de l'intergiciel***, qui présente la mise en oeuvre de l'architecture intergicielle proposée dans le chapitre IV. Dans la première partie du chapitre, nous présentons les outils et plateformes logiciels sur lesquels nous nous sommes appuyés pour cette mise en oeuvre. Dans la deuxième partie, nous décrivons en particulier le modèle d'implémentation des services robotiques *OSGi* et des agents constituant l'intergiciel, et nous étudions un scénario pour la validation du modèle proposé : Ce scénario concerne le développement des agents et des services robotiques pour l'exploration d'un environnement de désastre à la recherche des victimes.

En annexe, on trouvera les détails d'implémentation des services robotiques développés.

Chapitre 1 :

Middlewares pour robots

Introduction

Quand on parle de robotique, il faut distinguer la robotique industrielle classique (robots soudeurs sur une chaîne de montage) de la robotique de service. La robotique de service englobe les toutes dernières générations de robots qui apportent aux humains de façon mobile et autonome ou semi autonome une aide concrète dans une multitude de circonstance. Selon l'IFR (International Federation of Robotics) un robot de service est un robot opérant de manière semi ou complètement autonome, et exécutant des services utiles au bien-être des humains comme la surveillance des sites sensibles, l'assistance à la mobilité des personnes âgées ou handicapés, le guidage et l'assistance à la réalisation du geste en chirurgie [Schraft 00]. Un robot de service peut également exécuter des services utiles à l'entretien d'équipement comme le nettoyage de sols, de tube et de fenêtres.

Plus globalement, la robotique de service concerne plusieurs domaines d'application : L'agriculture, la sécurité, la maintenance, l'exploration spatiale, l'exploration maritime, la défense (drones, et dernièrement, soldats robots), le transport (véhicule automatiques, dirigeables, etc.) et la médecine. Un des domaines d'application les plus porteur est celui des robots d'intérieur. Ces robots sont destinés à assurer plusieurs fonctions telles que le loisir, l'éducation et l'assistance.

Les acteurs IT de tout premier plan (Microsoft, IBM, Intel...) s'intéressent de plus en plus fortement à ce secteur, cherchant à capter les profits annoncés d'un marché en croissance exponentielle. En effet, au sein d'un robot de dernière génération (que l'on qualifie parfois de "symbiotique") peuvent se croiser une multitude de technologies : Wi-Fi, RFID, reconnaissances vocale, spatiale (3D) et biométrique, capteurs, protocoles réseaux, intelligence artificielle, échanges inter-robots, etc. Ainsi un robot de service peut être vu comme un objet intelligent, communicant avec d'autres objets et offrant des services.

Les enjeux de ce marché de la robotique de service (qualifié pour le moment d'émergent) sont considérables. Peu d'études se hasardent vraiment à le quantifier, mais quelques chiffres filtrent néanmoins. Selon l'Association Robotics de Japan (*Japan Robotics Association*), le marché global de la robotique, estimé à 9 milliards de dollars en 2005, pourrait passer à 19 milliards de dollars en 2010 avant de s'envoler vers des sommets en 2025 à 50 milliards de dollars.

La convergence des réseaux et de l'électronique embarquée rend possible le déploiement d'une nouvelle gamme de services dans des environnements divers. Ces services prennent toute leur pleine signification dans le contexte d'aide ou d'assistance aux personnes âgées ou handicapées. Mais l'exploitation des technologies existantes se heurte à un certain nombre de difficultés, et de nombreux équipements souffrent de problèmes d'interopérabilité. Ainsi, l'intégration de façon transparente des objets communicants hétérogènes, tels que les terminaux mobiles, les capteurs/actionneurs et les robots pour aboutir à une infrastructure intelligente, permet de fournir des services sensibles au contexte d'utilisation, et ces services doivent s'adapter dynamiquement à leur contexte d'utilisation et forment ainsi un environnement ubiquitaire. Cette infrastructure intelligente met en lumière d'autres problématiques de recherche telles que la représentation et le traitement intelligent des connaissances associées aux services.

Le secteur de la robotique de service se trouve, pour le moment, confronté à une limite due à son jeune âge, empêchant la progression vertigineuse qui lui est prédite, d'avoir lieu : Le manque de standard.

1.1. Introduction à l'intergiciel

1.1.1. Pourquoi l'intergiciel ?

Dans leurs efforts pour relever les défis liés à l'interopérabilité entre applications, les concepteurs et les réalisateurs d'applications informatiques rencontrent des problèmes plus concrets dans leur pratique quotidienne.

Dans le cas des applications dites patrimoniales, qui ont été développées avant l'avènement des standards ouverts actuels et qui utilisent des outils propriétaires et nécessitent des environnements spécifiques, ne peuvent être utilisées qu'à travers une interface spécifiée, et ne peuvent être modifiées, et dans la plupart du temps, l'application doit être reprise telle quelle car le coût de sa réécriture serait prohibitif.

Pour ces applications, et pour d'autres cas similaires, l'utilisation d'une couche logicielle intermédiaire s'avère une alternative très prometteuse dans le sens où elle doit :

1. Cacher la répartition, c'est-à-dire le fait qu'une application est constituée de parties interconnectées s'exécutant à des emplacements géographiquement répartis ;
2. Cacher l'hétérogénéité des composants matériels, des systèmes d'exploitation et des protocoles de communication utilisés par les différentes parties d'une application ;
3. Fournir des interfaces uniformes, normalisées, et de haut niveau aux équipes de développement et d'intégration, pour faciliter la construction, la réutilisation, le portage et l'interopérabilité des applications ;
4. Fournir un ensemble de services communs réalisant des fonctions d'intérêt général, pour éviter la duplication des efforts et faciliter la coopération entre applications.

Cette couche intermédiaire de logiciel, schématisée sur la figure 1.1, est désignée par le terme générique d'intergiciel *(middleware)*. Un intergiciel peut être à usage général ou dédié à une classe particulière d'applications.

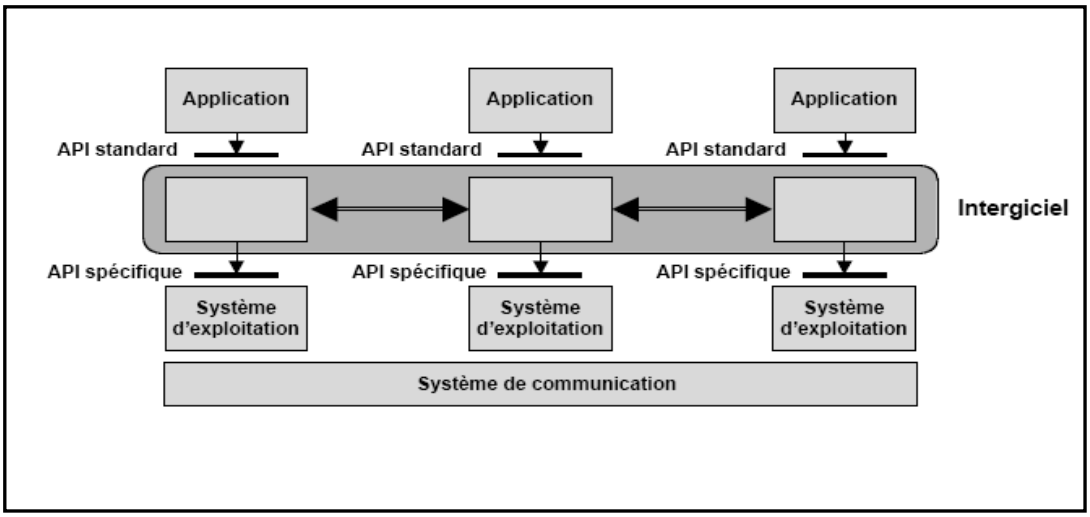

Figure 1.1 : Architecture de l'intergiciel

L'utilisation de l'intergiciel présente plusieurs avantages dont la plupart résultent de la capacité d'abstraction qu'il procure : Cacher les détails des mécanismes de bas niveau, assurer l'indépendance vis-à-vis des langages et des plateformes, permettre de réutiliser l'expérience et parfois le code, faciliter l'évolution des applications. En conséquence, la réduction du coût et de la durée de développement des applications (l'effort étant concentré sur les problèmes spécifiques et non sur l'intendance), l'amélioration de leur portabilité et de leur interopérabilité.

Un inconvénient potentiel est la perte de performances liée à la traversée de couches supplémentaires de logiciel. L'utilisation de techniques intergicielles implique par ailleurs de prévoir la formation des équipes de développement.

1.1.2. Historique et définitions

Le domaine de l'intergiciel, apparu dans les années 1990, a pris une place centrale dans le développement des applications informatiques réparties. L'intergiciel joue aujourd'hui, pour celles-ci, un rôle analogue à celui d'un système d'exploitation pour les applications centralisées : Il dissimule la complexité de l'infrastructure sous-jacente, il présente une interface commode aux développeurs d'applications et il fournit un ensemble de services communs.

Les acteurs de l'intergiciel sont nombreux et divers : Les organismes de normalisation, les industriels du logiciel et des services et les utilisateurs d'applications. Les consortiums et organisations diverses qui développent et promeuvent l'usage du logiciel libre tiennent une place importante dans le monde de l'intergiciel. Enfin, ce domaine est l'objet d'une recherche active, dont les résultats sont rapidement intégrés. Une conférence scientifique annuelle *ACM-IFIP-Usenix* Middleware, est consacrée à l'intergiciel, et de nombreux ateliers spécialisés sont fréquemment créés sur des sujets d'actualité.

Le domaine de l'intergiciel est en évolution permanente. Pour répondre à des besoins de plus en plus exigeants, les produits doivent s'adapter et s'améliorer. De nouveaux domaines s'ouvrent, comme l'informatique ubiquitaire, les systèmes embarqués, les systèmes mobiles et les robots de service. Le cycle de cette évolution est rapide : Chaque année apporte une transformation significative du paysage des normes et des produits de l'intergiciel.

Parmi les définitions les plus répondues du mot intergiciel nous trouvons :

Définition 1 : Un intergiciel est une couche logicielle autorisant des logiciels distribués sur différentes stations (ou noeuds) distantes à communiquer de manière transparente, indépendamment des standards réseaux sous jacents **[JDN]**.

Définition 2 : Un intergiciel est un logiciel présent sur un noeud du système réparti, qui offre aux composants applicatifs présents sur ce noeud les abstractions d'un modèle de répartition **[Quinot 01]**.

Définition 3 : Un intergiciel est un logiciel central autorisant des applications à communiquer les unes avec les autres par le biais de messages, alors qu'elles n'étaient pas conçues jusque-là pour dialoguer ensemble. Il facilite ainsi l'accès à des données stockées dans des systèmes qui ne sont pas toujours compatibles (Figure 1.2) **[OSGi 07]**.

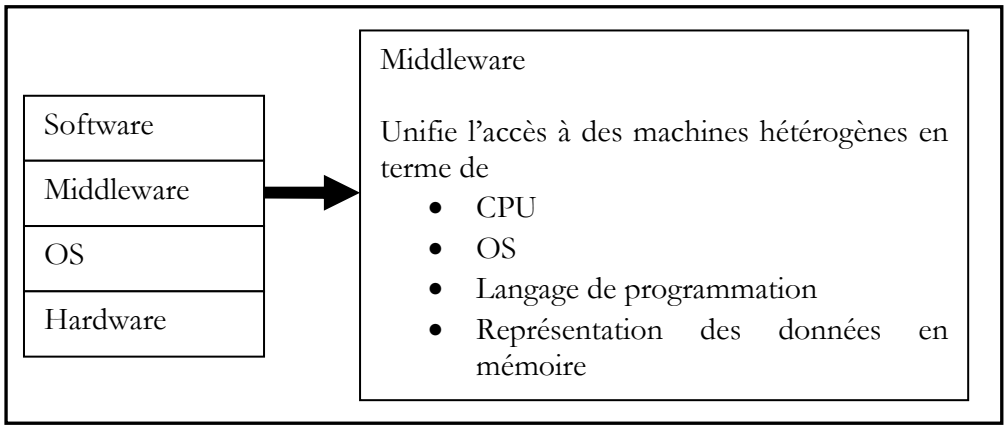

Figure 1.2: Architecture de l'intergiciel

1.2. La Robotique de service

Par opposition à la robotique industrielle, dont les applications sont essentiellement manufacturières, la robotique de service vise à sortir le robot du cadre contraignant de l'atelier ou de la chaîne de montage. La robotique de service couvre un très large champ d'application, de la médecine à l'agriculture, en passant par la surveillance, le marketing, les loisirs et l'assistance aux personnes. La sécurité active et la fiabilité sont, par conséquent, des impératifs fondamentaux de la mise en oeuvre des robots de service.

Un robot de service évolue dans un environnement aménagé par l'homme et donc a priori structuré. Toutefois, cet environnement n'est pas conçu de manière à faciliter les tâches du robot. Il est variable et évolutif et nécessite de la part du robot une autonomie décisionnelle pour un fonctionnement robuste. Ainsi, le robot doit posséder des moyens de perception et de raisonnement permettant une grande facilité de programmation, une capacité de détecter et de traiter une grande classe de situations notamment dans des environnements dynamiques ou sujets à perturbations.

Le secteur de la robotique de service se trouve à l'heure actuelle confronté à une limite due principalement au manque actuel de standards. Des verrous technologiques apparaissent aux niveaux de l'interopérabilité des services et des communications, liés à l'hétérogénéité des équipements. Aboutir à une infrastructure «intelligente» intégrant de façon transparente des objets communicants hétérogènes (terminaux mobiles, capteurs/actionneurs, robots, etc.) permet de fournir des services dits sensibles au contexte d'utilisation (paramètres espace-temps/temps, profilent des personnes à qui sont destinés les services : âgées, handicapées ou enfants, conditions d'utilisation). Ainsi ces services doivent s'adapter dynamiquement à leur contexte d'utilisation. On parle alors d'environnement ambiant ou ubiquitaire. Du point de vue des domaines d'application, cette thématique de recherche est par exemple particulièrement justifiée par le vieillissement de la population qui amènera de plus en plus de monde à souhaiter un contrôle partiel de leur environnement immédiat.

Il existe plusieurs environnements de développement logiciel, dont plusieurs sont propriétaires, dédiés à la conception de robots mobiles et autonomes, qui proposent chacun une solution spécifique selon les exigences considérées (par exemple, RobotFlow, Saphira, Orocos, Mobility et MIRO). Cette multiplication d'environnements de développement logiciel robotique comporte plusieurs effets indésirables :

- Elle limite la portabilité des mises en oeuvre faites à partir d'environnements de développement logiciel différents ;
- Elle rend difficile l'évaluation des performances des architectures décisionnelles entre elles ;
- Elle tend à diviser le domaine par la difficulté de communiquer et d'interpréter les résultats obtenus.

1.3. Middleware pour robot

Une architecture robotique précise, autant que possible, les composants logiciels et matériels utilisés pour mettre en place un robot de service, et les modalités d'interaction de ces composants. Dans le présent travail, c'est l'architecture logicielle qui nous intéresse bien que certains aspects matériels puissent être évoqués, la question de la nécessité d'une architecture pour la mise en oeuvre de robots de service autonomes reste entière. La réponse tient principalement dans le fait que les robots sont généralement des systèmes complexes. Ils font intervenir un grand nombre de capteurs et d'actionnaires, un grand nombre de traitements tels que du traitement du signal, du raisonnement géométrique, des processus de planification d'actions ou des boucles de contrôles. Certains de ces traitements doivent se faire en temps réel, alors que d'autres, qui sont en général de complexité exponentielle, prennent un temps difficilement prévisible. Quoiqu'il en soit, une architecture logicielle indique comment ces différents composants sont mis en œuvre ? comment ils sont organisés ? comment ils communiquent et interagissent ?

Les principales propriétés attendues d'une architecture pour robot peuvent être résumées en :

- **Programmabilité :** Une architecture doit permettre aux robots d'être des machines hautement et facilement programmables (tant du point de vue du programmeur de la machine que celui de l'utilisateur final). Du niveau fonctionnel au niveau décisionnel, il doit être possible de programmer des boucles de contrôles et des traitements de bas niveau, des contraintes de fonctionnement, des procédures d'affinement de buts, et aussi de spécifier des modèles d'actions.
- **Autonomie, Adaptabilité et Cohérence :** Le robot de service exécute les actions, affine et adapte ses plans et ses comportements en fonction de ses objectifs et de l'environnement tel qu'il le perçoit. Son comportement et ses réactions sont guidés par ses objectifs.
- **Réactivité :** Les différents composants de l'architecture doivent être capables de réagir de façon appropriée aux stimuli spécifiques qu'ils reçoivent.
- **Robustesse :** L'architecture doit permettre d'exploiter la redondance des sources d'information, des traitements, et la multiplicité des processeurs.
- **Sûreté :** L'évolution de la robotique de service et l'utilisation de robots dans des situations critiques requièrent l'utilisation de méthodes qui garantissent certaines propriétés de sûreté.
- **Extensibilité :** La modularité de l'architecture doit permettre d'ajouter de nouvelles fonctionnalités, sans remettre en cause l'existant.

Dans ce qui suit, nous présentons une synthèse sur les travaux de recherche relatifs au domaine des middlewares pour robot. Actuellement, plusieurs technologies logicielles (les architecture à couches, les techniques de programmation modulaire, bibliothèques communes, etc.) sont proposées et implémentées. Mais, la plupart sont développées indépendamment les unes des autres, et sont guidées par des applications et objectifs spécifiques. Parmi ces technologies, les technologies à base de composants logiciels et les middlewares pour l'informatique distribuée ont été largement appliqués dans des projets de recherche en robotique.

Composants logiciels : Un composant logiciel est un module autonome qui peut être installé sur différentes plateformes et exporte différents attributs ou méthodes sous forme d'interfaces. L'objectif principal de la notion de composant logiciel est la réutilisation de ce module logiciel, que ce soit lors de la construction de nouvelles applications par assemblage de composants existants ou lors de l'intégration de nouveaux composant dans des logiciels existants.

Cette approche à base de composants logiciels présente l'avantage de :

- Centralisation des compétences : L'équipe de développement peut être divisée en sous-groupes, et chacun se spécialise dans le développement d'un composant.
- Composant sur étagères/sous traitance: Des composants qui nécessitent des connaissances spécialisées doivent être développés par un fournisseur spécialisé.
- Facilité de déploiement et de mise à jour : Le déploiement et la modification d'un composant ne nécessitent pas la recompilation du projet complet.

Middlewares à architecture distribuée : On appelle Middleware tout ce qui se situe entre le réseau et les applications, parmi ces middlewares CORBA (*Common Object Request Broker Architecture*) est largement adopté pour les middlewares pour robot.

Récemment, la communauté des roboticiens a combiné ces deux technologies en un champ de recherche appelée Middleware Robot.

Un middleware robot peut être vu comme une couche logicielle qui se situe au dessus de la couche système d'exploitation. Actuellement, plusieurs équipes de recherche utilisent cette technologie de middleware pour le développement et l'intégration de leurs systèmes robotiques.

1.4. Vers la standardisation des Middlewares Robotiques

La plupart des middlewares ont été développés indépendamment les uns des autres, et sont guidés par des applications et objectifs spécifiques ce qui rend l'intégration et la réutilisation de leurs composants presque impossible. Cette problématique a donné naissances à des activités de standardisation dans le domaine de la robotique de service, basé sur les objectifs partagés entre différentes communautés.

L'objectif initial des différents groupes de travail était de faire interopérer les middlewares existants, et de réaliser des composants réutilisables par d'autres middlewares, tout en assurant leurs compatibilités. Parmi ces groupes de travail, *Robotics Domain Special Interest Group* (*Robotics-DSIG*) **[DSIG]** établie en février 2005, a fait des efforts considérables pour augmenter l'interopérabilité dans le domaine des middlewares robotiques, tout en coordonnant avec d'autres sous groupes de l'organisme international OMG. Ce groupe défini un système robotique comme tout système qui fournit des services intelligents dans un environnement ubiquitaire, via l'utilisation des différents capteurs, actionneurs, et des interfaces humaines.

En parallèle, Dans ce domaine de standardisation, plusieurs projets ont été démarrés. Parmi ces projets, quatre sont les plus répondus, et les plus influents sur le processus de standardisation a l'échelle mondial: *Ubiquitous Robitic Companion* (URC) du corée, *Common Platform for robots et Network Robot Forum* du Japon, *Ubiquitous Networking Robotics in Urban Setting* (URUS) lancé par un groupe Européen, et *IEEE Society of Robotics and Automation's technical Commitee on Networked Robots* lancé par les états unies.

Le projet coréen *Ubiquitous Robotic Companion* (URC) a été lancé par le ministère de l'information et de la communication (*MIC*) depuis 2004 pour une durée de quatre ans dont les objectifs sont de diriger les technologies robotiques existantes vers l'adoption des services robotiques intelligentes, d'agrandir le champ de coopération entre les robots et les humains, et d'étendre l'utilisation des technologies robotiques vers d'autres domaines industriels. Ce projet tourne autour du principe fondamental suivant : Rendre les services robotiques accessibles n'importe quand et n'importe où l'utilisateur désire. Un autre objectif visé par l'URC est de permettre l'intégration et la réutilisation des composants développés pour un robot spécifique dans d'autres robots sans aucune contrainte supplémentaire, tout en garantissant des services sécurisés et en temps réel.

Le projet *URUS* quand à lui, a été lancé par le groupe européen de l'atelier de recherche sur les systèmes robotiques en réseaux (*Networked Robot Systems*). Ce projet vise à définir un environnement de coopération entre les robots et les humains dans les tâches d'assistance, de transport du bien, et de la surveillance dans les secteurs urbains, tout en intégrant des capteurs intelligents avec d'autres équipements intelligents **[Alberto]**.

Le Japon, qui est un acteur très actif dans ce domaine de standardisation robotique, a créé en 2003 le *Network Robot Forum*, pour guider les recherches dans les réseaux robotiques, avec l'intention de standardiser l'information distribuée entre les réseaux ubiquitaires, les réseaux de capteurs, et les réseaux de robots.

Les états unis avec le groupe *IEEE RAS Technical Committee on Internet and Online Robots* crée en 2001, est focalisé sur des robots téléopérés à base des technologies de l'Internet, avant qu'il change le nom à *Networked Robots* en mai 2004.

Ajouter aux quatre précédents groupes de standardisation, le groupe Japonais *group of Standardisation Activities on the Robotics Middlewares*. Selon ce groupe, les futurs middlewares pour robot, doivent être interopérables avec un haut niveau de connectivité, et modulaires avec une totale réutilisation des composants. Le but principal de ce groupe est d'aller de la situation actuelle caractérisée par différents softwares spécialisés par robot, vers une plateforme commune des systèmes robotiques. Ce groupe international de standardisation propose un middleware standardisé avec un framework standard de communication, une interface standard d'application, et un schéma standard de donnés pour robot. Ce groupe de standardisation a proposé ORiN (*Open Resource interface for the Network/Open Robot interface for the Network*) qui est un middleware qui offre une interface de communication standardisée au-dessus d'équipements industriels automatisés (*Factory Automation*) incluant des robots **[Makoto 02]**. La spécification d'ORiN est publiée par le consortium ORiN auquel la plupart des fabricants de robots industriels japonais sont affiliés. ORiN a été construit autours d'un contrôleur d'access orienté objet basé sur les technologies objet distribuées tels que DCOM et CORBA, d'un Contrôleur d'Access de Protocole basé sur la technologie SOAP, et d'un Contrôleur de Langage de Définition de Ressource (CRD) pour représenter les données d'équipements basé sur XML.

1.4.1. RT-Middleware et standardisation

Le projet RT-Middleware (RT pour *Robot Technology*) est un projet développé au Japan par l'institut national AIST (*Advenced Industrial Science and technology*). Ce projet, qui a démarré en 2002, a pour but de développer un middleware pour robots, destiné à simplifier le développement et l'intégration de systèmes robotisés flexibles. Dans ce cadre, une nouvelle approche modulaire à base de composants logiciels a été proposée (*RT-Components*) dont les fonctions requises sont les suivants:

- **Granularité des modules** : Ou trois niveaux de modularités existent.

  - Granularité fine concerne les capteurs, les moteurs et les contrôleurs...

  - Granularité moyenne : concerne quand à elle, les algorithmes de traitement d'images et les robots mobiles avec capteurs, etc.

  - Granularité élevée : concerne les robots humanoïdes, et un environnement ubiquitaire intelligent incluant un réseau de capteurs et des robots.

- **Module actif** : Les éléments RT, contrairement aux objets distribués passifs, nécessitent une activité de collecte d'informations nécessaires pour notifier d'un événement les autres éléments et pour assurer les contraints temps réels.

- **Gestion du temps réel** : le middleware RT doit supporter le temps réel, qui est une fonction indispensable dans le systeme RT, avec une grande vitesse de communication et une coopération étroite entre les différents modules.

- **Gestion du temps** : Pour assurer la synchronisation et la coopération entre deux ou plusieurs modules selon un ordonnancement temps réel, il faut qu'une synchronisation temporelle entre les modules existe.

- **Réutilisation du soft** : un ensemble de librairies a été créé, pour éviter la réécriture des composants logiciels à chaque fois.

Comme plusieurs autres intergiciels, le modèle de composant RT est construit autour du middleware CORBA et il est constitué des différents objets et interfaces (Figure 1.3) comme les composants objets, l'Activité, les ports d'entrées, les ports de sortie et les interfaces de commande.

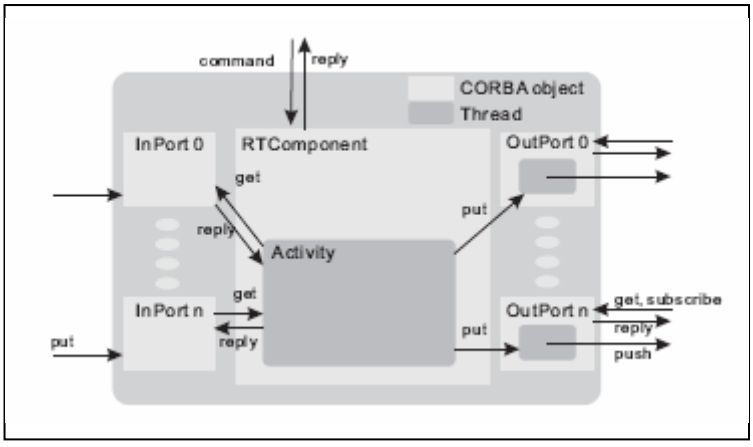

Figure 1.3: Architecture interne d'un RT-Component [Noriaki]

L'activité d'un composant RT peut être dans l'un des 11 états suivant : Création, Initialisation, Prêt, Démarrage, Actif, Arrêt, Annulation, Erreur, En cours d'arrêt, Erreur fatale, Inconnu. Et chaque état correspond à une classe que le développeur doit implémenter pour chaque composant RT.

L'utilisation de La technologie *RT-Middleware* permet d'aboutir à une structure hiérarchique du système robotique à base de composition. Ce principe de composition permet de

construire deux types de composants composites : synchrone, caractérisé par des états d'activité complètement synchrones, sérialisés et temps réel, et des composants composite asynchrone, qui ne nécessite pas des états internes synchronisés avec l'exécution parallèle des activités.

Sur la base du modèle précédent, un système de commande en effort d'un manipulateur a été implémenté à partir d'une distribution open source appelée « OpenRTM-aist » du middleware.

1.5. Approches existantes

L'une des approches la plus utilisée dans le domaine des middlewares robotique est le *Component-Based Software Engineering* (*CBSE*) **[Szyperski 02]**. C'est une approche modulaire basée sur des composants réutilisables. YARP, ORCA, OROCOS, et CARMEN sont des middlewares robotiques basés sur cette approche, et offrant ainsi la possibilité de réutiliser leurs composants.

Plusieurs facteurs jouent un rôle important dans la classification des middlewares robots. Selon le mécanisme de communication deux catégories de middleware peuvent être distinguées, les middlewares qui utilisent un modèle de communication spécifique comme le protocole de communication de base niveau TCP utilisé par CARMEN, et le ICE (*the Internet Communication Engine*) utilisé par ORCA. D'autres middlewares adoptent un mécanisme de communication distribué en se basant sur le middleware COBRA, parmi ces middlewares on trouve MIRO, OROCOS, et HPR (*Humanoid Robotics Project*).

D'autre middleware (tel que CLARAty) utilise une structure à base de couches et décompose ainsi le système robotique en plusieurs niveaux.

1.5.1. CORBA et les Middlewares Robotiques

Actuellement, plusieurs équipes de recherche utilisent la technologie middleware pour le développement et l'intégration de leurs systèmes robotiques. La plupart des plateformes logicielles développées s'appuient sur CORBA. CORBA est une plateforme répartie portable et ouverte qui fournit une infrastructure logicielle permettant à des applications distribuées et hétérogènes de communiquer. Elle a été mise en place par l'organisme international (OMG) regroupant différents acteurs informatiques. CORBA permet également l'intégration d'applications patrimoines existantes par l'encapsulation de celles-ci dans des objets réutilisables.

CORBA a été adopté par la plupart des middlewares robotiques, motivé par deux principaux facteurs : CORBA est un standard ouvert multi langages/multi plateformes, et relativement stable car c'est un standard ouvert pour l'utilisation et l'extension, mais totalement fermé pour les modifications provenant hors de l'organisme des standards OMG.

Alors, les middlewares basés sur CORBA offrent des grands avantages pour le domaine de la robotique ainsi que pour les applications à capteurs embarqués. Et les middlewares robotique ne cessent pas à l'adopter comme standard de communication et comme alternatif d'interopérabilité entre différents systèmes et langages. Parmi ces middlewares, nous trouvons :

- **MIRO** : (*Middleware for Robot*) qui est un middleware pour les robots mobiles, basé sur le TAO de CORBA est offre des classes génériques pour le contrôle des fonctionnalités des robots.

- **OROCOS** : (*Open RObot COntrol Systems*) basé aussi sur CORBA, et dont l'objectif est de développer des framework partageables, réutilisables et distribués.

- **Le HRP** : (*Humanoid Robotics Project*) utilise CORBA dans son OpenHRP plateforme de développement robotique.

1.5.2. Middlewares à composants logiciels basés sur CORBA

1.5.2.1. MIRO (*MIddleware for RObots*)

MIRO est un framework robotique orienté objet distribué, dont l'objectif est de contrôler des robot mobile et faciliter l'intégration des softwares hétérogène par la réutilisation des composants. Il est basé sur la technologie CORBA et ces composants de noyau ont été développés dans l'environnement de programmation C++ pour être utilisés sous Linux avec l'utilisation de l'ACE qui est un environnement adaptatif de communication.

En effet, en raison de l'indépendance de langage de programmation de CORBA, d'autres composants peuvent être écrits dans n'importe quel langage et sur n'importe quelle plateforme qui fournit des implémentations CORBA.

Comme le montre la figure 1.4, MIRO fournit deux ensembles de fonctionnalités, un ensemble de services pour adresser l'ensemble des capteurs actionnaires du robot mobile, et un framework qui contient un ensemble de classes pour le contrôle du robot mobile. L'ensemble des capteurs/actionnaires peut être à son tour modélisé comme des objets. De cette façon, un robot selon la vision MIRO peut être vu comme une agrégation de plusieurs capteurs, actionnaires et d'autres objets cognitifs **[Stefan]**.

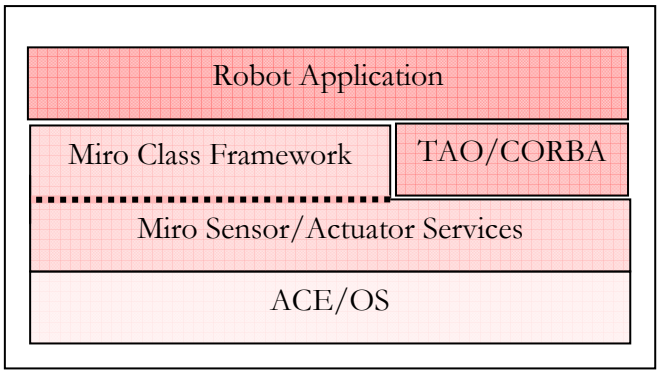

Figure 1.4: Architecture intergicielle de MIRO [Miro]

Actuellement, MIRO a été implémenté pour trois robots mobiles qui sont équipés avec plusieurs capteurs, actionnaires, ainsi que d'autres équipements comme les ordinateurs embarqués.

Le robot B21est contrôlé par un PC embarqué, le robot Pioneer-1 est contrôlé par portatif embarqué et par un ordinateur avec une liaison série, et enfin, le robot Sparrow-99 est contrôlé par un ordinateur embarqué.

Les principaux avantages présentés par la plateforme MIRO est sa portabilité sur différents environnements à condition que l'ACE et le TAO soient présents. Son architecture adresse d'une manière très efficace les problèmes liés à la structuration des softwares robotiques, et il offre une technologie de communication extensible avec un ensemble riche de capteurs.

Cependant, MIRO offre un nombre limité de services tels que le motionService, le LaserService et l'imageService, et il n'exploite pas les événements CORBA et il n'implémente pas le démarrage automatique des services requis. Un autre point faible de MIRO réside dans l'impossibilité de la gestion d'une plateforme à distance.

1.5.2.2. OROCOS (*Open RObot COntrol Software*)

Le projet européen OROCOS été lancé dans le but de construire un software ouvert pour le contrôle des robots. Il supporte quatre bibliothèques (Figure 1.5) écrites en C++ : la bibliothèque du Toolkit en temps réel, la bibliothèque de cinématique et de dynamique, la bibliothèque de filtrage bayésien et la bibliothèque des composants d'OROCOS **[Bruyninckx 01]**.

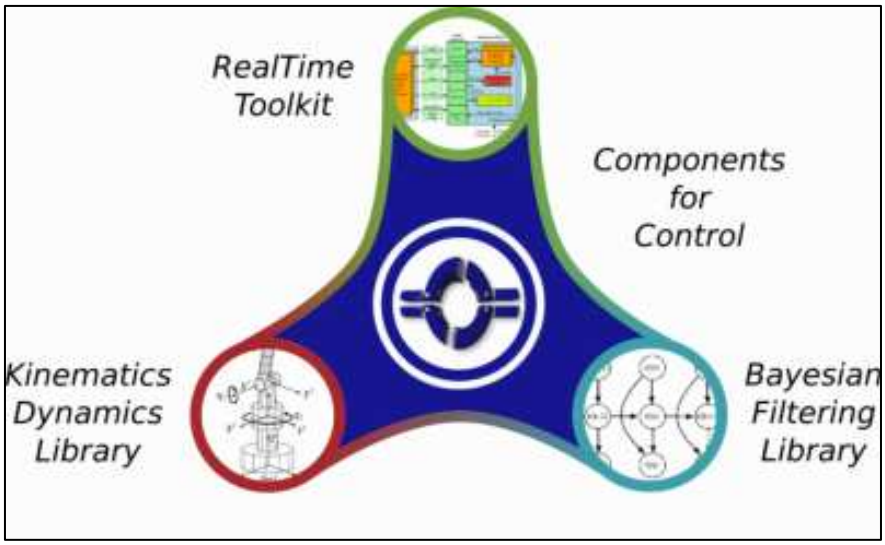

Figure 1.5: Architecture intergicielle d'OROCOS [Bruyninckx 01]

- ✓ L'Orocos Toolkit à temps réel (*RTT Real Time Toolkit*) n'est pas une application en soi-même même, mais il fournit l'infrastructure et les fonctionnalités nécessaires aux autres applications robotiques pour la construction des composants de très haut niveau d'interactivité en temps réel.
- ✓ La bibliothèque des composants d'Orocos (*OCL Orocos Components Library*) fournit un certain nombre de composants de contrôle prêt à exploiter. Les composants de commande, de gestion et de contrôle, ainsi que les composants d'accès au matériel sont disponibles.
- ✓ La bibliothèque de cinématique et de dynamique d'Orocos (*KDL Kinematics and Dynamics Library*) est une bibliothèque qui permet de calculer les chaînes cinématiques en temps réel.
- ✓ La bibliothèque de filtrage bayésien d'Orocos (*BFL Bayesian Filtering Library*) fournit un framework indépendant pour l'inférence dans les réseaux bayésiens dynamiques, c-a-d, elle offre un processus récursif d'information et des algorithmes d'estimation basés sur la règle de Bayes, telle que les filtres (prolongés) de Kalman, les filtres de particules (méthodes séquentielles de Monte), etc.
- ✓

1.5.2.3. ORCA (*Components for Robotics*)

ORCA est un middleware open source dont l'objectif est de développer des systèmes robotiques modulaires basés sur l'approche à base de composants. Comme tout autre middleware robot, le principal objectif d'ORCA est de promouvoir la réutilisation du software robotique tout en garantissant une totale flexibilité, extensibilité, et généralité.

ORCA utilise l'ICE (*the Internet Communication Engine*) **[Henning]** qui est une implémentation propriétaire des middlewares similaire au CORBA, et elle utilise Slice qui est un langage de spécification analogue au IDL de CORBA.

L'avantage principal de ORCA réside dans l'utilisation de l'ICE, qui présente d'autres avantages que CORBA, dont le principal est qu'il support plusieurs langages de développement tels que C++, java, python, php, c# et visual basic, et disponible pour linux, windows, solaris et MacOS, ainsi que pour des équipements embarqués.

La figure 1.6 montre un exemple de deux composants ORCA écrits par des langages différents implémentés sur différents systèmes d'exploitation.

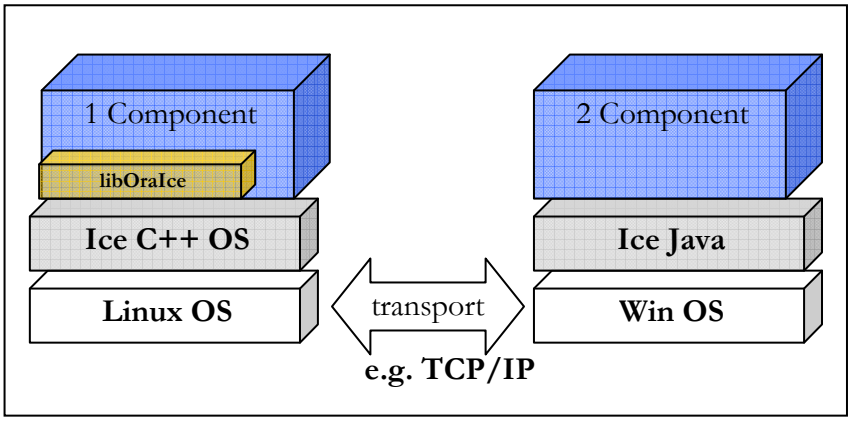

Figure 1.6: Deux composants ORCA

ICE présente encore d'autres avantages liés à la facilité de déploiement et d'administration et à l'optimisation de la communication entre composants.

ORCA supporte l'architecture client/serveur, l'abonnement notification, ainsi que les deux modes de communication par message asynchrone et synchrone.

ORCA est extensible avec une technologie orientée objet qui permet la réutilisation du code, portable puisqu'il utilise java, transparent en utilisant HORB, qui est un langage de robot extensible en utilisant Python.

1.5.3. Middlewares à architecture distribuée basée sur CORBA

1.5.3.1. MARIE (***M****obile and* ***A****utonomous* ***R****obotics* ***I****ntegration* ***E****nvironnement*)

Le principe de la réutilisation des softs dans le domaine des middlewares robotiques peut être vu à travers deux niveaux différents, le niveau fonctionnel et le niveau système. Dans le premier niveau, des petits éléments fonctionnels connu sous le nom de composants sont créés et rattachés pour construire un système robotique global. Tandis que, dans le deuxième niveau, les applications, les outils, et les environnements de programmation sont créés et ensuite rattachés **[Cote 04]**.

Selon la seconde approche qui se base sur des middlewares et outils existants pour la création d'un environnement robotique distribué, MARIE vise à la réutilisation des applications, des outils et des environnements de programmation, tout en trouvant des solutions efficaces aux problèmes liés à la communication entre différentes applications dans un environnement distribué. Dans un tel système, les applications ne partagent pas nécessairement les mêmes protocoles ou mécanismes de communication.

Si plusieurs middlewares tels que Player, CARMEN, et MIRO proposent des protocoles et mécanismes de communication spécifiques, qui doivent être implémentés par toutes les applications pour construire des blocks interopérables, ce qui présente quelques limitations liées à l'impossibilité par fois de modifier le code source d'une application. MARIE propose une autre solution, illustrée dans la figure 7, et adopte un système de médiation entre les applications.

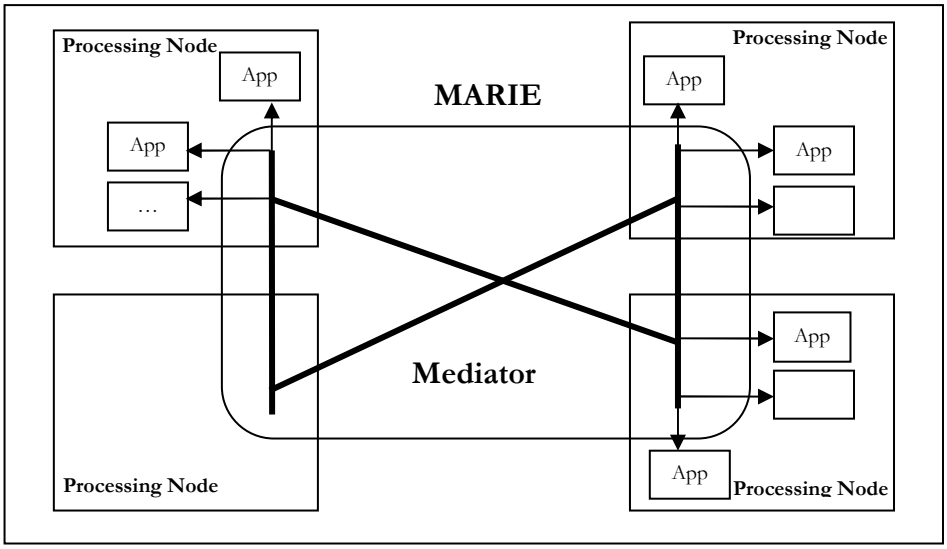

Figure 1.7: Architecture intergicielle de MARIE [Cote 04]

Le médiateur entre en interaction avec toutes les applications d'une manière indépendante, dans l'objectif de coordonner entre elles pour réaliser le système désiré.

Avec l'adoption de ce mécanisme de médiation, MARIE laisse les applications telles quelles, et introduit un faible couplage entre ces applications. Ainsi, les applications ne prennent pas en considération les autres applications présentes dans l'environnement, et limitent ces interactions seulement avec le médiateur. Un autre avantage très important du middleware MARIE est qu'il remplace l'ancien modèle d'interaction plusieurs à plusieurs avec un modèle simple de type un à plusieurs, ce qui implique que chaque interface peut utiliser sa propre mode

de communication une fois que le médiateur sait comment communiquer avec toutes les interfaces de communication.

MARIE définit quatre composants fonctionnels nécessaires pour cette interaction via une unité de contrôle centralisée (Figure 1.8).

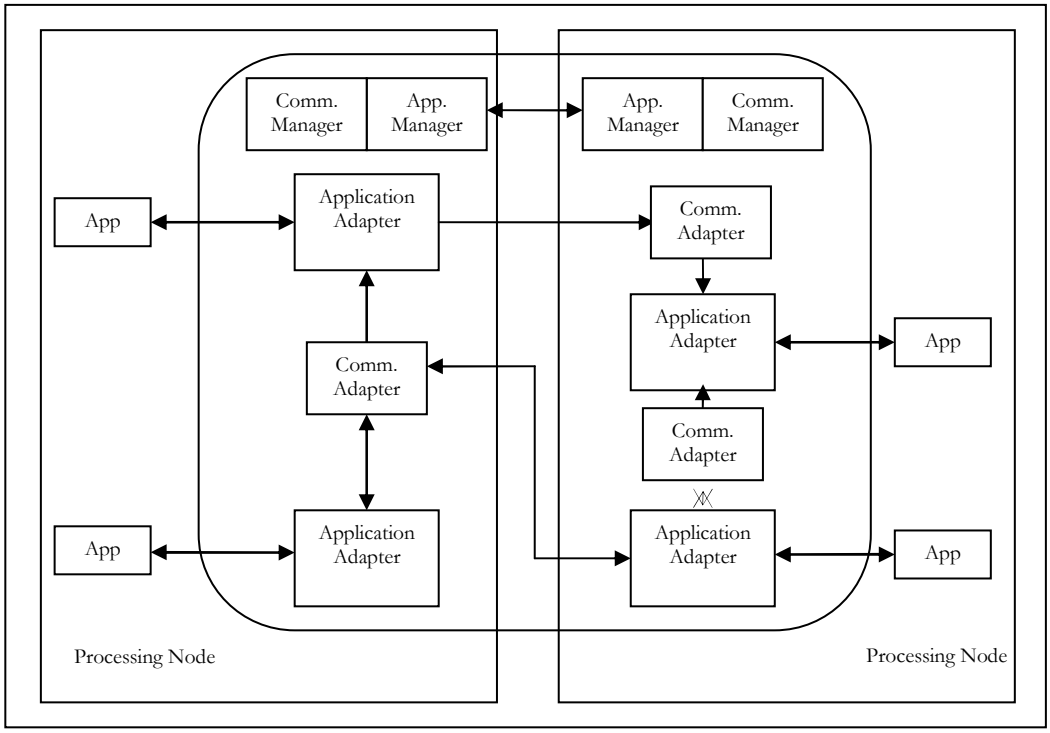

Figure 1.8: Exemple d'un systéme distribué avec le middleware MARIE[Cote 04]

L'adaptateur d'application (*Application Adapter*) est responsable de l'envoi des requêtes de service et de la communication de l'application par l'unité centrale de contrôle, et vice-versa. L'adaptateur de communication (*Communication Adapter*) est responsable du transfert d'information entre différents protocoles et mécanismes de communication. Enfin, les *Communication Managers* sont responsable de la création et de la gestion des liens de communication entre les adaptateurs d'applications qui nécessitent une connexion entre eux **[Cote 04]**.

MARIE a été programmé en C++ et utilise ACE comme un framework générique de communication, ce qui rend l'implémentation de multiples mécanismes et protocoles de communication possible.

Cependant, MARIE présente quelques inconvénients liés à son architecture. Premièrement, les performances du système peuvent être affectés par l'ajout des softwares additionnels. De plus, l'utilisation de plusieurs interfaces de communication hétérogènes influ d'une manière négative sur la cohérence et la stabilité du système. Enfin, chaque application doit implémenter en plus une méthode claire dont le but est l'interaction avec MARIE.

1.5.3.2. Middlewares Robot orienté client serveur (Player / Stage / Gazebo)

L'intergiciel Player/Stage/Gazebo **[Gerkey 03]** est conçu pour être une interface de programmation et un environnement de développement pour robot. Player peut être défini comme un serveur réseau pour le contrôle des robots, il offre aux clients une simple interface pour les capteurs et les actionnaires à travers le protocole réseau TCP.

Dans le côté client, les programmes utilisent l'interface socket pour lire les données depuis les capteurs, et envoient des commandes aux actionnaires, qui peuvent être des commandes de configuration.

Dans Player, un au minimum de contraintes est appliqué sur l'utilisation des dispositifs (capteurs ou actionnaires) ; le fait que les serveurs communiquent par l'intermédiaire d'une interface de socket signifie que les programmes de client peuvent être écrits en utilisant n'importe quel langage qui support l'utilisation de socket. Selon **[Gerkey 03]**, les bibliothèques actuellement disponibles incluent ceux qui sont écrits en C, C++, TCL, python, Java, et LISP commun. En raison de l'utilisation répandue de la communication par socket, le système Player adapte en soi le paradigme de l'informatique répartie. Le code de client peut être utilisé au-dessus de n'importe quel hôte connecté au réseau à n'importe quel endroit. Autres que sa plateforme de base ActivMedia Pioneer 2family, Player support plusieurs robots avec une multitude de capteurs.

Stage qui est un simulateur à deux dimensions, simule une population de robots mobiles dans un environnement 2D. Avec une multitude de capteurs supportés tels que : Sonar, Scanning laser rangefinder, pan-tit-zoom caméra et odometry, il offre l'avantage d'utiliser le code simulé sur des robots réels sans aucune modification.

Gazebo, quand à lui, est un simulateur en trois dimensions de haute fidélité. Similaire dans ces fonctions de base au précédent simulateur Stage.

L'aspect le plus important du middleware Player/Stage/Gazebo est dans la structure de l'infrastructure qui a été développée comme une libre infrastructure software pour robots, et dans le système des capteurs embarqués, ce qui accélère la recherche et le développement dans le secteur des systèmes robotiques.

1.5.4. Middlewares Robot à base de couches

1.5.4.1. CLARAty (*Coupled Layer Architecture for Robotic Autonomy*)

L'intergiciel CLARAty **[Volpe 00]** développé par NASA JPL et CMU est une architecture à deux niveaux (Figure 1.9) qui se positionne comme une évolution des architectures à trois niveaux. Les arguments principaux avancés par les auteurs sont les suivants :

- Les architectures à trois niveaux ont tendance à cantonner et à cloisonner les activités de type décisionnel au plus haut niveau, sans permettre la mise en oeuvre d'algorithmique de ce type au plus bas niveau.
- Il faut un couplage fort entre la couche décisionnelle et la couche fonctionnelle qui peuvent ainsi interagir à tous les niveaux de granularité. Ainsi, l'architecture CLARAty se compose de deux couches :

Un schéma résumant cette architecture est présenté à la Figure 1.9. On remarque seulement deux couches : la couche fonctionnelle (*Functional Layer*) et la couche décisionnelle (*Decisional Layer*).

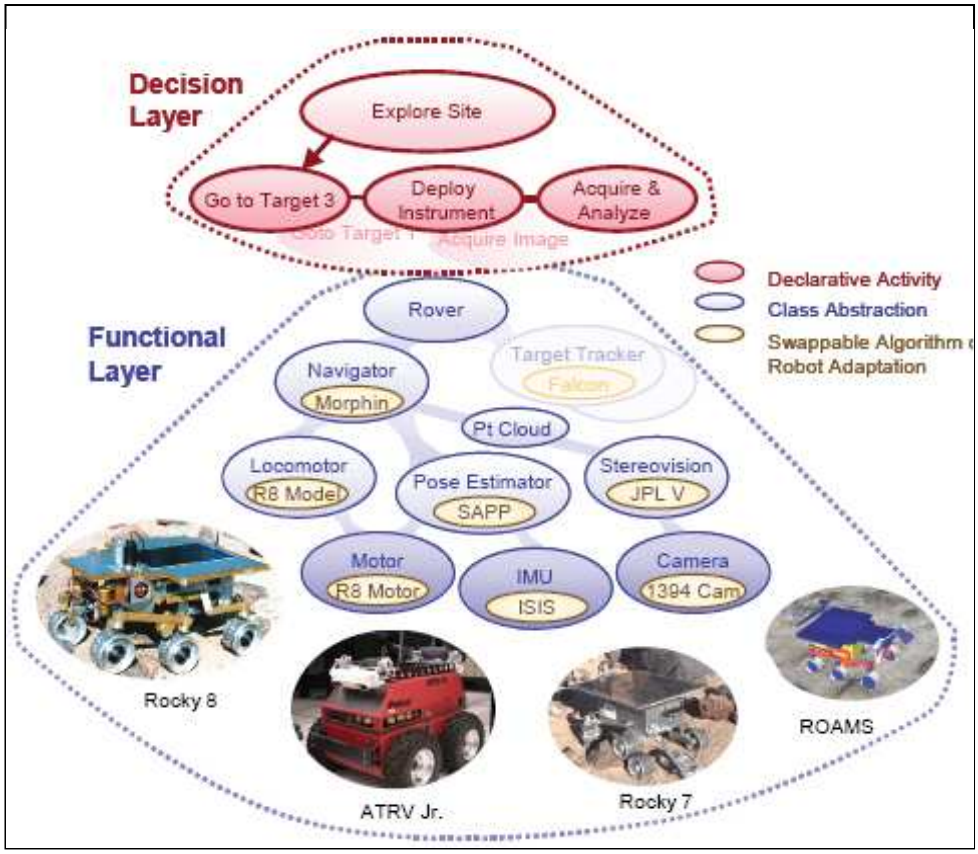

Figure 1.9: Architecture de l'intergiciel CLARAty [Nesnas 03]

La couche fonctionnelle définit une multitude de composantes génériques réutilisables pour interfacer et contrôler divers robots et diverses composantes de systèmes robotisés. Grâce à l'utilisation de patrons de conception pertinents en programmation objet, les composantes définies sont hautement modulaires. La couche fonctionnelle cumule les rôles de planification et d'exécution (deux couches distinctes généralement). La couche décisionnelle considère les ressources du système ainsi que les objectifs de mission pour planifier et exécuter des plans d'activités. L'interaction entre les deux couches se fait par une approche client-serveur et bidirectionnelle et possible à tous les niveaux hiérarchiques.

CLARAty est vu à ce jour comme l'architecture du futur pour les robots de la NASA. Ainsi, elle est devenue en quelques années un passage quasiment obligé et de nombreuses plateformes de la NASA l'utilisent : Rocky 7 et 8, K9 et Fido, ATRV avec des taux de réutilisabilité de l'ordre de 80%.

1.5.4.2. CARMEN (*CARnegie MEllon Robot Navigation Toolkit*)

CARMEN **[Montemerlo 03a, 03b]** est un intergiciel robotique ouvert dont le but est la gestion et le contrôle des robots à travers la fourniture d'une interface cohérente avec ensemble de primitifs de base pour la recherche dans le domaine de la robotique.

CARMEN est une architecture à trois couches, dans laquelle la première couche représente l'interface matérielle, qui fournit le contrôle et l'intégration des capteurs et les données de mouvement, la deuxième couche quand à elle, concerne les tâches de base du robot tels que la navigation, la localisation, le cheminement d'objets, et la planification de mouvement. Enfin, la troisième couche est la couche d'application définie pour l'utilisateur, qui se base sur les primitifs des couches inférieures.

CARMEN est une architecture modulaire basée sur le système de communication interprocessus (*IPC Inter-Process Communication System*). CARMEN supporte un certain nombre de plateformes robotiques telles que (MobileRobots, Nomadic Technologies Scout et XR4000, Segway, robot ATRV, ATRVjr, et B21R) et des primitives de navigation (map-making, Monte-Carlo particle filter localization **[Thrun 00]**. CARMEN fournit également des outils de configuration, un simulateur, et une interface graphique.

IPC **[Simmons 04]** fournit un support de haut niveau pour une connexion entre les processus en utilisant les sockets TCP/IP. Le mécanisme de communication entre ces différents processus peut être de type abonnement/notification ou de type client/serveur. La bibliothèque d'IPC contient des fonctions pour la sérialisation et la désérialisation des données et exécute une fonction semblable à un service de nommage (*naming service*).

1.5.5. Autres Middlewares Robotique

1.5.5.1. ARIA (***A****dvanced* ***R****obotics* ***I****nterface for* ***A****pplications*)

ARIA est un ensemble de classes en C++ qui permettent de décrire la structure d'un robot (ses capteurs, ses actionnaires, et ses caractéristiques physiques) et mettant en accent l'interaction entre les composants logiciels et les composants matériels du bas niveau. Quand au niveau haut, il inclut quelques fonctionnalités d'interprétation, et des actions de base **[LaFary 05]**.

ARIA est un produit de MobileRobots, donc les seuls plateformes qu'il supporte sont celles de MobileRobots. ARIA définit les caractéristiques d'un robot à travers des fichiers de configuration, ces caractéristiques incluent des informations sur le corps de robot, les capteurs (par exemple, le nombre et la position du capteur), et les actionneurs. Les paramètres sont employés par ARIA pour différents calculs (par exemple, le paramètre de RobotRadius est utilisé pour déterminer les limites du tour du robot). ARIA supporte la programmation distribuée et fournit le package Ar-Networking pour la gestion de réseau autour d'une communication basée sur les sockets. En plus, il utilise le générateur d'interface Swig qui est un outil de développement qui supporte l'utilisation de Java et de python.

1.5.5.2. Pyro (***Py****thon* ***ro****botics*)

Pyro est un environnement de programmation de robot, dont le but est de fournir une approche de conception de haut en bas pour les contrôleurs robotiques, il cache les abstractions de bas niveau tout en préservant l'accès **[Blank D]**. Certaines de ces abstractions incluent : les capteurs, les unités de robot, les commandes de mouvement, et d'autres équipements.

Pyro offre aux utilisateurs une enveloppe pour l'utilisation des middlewares Player/Stage et ARIA, dans le sens où n'importe quel composant écrit pour l'une des précédents middlewares peut être directement intégré et utilisé par les utilisateurs de Pyro.

Un grand avantage de Pyro est qu'il supporte un grand nombre de plateformes de robots, y compris K-Team Kheperas et Hemission, MobileRobots Pionner, Handyboard, Sony Aibo, et tous les robots supportés par Player/Stage.

Un module éducatif est disponible pour démontrer les paradigmes de contrôle (par exemple, réseaux neurologiques, algorithmes évolutionnaires, traitement de vision, et machines à état réactif, etc.).

Python, un langage interprété, a été choisi comme base du système à cause de sa facilité d'utilisation pour les étudiants débutants, tout en permettant à des concepteurs plus bien expérimentés d'écrire des codes plus avancés.

La construction de la visualisation graphique des opérations de robot est explicitement supportée par l'utilisation de l'interface OpenGL du Python.

1.5.5.3. MCA2 (*Modular Control Architecture*)

MCA2 est un software avec des capacités temps réel, son application principale est le contrôle des robots autonomes.

Dans MCA2, toutes les méthodes sont réalisées par un simple module avec une interface standard, connecté par un pont de transport de données, le module d'interface est représenté par un simple tableau de valeurs flottantes, le pont ne fait que copier les valeurs depuis l'interface d'entrée vers l'interface de sortie (Figure 1.10).

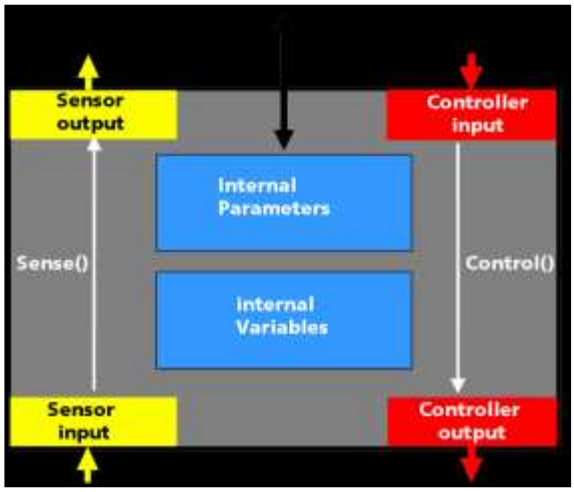

Figure 1.10: Architecture intergicielle MCA2 [Mca2]

Chaque module est structuré dans le même sens, il y a cinq interfaces de communication, quatre connexions via le pont vers les autres modules qui sont arrangés au-dessus ou au-dessous du module, et dans chaque direction, l'on trouve une interface pour l'envoi et la réception des données. Ces interfaces peuvent être interprété comme des vecteurs de données en quelque sorte. Le pont peut prendre n'importe quel vecteur en sortie et le permuté ou le copier dans un autre vecteur d'entrée d'un autre module. Une cinquième interface additionnelle peut être utilisée pour écrire et changer les paramètres d'événement d'un module en exécution.

Chaque module a deux fonctions : capteur *Sense()*et contrôle *Control()*, ces fonctions contiennent les fonctionnalités actuelles du module. Sense() est responsable du processus de capture des données, et Control() est responsable de le contrôler.

Plusieurs applications ont été développées en utilisant MCA2. MCA2 a été utilisé dans plusieurs projets tels que KAIRO et KAIRO II, Airbug, LAURON, ARMan, BISAM, Odete et LTC2.

1.5.5.4. Middleware YARP (*Yet Another Robot Platform*)

YARP (Yet Another Robot Platform), est une plateforme open source pour la robotique humanoid. Le but de YARP est de réduire au minimum l'effort consacré au développement de l'infrastructure logicielle en facilitant la réutilisation de code, la modularité, ainsi maximisant le développement et la collaboration des niveaux de recherche **[Yarp]**.

La réutilisation et l'entretien de code sont donc un défi significatif. En bref, les caractéristiques principales de YARP sont : il supporte la communication interprocessus, le traitement d'image, il contient une hiérarchie de classe pour faciliter la réutilisation de code à travers différentes plateformes matérielles. YARP est actuellement employé et examiné sous Windows et Linux.

1.5.5.5. Autres middlewares

Beaucoup d'autres travaux dans ce domaine d'intergiciels robotiques existent encore. Parmi ces intergiciels on peut encore citer le middleware RAPI (*The Robot Information Framework and Application Program Interface*) dont l'objectif est de définir des méthodes standard pour unifier l'accès au système d'information d'un robot. Le middleware Saphira **[Konolige 97]**, qui est un intergiciel de navigation capable de contrôler des robots allant de *Flakey*, jusqu'au petit Khepera, grâce à un fichier décrivant physiquement le robot à piloter (taille, capteurs, vitesses, etc.). Cet intergiciel est complémentaire au middleware ARIA qui fournit les éléments de base pour l'interprétation des actionnaires et des capteurs, quand au Saphira, il contribue au contrôle de l'architecture.

Enfin, MIRPA-X (*Middleware for Robotic and Process Control Applications*) est un middleware pour robot qui peut être utilisé simplement comme un objet serveur de communication pour mettre en transparence les données et les requêtes pour le système de contrôle de robot **[Diethers 04]**. Il fournit un mécanisme additionnel pour gérer le temps réel.

MIRPA-X fournit un mécanisme de communication basé sur l'usage d'une mémoire partagée, le mécanisme de la mémoire partagée utilise le naming service de MIRPA-X qui garantit une totale transparence pour les processus d'application.

1.5.6. Vers la notion de service robotique

1.5.6.1. Le protocole ROBOLink

ROBOLink est une interface commune et un protocole de communication réseau orienté services Web, spécifié par le constrium Robot Link qui est un groupe de travail au sein du forum robotique ERF (*Entertainment Robot Forum*).

ROBOLink vise à couvrir une large variété de fonctionnalités liées au domaine de la robotique, depuis les fonctionnalités standard de base jusqu'aux fonctionnalités de haut niveau spécifiques à un type particulier. Il vise aussi à la réalisation d'une communication faiblement couplée, incluant un protocole de message asynchrone, et à être déployé sur un grand nombre de standards ouvert d'ordre public pour la communication internet.

Les services fournis par ROBOLink consistent en un groupe de profils de service de base sous la forme de fonctions primitives du robot, et un ensemble de services d'applications telles que les services orientés utilisateur.

Ce protocole offre toutes les interfaces communes nécessaires pour la communication avec d'autres robots et utilisateurs à travers un réseau. Ces interfaces sont : la convention de nommage, la gestion des sessions, la conversation et sécurité. Il offre aussi quatre profils qui sont :

- Le profil de base : utilisé pour démarrer et arrêter une communication avec les robots,
- Le profil de mouvement : utilisé pour bouger le robot (en avant, en arrière, à gauche et à droite) et pour obtenir les informations liées à la position du robot,
- Le profil de danse : utilisé pour faire fonctionner chaque partie de robot,
- Le profil de modèle de mouvement : est utilisé pour charger des modèles de mouvement prédéfinis au robot, d'autres fonctionnalités spécifiques peuvent être ajoutées comme des nouveaux profils (Figure 1.11) **[Makoto]**.

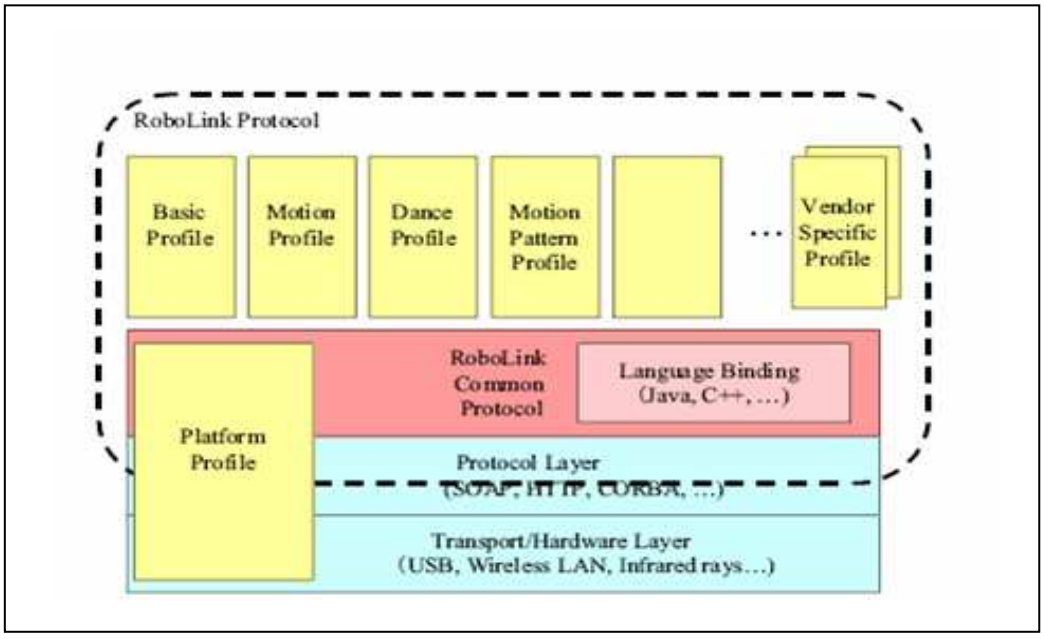

Figure 1.11: Le protocole ROBOLink [Makoto]

Pour une meilleure interopérabilité entre plusieurs robots qui utilisent déférents langages de programmation, le protocole définit une représentation WSDL et une entête SOAP.

1.5.7. Vers des middlewares robotique UPnP

Dans **[Sang 06]**, les auteurs présentent une approche pour utiliser UPnP (*Universal Plug and Play*) comme middleware pour robots dans un environnement distribué dynamique. L'idée de base de cette approche est que le système robotique lui-même peut devenir un environnement informatique distribué dynamique. Dans le futur, chaque composant du robot sera modulaire et sera connecté via un réseau. Il s'agit d'une approche commune de construction d'un système robotique. De plus, la plupart des composants d'un robot seront installés et désinstallés dynamiquement. Ce qui permet de transformer le lui-même système robotique en un environnement dynamique distribué. Par conséquent, un middleware plus approprié que CORBA est nécessaire pour le robot.

La technologie UPnP est la technologie la plus adaptée à ce genre d'environnements distribués et dynamiques, et leurs principales caractéristiques font d'elle la technologie adoptée par la vision précédente. UPnP supporte le déploiement dynamique des entités, et gère des services et pas des objets comme le cas de CORBA, en plus, UPnP présente beaucoup d'autres avantages pour le déploiement des services dans les systèmes embarqués. UPnP supporte les protocoles de communication TCP, UDP, IP, HTTP, SOAP quand TAO de CORBA n'accepte que les deux protocoles GIOP et IIOP, et enfin, UPnP utilise les fichiers XML pour la description de ces services.

Le robot du futur se connectera dynamiquement et communiquera avec le réseau domotique maison ou bureautique. En d'autres termes, un robot aura à communiquer avec l'environnement informatique distribué dynamique externe. Cela constituera une autre exigence pour les middlewares du robot.

1.5.8. Le middleware OSGi et la robotique

L'approche modulaire à base de composants réutilisables est l'une des approches les plus utilisées dans le développement des middlewares robotiques, dans laquelle le problème de dépendance entre les différentes parties d'un middleware est totalement réduit, et le système devient ainsi plus flexible et plus robuste.

L'OSGi (*Open Services Gateway initiative*) qui est un ensemble de spécifications qui définissent un environnement standardisé destiné à faciliter la mise en place de services en réseaux. Ces spécifications concernent les passerelles situées entre un réseau local et l'Internet. Elles définissent : Une architecture logicielle, des services, et la façon dont se déroule l'accès aux dispositifs.

Initialement, la spécification OSGi définit un modèle d'administration d'un parc de passerelles dans lequel l'administration est transparente à l'utilisateur bénéficiaire de services qui lui sont offerts (Figure 1.12). Cependant, la nature dynamique de la spécification OSGi font de cette plateforme un standard ouvert pour beaucoup d'autres domaines tels que dans les futurs smart phones de Nokia, les futures telematics plateformes de BMW, et dans les spécifications GST de l'industrie des véhicules.

a) L'architecture d'OSGi

La plateforme OSGi est une plateforme d'exécution d'applications Java colocalisées au sein de la même machine virtuelle Java **[Gong 01]**. Comme illustre la Figure 1.12, l'architecture d'OSGi peut être définie comme une pile de couches.

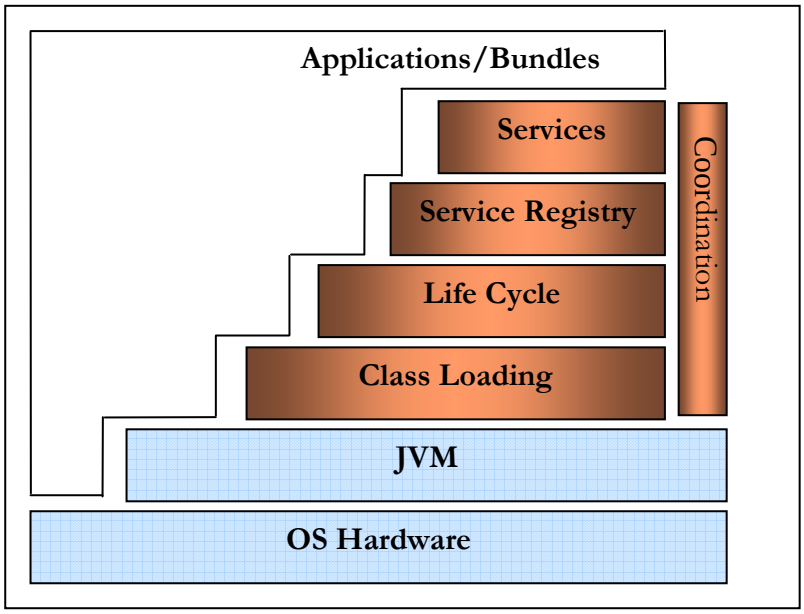

Figure 1.12: Architecture intergicielle d'OSGi [OSGi wp]

La section suivante décrit les couches nécessaires pour le fonctionnement de OSGI.

1. ***JVM*** **:** Les spécifications OSGI sont basées sur la machine virtuelle Java puisque l'environnement Java pourvoit toutes les fonctionnalités requises pour un environnement sécurisé, ouvert, fiable, développable, riche et portatif. L'exécution de plusieurs applications dans la même machine virtuelle crée plusieurs résultats qui peuvent être partagés et doivent être adressés. L'OSGI Framework est le composant responsable de l'adressage de ces résultats.
2. ***Class Loading*** **:** Le contenu d'un Bundle est une archive JAR qui contient des classes qui peuvent être partagés avec d'autres bundles, et permettent ainsi de minimiser la mémoire occupée par les applications ce qui les rend plus flexibles. Un bundle peut rendre disponible plusieurs packages en exportant ces classes, ou utiliser (*importer*) des classes exportées par d'autres bundles.
3. ***Gestion du cycle de vie*** **:** L'accès aux fonctions d'installation d'un bundle se fait via une API qui est disponible pour tous les bundles. L'API de la framework OSGI est définie dans l'objet *"BundleContext"* qui contient plusieurs méthodes pour installer des nouveaux bundles ou pour lister ceux qui existent. Il est fourni au bundle à son démarrage. Durant le cycle de vie d'un bundle, plusieurs événements peuvent avoir lieu dans la JVM tel que la désinstallation d'un bundle qui exporte des classes à celui-ci, cela n'affecte pas le bundle importateur directement puisque les packages exportés par le bundle désinstallé demeurent disponibles tant que des bundles en dépendent.
4. ***Registre des services*** **:** L'avantage qu'a OSGI est sa nature dynamique (un bundle peut soudainement devenir actif sont fourni ainsi de nouvelles fonctionnalités ou coopérer avec d'autres bundles). Un autre avantage est que OSGI n'a pas besoin de structure ou d'environnement statique pour construire de nouvelles applications, mais il est aussi possible d'enlever des parties du système sans que cela ne dérange l'environnement en général. Dans cette perspective, Le registre de services OSGI a été construit pour pallier au problème de la nature dynamique de l'environnement. Grâce à ce registre de services les bundles peuvent enregistrer des objets, consulter le registre pour des objets qu'il cherche, et recevoir des notifications quand un service devient enregistré ou supprimé.

b) Les services OSGi

Dans la plateforme OSGi, les services sont le moyen de coopération entre les bundles. Le contrat de service OSGi est à la fois syntaxique et quantitatif. Il est constitué par une ou plusieurs interfaces java qui servent par la suite à la négociation syntaxique.

Plusieurs services ont été spécifiés par l'Alliance OSGi, parmi ces services on trouve les services du framework, les services systèmes, les services de protocoles, et d'autres services. Les services du framework offrent le service Permission Admin, le service Package Admin, et le service URL handler. Les services systèmes offrent quelques fonctionnalités nécessaires dans plusieurs systèmes comme le service de log, le service Configuration Admin, le service Event Admin, le service User Admin, et le service IO Connector. Les services de protocole ont pour rôle de faire en sorte qu'un protocole comme http soit intégré au sein d'OSGi comme un service. Enfin, d'autres services existent, comme le service Wire Admin, et le service XML Parser.

c) OSGi et la sécurité

Un des buts principaux de la plateforme OSGi est d'exécuter des applications issues de différentes sources sous le contrôle du système d'administration, la plateforme utilise les mécanismes suivants :

- Sécurité du code Java2 : Permissions qui protégent les ressources, comme les permissions sur les classes…etc.

- Minimiser l'exposition du contenu des bundles : Un bundle possède plusieurs permissions qui peuvent être changée à la volée, elles sont attribuées de manière efficace, par exemple si A appelle B et B accède à une ressource protégée, A et B ont besoin d'avoir accès à cette ressource. C'est une stratégie qui permet d'éviter les désordres.

- Communications contrôlées entre les bundles : Par les Services Permissions qui permettent au bundle d'avoir ou d'enregistrer un service à partir du registre de services.

d) OSGi et l'informatique ubiquitaire

L'intergiciel OSGi ouvre d'autres horizons pour l'informatique ubiquitaire en générale, et pour les systèmes robotiques en particulier. L'OSGi offre un framework pour interopérer des capteurs, des actionnaires, et d'autres équipements hétérogènes dans un environnement ubiquitaire, il offre ainsi un middleware pour les applications robotiques, dans les prochaines générations des robots, où le robot lui-même peut être dynamiquement distribué et chaque élément du robot composera un composant à part. En couplant les deux technologies UPnP et l'OSGi framework, le robot sera donc un ensemble de service (bundles au sens OSGi) qui permet aux utilisateurs finaux de contrôler les fonctionnalités du robot comme les fonctionnalités de mouvement (*turnLeft*, *turnRight*, *moveForward*, *moveBack*, *getSensorData*, …). Dans ce sens-là, R-OSGi (*Remoting-OSGi*) est un bon exemple pour le contrôle à distance du robot lego Mindstorms avec quelque bundles.

Dans le travail de **[Shengyuan 04]**, les primitives de base pour contrôler le robot sont implémentées comme différents bundles, et les tâches complexes du robot peuvent être réalisées à travers ces différents bundles par différents mécanismes d'interaction et de coopération entre eux.

1.6. Synthèse

La miniaturisation des puces électroniques, la généralisation de la connectivité avec et sans-fil et le développement de capteurs et actionneurs ouvrent ainsi la voie pour des nouveaux systèmes robotiques dites intelligents et communicants, capables de percevoir et d'agir sur leur environnement, d'offrir des nouveaux services dans des environnements dynamiques, et d'adapter leurs service selon le contexte de l'environnement. Ces systèmes robotiques deviennent de plus en plus complexes, et par conséquent, il apparaît un véritable besoin de technologies logicielles pour des développements optimisés et efficaces. Plusieurs technologies logicielles sont proposées et implémentées, par exemple les architectures basées sur CORBA comme MIRO, OROCOS, et ORCA. Les architectures à couches comme CLARAty et CARMEN. Les techniques de programmation modulaire comme YARP et MCA2. Les bibliothèques communes comme RAPI. Malheureusement, La plupart des middlewares ont été développés indépendamment les uns des autres, et sont guidés par des applications et objectifs spécifiques ce qui rend l'intégration et la réutilisation de leurs composants impossible. Cette problématique a donné naissances à des activités de standardisation dans le domaine de la robotique de service, basé sur les objectifs partagés entre différentes communautés. Autour de cette activité de standardisation plusieurs forums et groupes de travail ont été créés comme le *Robotics Domain Special Interest Group* (*Robotics-DSIG*), l'*Ubiquitous Robotic Companion* (URC) de la Corée, le *Common Platform for Robots* et le *Network Robot Forum* du Japon, l'*Ubiquitous Networking Robotics in Urban Setting* (*URUS*) lancé par un groupe Européen, et enfin l'*IEEE Society off Robotics and Automation's technical Commitee on Networked Robots* des états unies.

Suite à ces efforts, Quelques forums et groupes de travail ont donné déjà leurs fruits, et le groupe Japonais de standardisation (*group of Standardisation Activities on the Robotics Middlewares*) a déjà proposé une spécification de standardisation (*ORiN Open Resource interface for the Network/Open Robot interface for the Network*) qui est un middleware standard offrant une interface de communication standardisée au-dessus d'équipement industriels automatisés. L'AIST à son tour a proposé un autre standard (le *RT-Middleware*) dont l'objectif est de simplifier le développement et l'intégration de systèmes robotisés flexibles.

Malgré les présents efforts de standardisation, on y encore loin d'avoir un middleware standard pour le développement et l'intégration des systèmes robotiques, et d'autres efforts doivent être orientés vers l'adoption et l'intégration des services web dans ces standards.

L'un des points faibles des middlewares présentés jusqu'au là, et l'absence totale d'une interopérabilité sémantique entre ces différents plateforme intergicielle, et la plupart des travaux ont focalisé leurs efforts sur l'aspect de l'interopérabilité de communication, l'on adoptant soient des intergiciels qui facilite ce genre d'interopérabilité comme CORBA, soit des protocoles de communication appropriés dans des environnements hétérogènes. Donc, l'ajout d'une couche sémantique s'avère plus que nécessaire pour les futures middlewares robotiques, ou l'information transmise ou partagée aura un sens sémantiquement compréhensible et partageable par les autres intergiciels.

Chapitre 2 :

Les intergiciels sensibles au contexte

2.1. L'informatique ubiquitaire

Avec l'arrivée de la miniaturisation, de la mobilité et de l'accès sans fil, l'informatique omniprésente et invisible, appelée également informatique ubiquitaire (*Ubiquitous Computing*), est l'un des enjeux de demain. Elle a 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 ou encore la médecine.

Il faut noter que ce type d'informatique propose une approche et un ensemble de services différents par rapport à l'utilisation actuelle que l'humain a à sa disposition pour accéder à l'information. Tout d'abord, la différence essentielle entre l'informatique actuelle et l'informatique ubiquitaire est la mobilité, et par conséquent la prise en compte de l'environnement. Il est nécessaire pour un objet de communiquer avec son environnement afin de pouvoir coopérer avec les objets qui l'entourent pour accéder à l'information. L'utilisateur sera alors pris en compte avec son contexte physique, ce qui permettra de lui offrir les meilleures conditions de service. Le second point qui différencie l'informatique ubiquitaire est l'intégration de l'informatique au service de l'homme par une augmentation de sa capacité d'accès à l'information et non un renforcement de la dépendance de l'homme à sa machine. Pour faciliter cette intégration à l'environnement et aux humains, il est alors nécessaire de rendre les technologies utilisées invisibles autant que faire se peut, soit par une invisibilité réelle via la miniaturisation des technologies, soit par une invisibilité mentale via la familiarisation de l'utilisateur avec l'outil. Finalement, il faut souligner également que l'interface unique telle que l'ordinateur classique avec son écran et son clavier est obsolète dans un tel environnement. En effet, on étend la notion d'interface à tous les objets de l'environnement en vue de prendre en compte leur usage intuitif. Les exemples les plus simples sont l'arrêt de bus qui affiche les horaires et les itinéraires à suivre sur un PDA, ou encore une visite de musée qui permet au passage devant les différentes oeuvres d'obtenir des informations sur l'artiste. L'interface devient alors intelligente et répartie. Elle réagit aux manifestations de l'humain, c'est-à-dire ses déplacements, ses paroles, ses activités ou encore ses habitudes.

Il est alors indiscutable que de nouvelles méthodes de conception d'applications ainsi que de nouvelles infrastructures logicielles sont nécessaires pour permettre la prise en compte des contraintes liées à ces environnements ubiquitaires. Elles doivent alors offrir des garanties de sûreté de fonctionnement et de qualité de service dans leur comportement et dans les services offerts. Cela amène à repenser les infrastructures logicielles pour les rendre adaptables dynamiquement, configurables et auto-administrables en fonction de l'environnement dans lequel elles se trouvent. Ces infrastructures prendront en compte la répartition, l'hétérogénéité, la mobilité mais également les ressources limitées des supports matériels.

2.2. Les intergiciels sensibles au contexte

Les progrès technologiques réalisés ces dernières années dans le domaine des réseaux sans fil, de l'informatique ubiquitaire, et de la miniaturisation ont rendu les terminaux tels que les assistants personnels numériques (*PDAs*) et les téléphones portables suffisamment puissants pour y installer des applications grand public ; Ceci a permis un développement rapide et croissant de l'utilisation de ces terminaux dans la vie de tous les jours. L'environnement de ces terminaux est caractérisé par une variabilité des capacités matérielles (mémoire, bande passante, batterie...), de plus, la mobilité de l'utilisateur introduit une variabilité de l'environnement qui entoure l'application. Par conséquent, les applications traditionnelles ne peuvent s'exécuter dans ce type d'environnement. Donc, il est nécessaire de considérer un nouveau type d'applications qui détectent les changements dans l'environnement et qui tirent parti de ces changements pour adapter leurs comportements en conséquence. Ce type d'applications est appelé applications sensibles au contexte.
Un intergiciel sensible au contexte doit répondre aux propriétés fondamentales suivantes :

- **Interopérabilité** : Jusqu'à présent, les travaux sur l'interopérabilité se sont focalisés essentiellement sur l'hétérogénéité des équipements et des services. Les solutions proposées telles que JINI, UPnP, SOAP ainsi que les différentes techniques de découvertes de services, ont contribué au développement et à la diffusion massive de services pour environnements ubiquitaires, qui se résument à des interactions de base (accès aux fichiers, contrôle d'équipements, etc.). Le développement de services intelligents, autonomes et sensibles au contexte, nécessite de garantir en plus l'interopérabilité au niveau sémantique. Cette dernière doit se traduire pour des services intelligents et hétérogènes par une capacité à échanger et interpréter des connaissances contextuelles de haut niveau.

- **Personnalisation de services** : La personnalisation des services offre aux utilisateurs le moyen de définir et de modifier la manière dont les services leur sont livrés afin de satisfaire leurs préférences.

- **Sensibilité au contexte** : La possibilité d'utiliser des informations sur le contexte est cruciale pour le développement d'applications ubiquitaire adaptatives. Dans le cadre de la fourniture de services, les applications sensibles au contexte sont capables de fournir aux utilisateurs mobiles des services sur mesure qui répondent au mieux aux besoins de l'environnement.

- **Découverte de services sensibles au contexte** : Compte tenu du nombre, de la variabilité et de la diversité des services qui seront disponibles aux utilisateurs mobiles dans les futurs réseaux de communication, il s'avère indispensable de concevoir un mécanisme intelligent, efficace et flexible pour la découverte et la fourniture personnalisées des services aux utilisateurs mobiles.

- **Adaptabilité** : On appelle «adaptation» ou «adaptabilité» d'un système, un changement dans le système pour prendre en compte un changement dans l'environnement du système.

2.3. Le contexte

Le *contexte* n'est pas un concept nouveau en informatique : Dès les années soixante, systèmes d'exploitation, théorie des langages et intelligence artificielle exploitent déjà cette notion. Avec l'émergence de l'informatique ambiante, le terme est redécouvert et placé au coeur des débats sans pour autant faire l'objet d'une définition consensuelle claire et définitive. Toutefois, l'analyse de l'état de l'art conduit à ce double constat.

D'après wikipédia le contexte d'un évènement inclut les circonstances et les conditions qui l'entourent; le contexte d'un mot, d'une phrase, d'un long énoncé ou d'un texte inclut les mots qui l'entourent.

En communication, sociologie, et linguistique, le contexte est l'un des facteurs de la communication, qui influe sur le sens d'un message (comme une phrase) et sur sa relation aux autres parties du message (tel un livre). Il correspond à l'environnement dans lequel la communication a lieu, et à n'importe quelles perceptions de l'environnement général qui peuvent être associées à la communication. Ainsi, le contexte est le *"cadre"* de perception à travers lequel on émet ou on reçoit un message.

Dans les comédies de situation, le contexte est les problématiques et les tendances ambiantes de l'époque où la série se situe.

On trouve également plusieurs définition du terme contexte dans les dictionnaires, tels que le Petit Robert qui définit le contexte comme étant : "*L'ensemble du texte qui entoure un élément de la langue (mot, phrase, fragment d'énoncé) et dont dépend son sens, sa valeur*"ou encore" *ensemble des circonstances dans lesquelles s'insère un fait*". L'encyclopédie Larousse quand a elle définit le contexte comme un "*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*".L' Hachette Multimédia le définit comme un "*ensemble des éléments qui entourent un fait et permettent de le comprendre*". Et enfin, le Grand dictionnaire numérique en ligne donne la définition suivante : "*Le contexte est un énoncé dans lequel figure le terme étudié*" ou encore "*Ensemble d'un texte précédant ou suivant un mot, une phrase, un passage qui éclaire particulièrement la pensée d'un auteur*".

Dans le domaine de recherche informatique. Il existe en fait à peu près autant de définitions de contexte qu'il y a d'équipes de recherche. Toutefois, certaines caractéristiques communes peuvent être retirées.

(Schilit, 1994 ; Schilit et al. 1994) **[Schilit 94]** introduisent l'expression context aware pour désigner un système doté d'un modèle du contexte. Le contexte, selon *Schilit*, inclut la localisation et l'identité des personnes et des objets à proximité ainsi que les modifications pouvant intervenir sur ces objets. Etudier le contexte, c'est répondre aux questions « *Où es-tu ?* », « *Avec qui es-tu ?* », « *De quelles ressources disposes tu à proximité ?* ». Il définit donc le contexte comme les changements de l'environnement physique, de l'utilisateur et des ressources de calcul. Ces idées sont reprises par *Pascoe* **[Pascoe 98]** puis par (*Dey et al*). Un peu plus tard, *Brown* **[Brown 98]** restreint le contexte aux éléments de l'environnement de l'utilisateur, puis il introduit l'heure, la saison, la température, l'identité et la localisation de l'utilisateur.

Parallèlement aux travaux de *Brown*, des définitions émergent avec l'introduction explicite du temps et la notion d'état. (*Ryan et Pascoe, 1997*) assimilent le contexte à l'environnement, l'identité et la localisation de l'utilisateur ainsi que le temps. (*Ward et al, 1997*) **[Ward 97]** voient le contexte comme les états des environnements possibles de l'application. Puis *Dey* **[Dey 00]** précise la nature des entités en question : 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.

Dans le cas des services robotiques, le contexte peut-être définit comme toute information, personne ou un autre objet de l'environnement qui peut jouer un rôle dans la sélection de tel ou tel service intelligent du robot.

2.3.1. Définition de la sensibilité au contexte

On dit qu'un système qu'il est sensible au contexte s'il peut tirer, interpréter et utiliser des informations issues du contexte et adapter sa réponse en fonction du contexte d'utilisation. Cette définition correspond à peu près à la première définition de la sensibilité au contexte proposé par *Shilit* en 1994 **[Schilit 94]**. Depuis beaucoup d'enrichissements ont été apportés à cette définition. On fait maintenant le distinguo entre les applications qui utilisent le contexte, comme par exemple un service de météo qui aura besoin d'informations de localisation et de temps pour produire un bulletin et d'autres applications qui adaptent leur comportement en fonction du contexte. L'utilisation simple du contexte n'implique pas une modification du comportement du système.

Salbert et al **[Salber 98]** définissent la sensibilité au contexte comme étant la meilleure capacité d'un système à agir en temps réel avec des données provenant du contexte. *Dey* limite sa définition à l'interface homme-machine machine sans prendre en compte l'application en elle-même **[Dey 00]**.

D'autres définitions sont plus orientées vers l'adaptation au contexte : *Brown* dit d'une application sensible au contexte qu'elle doit automatiquement extraire de l'information ou effectuer des actions en fonctions du contexte utilisateur détecté par les capteurs **[Brown 98]**.

Enfin *Dey* et *Abowd* **[Dey 00]** proposent qu'un système soit sensible au contexte s'*il utilise des informations du contexte pour mettre à disposition des informations ou des services utiles à l'utilisateur*, l'utilité dépendant de la tâche de l'utilisateur. Avec cette définition, une application qui se contenterait d'afficher à l'écran le contenu des variables de contexte est une application sensible au contexte légitime, même si elle ne modifie pas son comportement.

Dans le cas de service robotique, un système est sensible au contexte s'il utlise le contexte pour sélectionner un service intelligent d'un robot parmi plusieurs.

2.3.2. Capture des Informations Contextuelles

Il existe trois types de capteurs pour capturer les informations contextuelles :

- Les capteurs physiques : Les capteurs les plus souvent utilisés sont des capteurs physiques. La plupart des capteurs physiques disponibles sont capables de capturer presque n'importe quelles données physiques. Le tableau 2.1 montre quelques exemples de capteurs physiques.
- Les capteurs virtuels : Les capteurs virtuels capturent le contexte à partir d'applications logiciel et des services. Par exemple, il est possible de déterminer l'endroit des employés non seulement en employant des systèmes de piste (*tracking*) (capteurs physiques) mais également par un capteur virtuel, par exemple, en explorant un calendrier électronique, un système de travel-booking, les email etc., pour des informations de localisation. D'autres attributs de contexte peuvent être captés par les capteurs virtuelles incluent, par exemple, l'activité de l'utilisateur par analyse des mouvements de la souris et les interactions claviers. Les capteurs logiques : Ces capteurs se servent d'un couple des sources d'informations, et combinent les capteurs physiques et virtuels avec l'information additionnelle des bases de données ou d'autres sources afin de résoudre des tâches plus élevées. Par exemple, un capteur logique peut être construit pour détecter la position actuelle des employés en analysant les logins aux ordinateurs de bureau.

Type of context	Available sensors
Light	Photodiodes, colour sensors, IR and UV-sensors etc.
Visual context	Various cameras
Audio	Microphones
Motion,	Mercury switches, angular sensors,
acceleration	Accelerometers, motion detectors, magnetic fields
Location	Outdoor : Global Positioning System (GPS), Global System for Mobile Communications (GSM) ; Indoor : Active Badge system, etc.
Touch	Touch sensors implemented in mobile devices
Temperature	Thermometers
Physical attributes	Biosensors to measure skin resistance, blood pressure

Tableau 2-A: Quelques capteurs physiques [Schmidt, v Laerhoven 01]

2.3.3. Caractéristiques des informations de contexte

Les informations de contexte sont des informations collectées à partir de plusieurs sources hétérogènes. De ce fait, elles ont des caractéristiques variables. Chaque contexte observable peut être statique ou dynamique. Les observables statiques, représentent des informations qui ne changent pas au cours du temps. Il suffit de les collecter au lancement de l'application. Parmi les observables statiques, nous avons la taille de l'écran ou le profil d'un utilisateur. Les observables dynamiques représentent des informations dont les valeurs changent fréquemment, une observation de leur état peut devenir très rapidement obsolète.

Les observations de contexte effectuées à partir de capteurs physiques peuvent être sujets à des bruitages ou des erreurs de capture. Donc, ces observations sont incorrectes lorsqu'elles ne reflètent pas l'état réel de l'environnement, incohérentes lorsqu'elles contiennent des informations contradictoires et incomplètes lorsque certains aspects du contexte ne sont pas renseignés.

Afin d'utiliser les informations de contexte dans des applications informatiques, il est nécessaire de connaître pour chaque observable les caractéristiques citées ci-dessus. De plus, il est très important d'avoir une description abstraite de ces informations afin de pouvoir les utiliser.

2.3.4. Le réseau de contextes

Selon **[Gaëtan Rey 05]**, un réseau de contextes *Rc*, est défini sur trois ensembles (*Roles*, *Relations*, *Entites*) et un prédicat (*joueRole*):

- Roles est l'ensemble des rôles considérés par le concepteur du système interactif. Roles = $\{r_1, r_2, ..., r_n\}$ où r_i est un rôle ;
- Relations correspondent à l'ensemble des relations entre les entités considérées par le concepteur du système interactif. Relations= $\{rel_1, rel_2, \ldots, rel_n\}$ où rel_i est une relation ;
- Entites désigne l'ensemble des entités considérées par le concepteur du système interactif. Entites = $\{e_1, e_2, ... e_n\}$ où e_i est une entité ;
- JoueRole(e,r) est un prédicat qui se vérifie si et seulement si l'entité e joue le rôle r avec $e \in$ entites et $r \in$ Roles ;

Chaque nœud du réseau R_c correspond à un contexte C_i. Un contexte C_i est défini par le couple (r_i, rel_i) ou :

- r_i est l'ensemble des rôles joués par les entités, avec $r_i \subseteq$ Roles. De plus, quel que soit $r \in r_i$,il existe une entité $e \in$ Entites telle que e joue le rôle r (propriété1).

$$\forall (r \in R_i), \exists (e \in Entites) / \ joueRole(e, r) \qquad (1)$$

- rel_i est l'ensemble des relations entre entités, avec $rel_i \subseteq$ Relations. De plus, quelle que soit la relation rel (d'arité j) $\in rel_i$, il existe j entités e_1 à $e_j \in$ Entites tel que les entités e_1 à e_j entretiennent la relation rel (propriété 2).

$$\forall ((rel \in \mathrm{Re}\, l_i), \vee (arity(rel) = i)) / \exists (e_1 ... e_j \in Entites) / rel(e_1 ... e_j) \qquad (2)$$

De ces définitions et propriétés, on déduit qu'il y a changement de contexte lorsque l'une des conditions suivantes devient vraie :

– l'ensemble R_i des rôles remplis change : apparition de rôles et/ou disparition de rôles ;

– l'ensemble Rel_i des relations entretenues change : apparition de nouvelles relations et/ou disparition de relations.

Ce qui s'écrit formellement :

$$\forall ((C_1 = (R_1, \mathrm{Re}\, l_1)) \in Rc), \vee ((C_2 = (R_2, \mathrm{Re}\, l_2)) \in Rc) /$$
$$(C_1 = C_2) \Leftrightarrow ((R_1 = R_2) \wedge (\mathrm{Re}\, l_1 = \mathrm{Re}\, l_2)) \qquad (3)$$

Le contexte C_i, noeud du réseau de contexte Rc, respecte les conditions suivantes :

$$\forall (C_1 \in Rc) \exists R_i, \mathrm{Re}\, l_i /$$
$$(C_i = (R_i, \mathrm{Re}\, l_i)) \Leftrightarrow (R_i \in P(Roles)) \wedge (\mathrm{Re}\, l_i \in P(\mathrm{Re}\, lations)) \qquad (4)$$

Où P (Roles) et P(Relations) sont, respectivement, l'ensemble des parties de Roles et l'ensemble des parties de Relations.

La cardinalité de R_c est le produit des cardinalités de P(Roles) et de P(Relations). Soit n la cardinalité de Roles et m la cardinalité de Relations, les formules suivantes détaillent le calcul de la cardinalité de R_c :

$$Card(P(Roles)) = \sum_{k=0}^{n} C_n^k = \sum_{k=0}^{n} \frac{n(n-1)...(n-k+1)}{k!} = 2^n \quad (5)$$

$$Card(P(relations)) = \sum_{k=0}^{m} C_m^k = 2^m \quad (6)$$

$$Card(Rc) = Card(P(Roles)) \times Card(P(relations)) = 2^n \times 2^m = 2^{n+m} \quad (7)$$

Avec $C_n^k = (n! \times k!)/(n-k)!$

Cette information, facilement calculable, permet aux concepteurs de systèmes interactifs de détecter dans leur analyse, l'oubli de contextes, oublis pouvant entraîner des comportements anormaux de la part du système. Chaque contexte est à son tour, un réseau de situations.

2.3.5. Contextes et situations

Selon **[Gaëtan Rey 05]**, Un contexte *C* = (*R*,*Rel*) est un réseau de situations tel que toutes les situations de *C* partagent les mêmes ensembles de rôles *R* et de relations *Rel* où $R \subseteq$ Roles et *Rel* $\subseteq$ Relations.

Au sein d'un contexte *C* = (*R*, *Rel*), une situation *S* est définie sur trois ensembles (*Ent*, *AssoRoEnt*, *AssoRelEnt*) et un prédicat *estPresent* :

- Ent est l'ensemble des entités présentes dans *S*, avec *Ent*$\subseteq$ Entites,

$$\forall(e \in Ent)/((e \in Entite) \wedge est\,\mathrm{Pr}esent(e)) \quad (8)$$

Où le prédicat *estPresent*(*e*) est vrai si et seulement si l'entité *e* est présente dans *S* ;

- AssoRoEnt est l'ensemble des associations entre les rôles de *R* et les entités de *Ent*. Rappelons que chaque rôle r_i est joué par au moins une entité et que chaque entité joue zéro, un ou plusieurs rôles ;

$$\forall(a \in AssoRoEnt), \exists(e \in Ent \wedge e \in R)/\, a = (e,r) \wedge jouerole(e,r) \quad (9)$$

- AssoRelEnt est l'ensemble des associations entre les relations de *Rel* et les entités de *Ent* ;

$$\forall(b \in Asso\,\mathrm{Re}lEnt), \exists(e_1,...,e_j \in Ent \wedge rel \in \mathrm{Re}l \wedge arity(rel) = j)/ \\ b = (rel, e_1,...e_j) \wedge rel(e_1,...e_j) \quad (10)$$

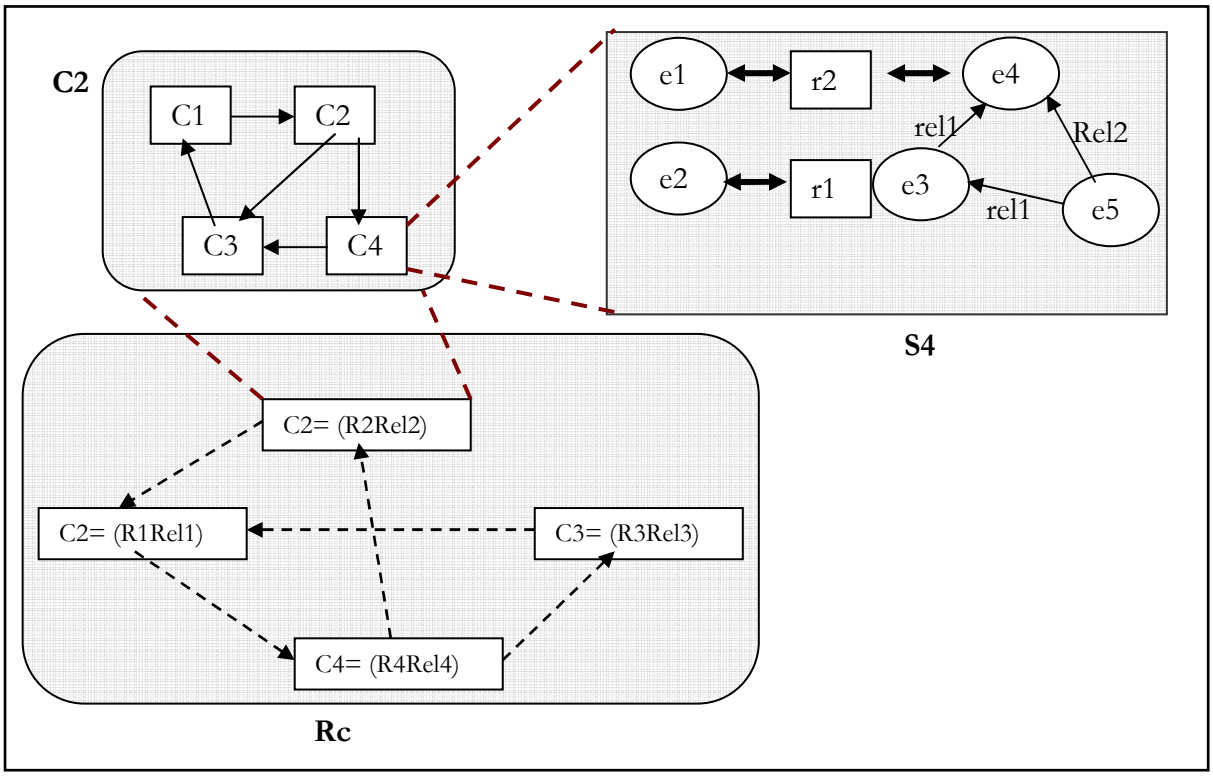

Figure 2.1: Décomposition d'un réseau de contextes en contextes et situations [Gaëtan Rey 05]

Au sein d'un contexte $C = (R, Rel)$, il y a changement de situation (figure 2.2) si l'une au moins des conditions suivantes est remplie :

- L'ensemble *Ent* des entités change : apparition et/ou disparition d'entité(s) ;
- L'ensemble *AssoRoEnt* des associations entre les rôles et les entités change : apparition et/ou disparition d'association(s) rôle-entité ;
- L'ensemble *AssoRelEnt* des associations entre les relations et les entités change : apparition et/ou disparition d'association(s) relation-entités.

La figure 2.2 illustre graphiquement les transitions entre situations.

De manière plus formelle, une situation S se définit comme suit :

Soit $C = (R, \mathrm{Re}l), R \subseteq Roles, \mathrm{Re}l \subseteq relations$ alors:

$$\forall (S \in C = (R, \mathrm{Re}l)), \exists Ent, AssoRoEnt, Asso\,\mathrm{Re}lEnt /$$
$$S = (Ent, AssoRoEnt, Asso\,\mathrm{Re}lEnt) \Leftrightarrow (Ent \in P(Entites)) \qquad (11)$$
$$\wedge (AssoRoEnt = f(Ent, R)) \wedge (Asso\,\mathrm{Re}lEnt = g(Ent, \mathrm{Re}l))$$

Où *Ent* est l'ensemble des parties de Entites, *f* est la fonction d'association entre une entité *e* et un rôle *r* telle que *e* joue *r* et *g* la fonction d'association entre une ou plusieurs entités e_1 ... e_j et une relation *rel* telle que *rel*(e_1, ..., e_j) est vraie.

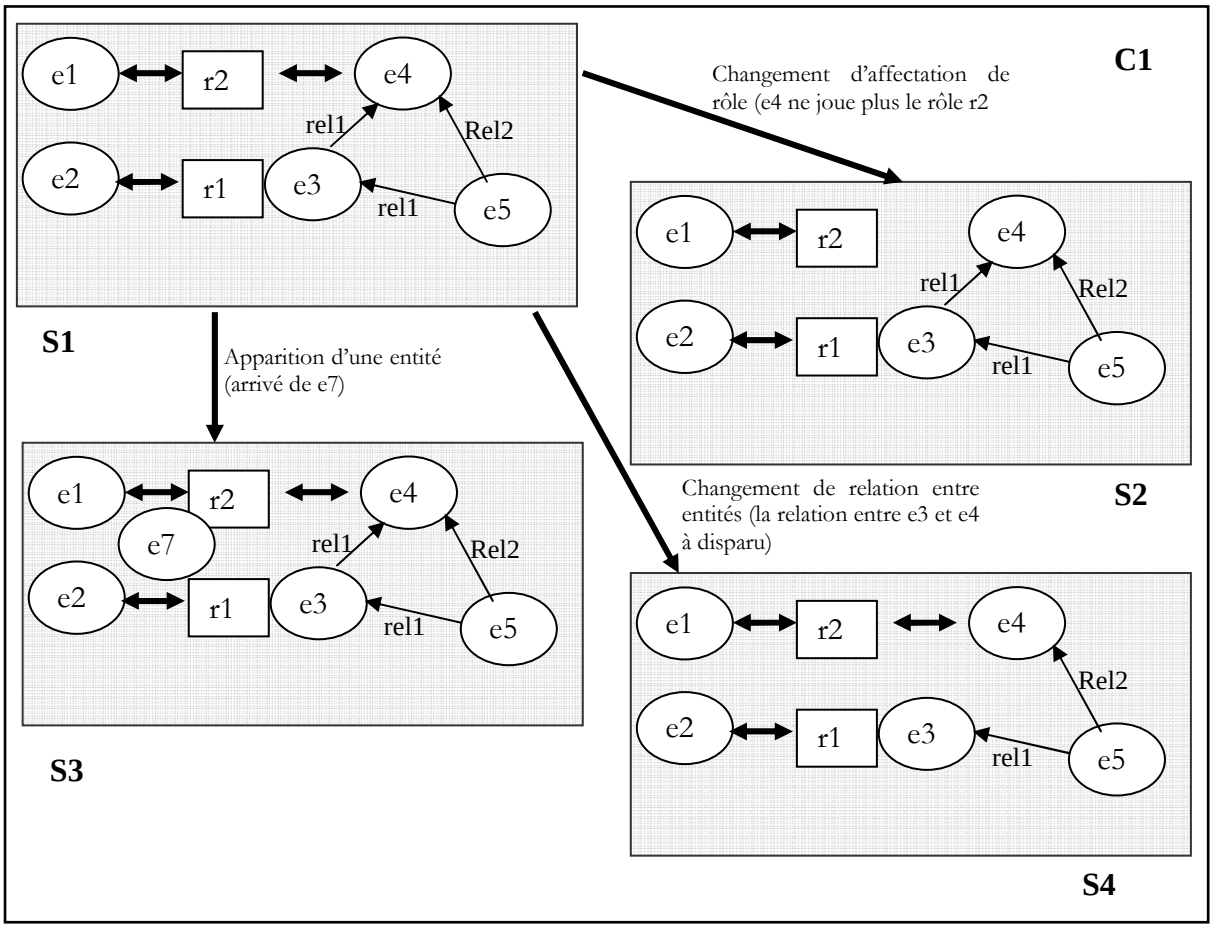

Figure 2.2: Changement de situation [Gaëtan Rey 05]

De plus, la propriété suivante est vérifiée quelle que soit la situation *S* :

$$\begin{aligned}&\forall(S_1 = (Ent_1, AssoRoEnt_1, Asso\,\mathrm{Re}\,lEnt_1) \in C),\\&\forall(S_2 = (Ent_2, AssoRoEnt_2, Asso\,\mathrm{Re}\,lEnt_2) \in C)/\\&(S_1 = S_2) \Leftrightarrow (Ent_1 = Ent_2) \wedge (AssoRoEnt_1 = AssoRoEnt_2)\\&\wedge' Asso\,\mathrm{Re}\,lEnt_1 = Asso\,\mathrm{Re}\,lEnt_2)\end{aligned} \qquad (12)$$

2.3.6. Etapes de modélisation du contexte

Phase 1 : Définition du domaine du contexte d'interaction.

La première étape consiste à définir le domaine du contexte d'interaction et donc des trois ensembles Entités, Rôles et Relations.

Phase 2 : Calcul du réseau de contextes.

Ayant défini les ensembles de base, il faut calculer le nombre de contextes qui composent le réseau Rc.

Phase 3 : Simplification du réseau de contextes.

Cette phase de simplification permet d'une part, de fusionner des contextes similaires pour le système en cours de conception, et d'autre part, de vérifier l'utilité des rôles et relations précédemment établis. Il est en effet essentiel de simplifier au maximum le nombre de contextes de manière à répondre le plus clairement possible au problème tout en levant les ambiguïtés que pourraient provoquer deux contextes identiques pour le système.

Phase 4 et 5 : Détail des contextes importants en situations.

Si le nombre de contextes est fini, celui des situations composant un contexte est potentiellement infini. De la même manière qu'il est recommandé de simplifier les contextes, il convient de vérifier si parmi les situations détaillées, certaines ne sont pas superflues et pourraient perturber le bon fonctionnement du système.

Phase 6 et 7 : Description des entités mises en jeu et association des observables à un composant de capture.

Une fois les contextes et les situations identifiés, il ne reste plus qu'à décrire les entités mises en jeu en termes d'observables. Une entité est en fait un regroupement d'observables qui sont des données qui pourront être capturées, mesurées, calculées par des capteurs ou autres composants de capture.

2.3.7. Modélisation des informations de contexte

Pour décrire la sensibilité d'une application à son contexte d'exécution, il faut déterminer les contextes auxquels cette application est sensible et les décrire dans un modèle. Par conséquent, la modélisation du contexte est la première étape dans le processus de création d'applications sensibles au contexte. Cette modélisation permet à l'application et/ou aux intergiciels de faciliter l'interaction avec le contexte en fournissant une description abstraite des observables. La diversité des informations de contexte et leur utilisation dans divers domaines engendrent différentes façons de les modéliser.

Dans cette section, nous classifions la modélisation du contexte en quatre types d'approches : L'approche paires/triplets, l'approche orientée modèles, l'approche basée sur la logique et l'approche orientée ontologie. Cette classification se base sur la façon dont le contexte est modélisé, les outils utilisés à cet effet, l'expressivité du modèle et la possibilité de le réutiliser.

a) Approches paires/triplets :

Les approches *paires/triplets* permettent de modéliser les valeurs observées du contexte ; Elles utilisent à cet effet des structures de données très simples pour les décrire. Les premiers travaux visant à décrire le contexte sont ceux de *Schilit*, *Theimer* et *Welsh* **[STW 93]**. Ils proposent d'utiliser un serveur d'environnement dynamique qui se charge d'observer un ensemble de contextes pour le compte d'une application. Chaque observation du contexte est sauvegardée dans une variable d'environnement constituée d'une paire *clés/valeurs*. Une variable d'environnement qui permet de décrire les imprimantes les plus proches est la suivante :

Nearest_Printer=HP:HP4:HP6

La modélisation *paires/triplets* utilise des structures de données simples à gérer. Par contre, cela ne permet pas une description complète du contexte, ni l'expression des relations qui peuvent exister entre les informations de contexte ou la manière de déduire un contexte de haut niveau. De plus, cela décrit une observation et non un observable. Ce type de modélisation ne

prend pas en compte la description des actions d'adaptation ni les conditions qui permettent de les déclencher.

b) 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 de permettre sa réutilisation. On trouve *Unified Modeling Language* (UML) : *Sheng* et *Benatallah* **[Sheng 05]** ont proposé un méta-modèle basé sur une extension d'UML qui permet de modéliser le contexte auquel des services web sont sensibles. Ce langage est appelé *ContextUML*.

- **Context Modeling Langage (CML**) : Afin de pouvoir modéliser les caractéristiques des informations de contexte et leurs propriétés, *Henricksen et al* **[Henricksen 02] [Henricksen 06]** ont proposé un langage orienté objet dérivé de l'ORM (*Object Role Modeling*) **[Henricksen 05]**. Ce langage appelé CML (*Context Modeling Language*) **[Henricksen 04]** qui permet de modéliser le contexte auquel une application est sensible d'une manière formelle. Un outil graphique assiste le concepteur d'applications dans la tâche de description du contexte auquel son application est sensible. Il lui offre un moyen de décrire les caractéristiques des informations de contexte (capturé, statique, dérivé ou information de profil) et les dépendances entre ces informations. CML permet aussi de spécifier la qualité de chaque information observée et sa validité temporelle.

- **Utilisation d'un langage de balises pour profil** : Le point commun entre toutes les approches qui utilisent un langage de balises est la structure hiérarchique des données modélisées. Cette modélisation consiste en un ensemble de balises avec des attributs. Dans XML (*eXtensible Markup Language*), les balises sont définies dans une DTD (*Document Type Definition*).

Cette modélisation, bien qu'elle soit efficace pour certains types d'applications, ne définit qu'un format d'échange de données. Donc, elle ne permet pas de décrire les contextes interprétés ni le profil de l'utilisateur. De plus, aucune description des relations entre les informations de contexte n'est prévue. Du fait de sa grammaire DTD figée, elle est spécifique à un ensemble limité d'applications.

- **Utilisation d'un langage de balises pour l'adaptation** : *Capra et al* **[Capra 01]** utilisent le langage XML afin de modéliser le profil de l'utilisateur ainsi que le profil d'une application. Cette modélisation est utilisée par un intergiciel réflexif appelé CARISMA **[Capra 00]** afin de reconfigurer une application ou de l'adapter aux changements du contexte.

Les approches orientées modèle utilisent un modèle formel pour décrire le contexte. Ce type d'approche permet de décrire le contexte de manière plus riche que les approches paires/triplets. De plus, cela offre aux concepteurs d'applications la possibilité de réutiliser le modèle pour d'autres applications. Les modélisations basées sur un langage de balises permettent de décrire des profils ou des contextes simples, sans offrir la possibilité de décrire des relations de dérivation et de dépendance entre ces informations. Les modélisations existantes basées sur UML permettent de décrire des relations entre les informations de contexte mais ne prennent pas en compte la description des dépendances entre ces informations, ni la qualité ou la validité temporelle des données décrites. CML est venu remédier à certains de ces manques en proposant un modèle avec visualisation graphique qui permet de typer le contexte, et de décrire un ensemble de relations entre plusieurs contextes observables. Le langage CA-IDL est un langage qui permet de décrire non seulement les situations pertinentes d'une application mais aussi les actions d'adaptation et les conditions de leur déclenchement. Ces conditions représentent des expressions régulières entre les situations pertinentes. Pour cela, CA-IDL offre une grammaire qui permet de décrire la sensibilité au contexte, mais cette grammaire reste limitée dans la mesure

où elle n'offre pas la possibilité d'ajouter de nouvelles sources de contexte et de nouveaux observables sans modifier la grammaire de CA-IDL, ni le moyen de décrire l'interprétation d'un contexte de haut niveau.

c) Approches basées sur la logique

Les modèles basés sur la logique sont caractérisés par un très grand degré de formalité. Ils utilisent l'algèbre booléenne et la logique du premier ordre pour modéliser le contexte. La logique permet de définir des conditions qui nécessitent de déduire des faits ou des expressions à partir d'un autre ensemble d'expressions ou de faits. Par conséquent, dans les modèles basés sur la logique, le contexte est défini comme des faits, des expressions ou des règles.

Les approches basées sur la logique utilisent l'algèbre booléenne ou la logique du premier ordre pour modéliser le contexte dans le but de raisonner sur les informations collectées. Cette modélisation ne permet pas de décrire la validité temporelle des informations ni les relations qui peuvent exister entre les informations de contexte, mais elle est très efficace pour raisonner sur le contexte et déduire des actions de réaction si une situation pertinente est détectée. L'approche basée sur la logique peut être utilisée dans l'informatique sensible au contexte afin d'intégrer et d'interpréter les données collectées.

d) Approches orientées ontologie

Une *ontologie* est une description sémantique, structurée et formelle des concepts d'un domaine et de leurs inter-relations **[Uschold 96]**. Plusieurs modèles d'ontologies ont été proposés pour décrire le contexte. Ces approches permettent non seulement de modéliser le contexte, mais aussi de raisonner sur les données décrites. Les approches orientées ontologies sont caractérisées par une possibilité d'extension et de partage des données.

Comme exemple d'ontologie on trouve : CoBrA-ONT (*Context Broker Architecture ONTology*) **[Chen 04a]** qui est une ontologie écrite en utilisant le langage OWL-DL. Elle permet la création d'applications multi agents sensibles au contexte. Cette ontologie a été créée pour décrire le contexte que les applications de maison intelligente multi agents utilisent **[Chen 04b]**. CoBrA-ONT est constituée de quatre sous-ontologies qui permettent de décrire des informations de localisation en général, des informations d'agents (des personnes ou des agents logiciels), la localisation de ces agents et leurs activités.

2.4. Les infrastructures intergiciels de gestion du contexte

L'architecture de référence d'une infrastructure intergicielle de gestion du contexte dans un environnement ubiquitaire est composée d'entités logicielles offrant six fonctions principales (Figure 2.3) : (i) acquisition (on parle également de collecte ou de capture) de données contextuelles, (ii) traitement de ces données (iii) fourniture des informations contextuelles aux applications et aux utilisateurs, (iv) découverte des fournisseurs de données contextuelles disponibles (v) contrôle d'accès aux fournisseurs de contexte et respect de la vie privé des utilisateurs (vi) gestion de l'historique du contexte **[Chibani 06]**.

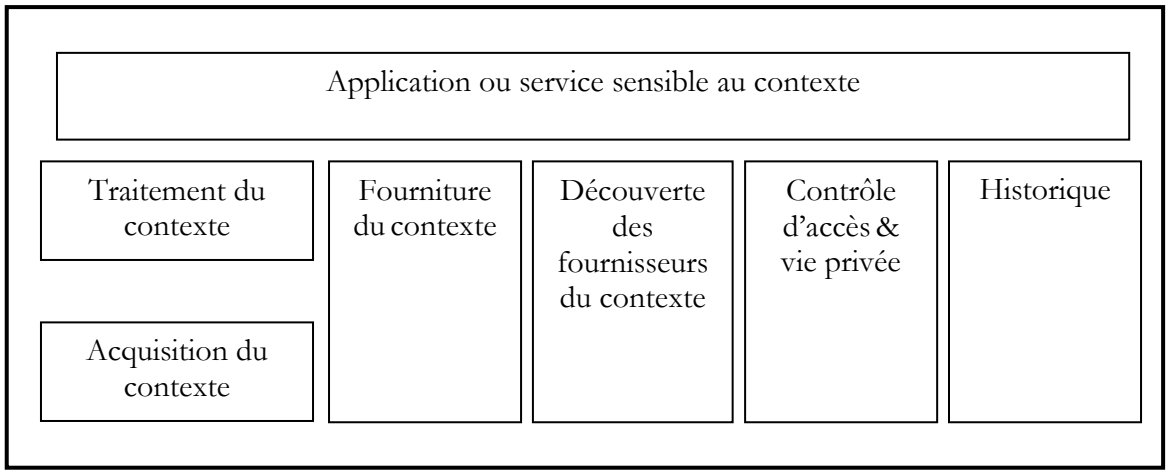

Figure 2.3: Architecture de référence d'une infrastructure intergiciel de gestion du contexte [Chibani 06]

Les applications qui utilisent la notion de contexte sont de plus en plus nombreuses. Devant la multiplication et les avancées logicielles du domaine, des équipes de recherche se sont intéressées à la définition d'une architecture commune pour les applications contextuelles, celles ci devant permettre l'interopérabilité et l'intégration de contextes de plus en plus riches sans pour autant avoir à reprendre toute l'architecture du logiciel de zéro.

2.4.1. CAMidO

CAMidO (*Context-Aware Middleware based on Ontology meta-model)*, est un intergiciel orienté composants sensible au contexte. CAMidO offre un métamodèle basé sur une ontologie pour décrire le contexte et déduire de nouvelles règles d'adaptation. Cette description permet non seulement de collecter le contexte et de détecter ses changements pertinents mais aussi d'automatiser le processus d'adaptation en générant des contrôleurs au niveau de chaque conteneur associé à un composant sensible au contexte.

Cet intergiciel, implémenté au-dessus de CCM **[CCM 02]**, offre un méta-modèle d'ontologie qui permet aux concepteurs de décrire le contexte ainsi que les politiques d'adaptation que l'intergiciel doit prendre en compte afin d'automatiser le processus de sensibilité au contexte. CAMidO prend en compte deux types d'adaptations : L'adaptation réactive qui consiste à rester à l'écoute du changement de l'environnement et à lancer la politique d'adaptation appropriée, quand l'intergiciel détecte une valeur pertinente du contexte, et l'adaptation pro-active qui concerne les appels des opérations offertes par les composants de l'application et qui consiste à anticiper l'adaptation du comportement de ces appels quand le contexte prend des valeurs pertinentes.

Le méta-modèle d'ontologie qu'offre CAMidO a été implémenté, ainsi que les entités de gestion du contexte, le compilateur CAMidO quand à lui est en cours de développement. Les règles d'adaptation décrites par le concepteur et celles qui sont générés par le compilateur CAMidO peuvent générer différents types de conflits. Les conflits intra et inter composants peuvent être résolus en ajoutant des priorités aux opérations, alors que les conflits intra-applications nécessitent un mécanisme de consensus.

En conclusion, CAMidO propose la construction dynamique du contexte par observation de l'environnement. Les adaptations peuvent être liés à chaque composant dans le mode réactif

ou définis sur un assemblage de composants dans le mode proactif. Elles définissent implicitement des modes d'exécution. Enfin, à l'exécution, un intergiciel à base de composants permet de gérer les contextes et les adaptations.

2.4.2. Context Toolkit

Le Context Toolkit est une infrastructure de type flot de données fondée sur cinq catégories de composants logiciels (figure 2.4) et deux types de composants matériels. Les *Context-Widgets* servent de lien entre les capteurs physiques et les applications, tout comme les Widgets graphiques servent d'intermédiaires entre l'utilisateur et le système interactif **[Salber 99] [Dey 01]**. Les *Context-Aggregators*, qui concatènent les données des Context-Widgets en une unité d'abstraction, fournissent aux applications une vue simplifiée de l'espace d'informations contextuelles. La liaison entre Context-Widgets, les Context-Aggregators et les applications se fait grâce au *Discoverer*. Le Discoverer maintient la liste des composants disponibles. Il peut être interrogé par les applications cherchant tels ou tels composants et/ou services. Maintenant que les applications reçoivent les données enregistrées par les capteurs via les Context-Widgets et les Context-Aggregators il ne reste plus qu'à donner un sens à ces données. Pour cela, les Context-Widgets, comme les applications, peuvent interroger des *Context-Interpreters*. Enfin, les *Services* permettent d'exécuter des actions (via les *Actuators*) au nom de l'application.

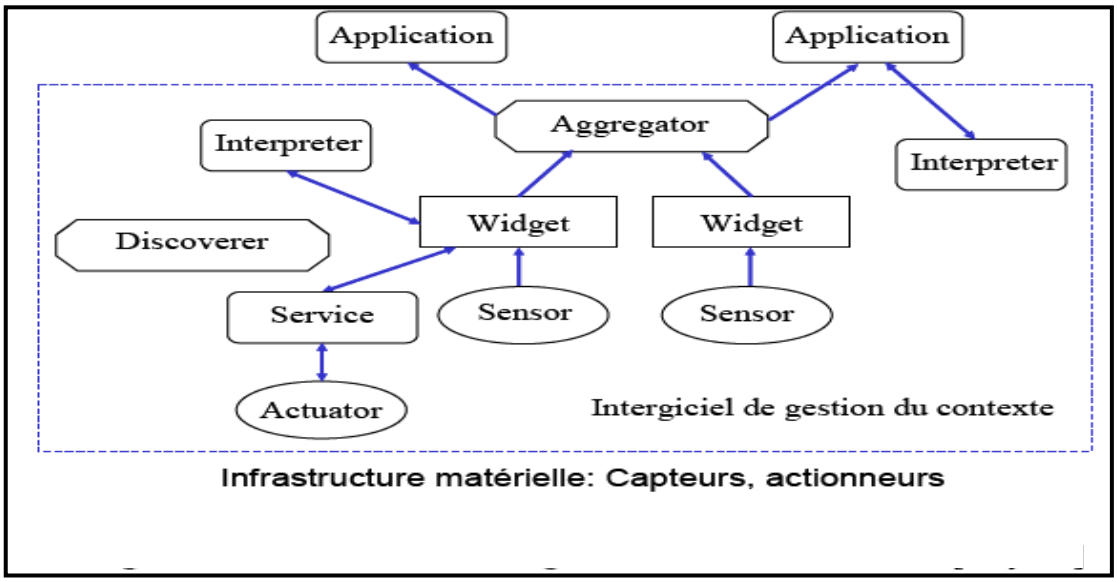

Figure 2.4: Architecture intergicielle de Context Toolkit [Dey 01]

Cette infrastructure, bien adaptée à la distribution des informations contextuelles, présente cependant quelques limitations :1) Elle ne prend pas explicitement en charge les métadonnées censées renseigner l'application sur la qualité et/ou la précision des informations fournies. 2) Bien que les composants du Context toolkit soient répartis, la machine qui héberge le Discoverer doit être connue par tous. Le Discoverer, élément central, constitue donc le point faible de cette infrastructure tant du point de vue du passage à l'échelle que de la mobilité.

Au bilan, le context toolkit est une infrastructure simple à comprendre, construite autour d'un modèle centré processus faiblement centralisé. Elle couvre les trois niveaux d'abstraction : Les context-widgets pour le niveau de capture, les interpréteurs pour le niveau interprétation et

les agrégateurs pour le niveau identification. Elle offre en plus un service d'actions basé sur les actuators.

Le service d'historique est disponible au niveau des context-widgets seulement, la découverte se fait par le biais du discoverer, et la sécurité et la protection de la sphère privée ne sont pas traitées. Le modèle n'inclut pas de mesures de la qualité (métadonnées). L'application doit donc faire confiance ou pratiquer ces mesures elle-même.

2.4.3. Projet TEA

Schmidt avait pour objectif dans le projet TEA (*Technology for Enabling Awareness*) de développer une infrastructure matérielle et logicielle permettant à des dispositifs mobiles tels qu'un PDA ou un téléphone mobile d'acquérir des informations contextuelles **[Gellersen 02]**.

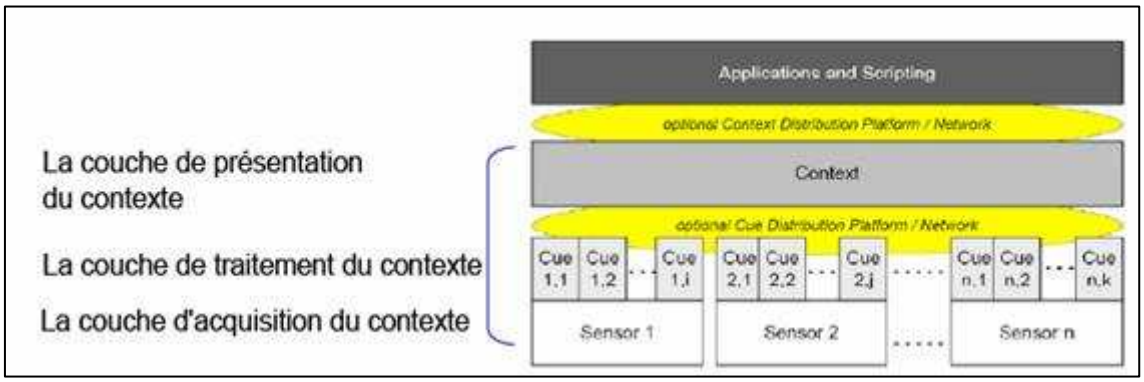

Figure 2.5: Architecture intergicielle proposée par Schmidt [Schmidt 02]

Comme le montre la figure 2.5, l'architecture de *Schmidt* **[Schmidt 02]** s'appuie sur un modèle en couches. La couche du niveau le plus bas est composée des capteurs nécessaires à l'application. Chaque capteur *i* est modélisé par une fonction *Si* ayant pour unique variable le temps *t*. Le résultat est un signal *Xi* appartenant à l'ensemble des signaux pouvant être émis par *i*, soit : $S_i : t \rightarrow X_j$

Le niveau intermédiaire est composé d'entités appelées indices (*cues*). Les indices sont des abstractions des capteurs physiques de la couche inférieure. À un capteur peuvent correspondre plusieurs indices. Chaque indice *j* est modélisé par une fonction *Cj* ayant pour paramètre l'ensemble des signaux *Si* émis entre l'instant *t-n* et l'instant *t* par le capteur *i* associé à *j*. Cette fonction renvoie un résultat *Yij* qui exprime une sémantique donnée. Soit :

$$C_j : S_i(t) \times S_i(t-l) \times ... \times S_i(t-n) \rightarrow Y_{ij} \qquad (2)$$

La strate de niveau supérieur regroupe l'ensemble des indices à partir desquels elle détermine le contexte actuel et en notifie l'application cliente. Cette strate se modélise aussi par une fonction mathématique *T* ayant pour paramètre l'ensemble des indices efficients à l'instant *t*. Le résultat fourni est un vecteur *h(v,p)* composé d'une valeur *v* symbolique d'une situation et d'une valeur *p* indiquant la valeur de confiance associée à *v*. Soit :

$$T : C_0(S_0,t) \times C_1(S_0,t) \times ... \times C_k(S_0,t) ... C_0(S_i,t) \times C_1(C_1,t) \times ... \times C_m(S_i,t) \rightarrow h \qquad (3)$$

Le vecteur *h* définit une surface, fonction du temps, qui est comparée à des surfaces de situations de référence.

Comme pour le context-toolkit, la proposition de *Schmidt* couvre trois niveaux d'abstraction : le niveau capteur correspondant à la couche de capture, le niveau indice pour la couche transformation et le niveau contexte pour la couche d'identification.

Contrairement au context-toolkit, Schmidt s'intéresse à la qualité des données avec des valeurs de confiance et la force du modèle (son fondement mathématique) pourra être perçue comme une faiblesse car éloigné du concepteur logiciel qui s'attend à un modèle plus proche de ses préoccupations de mise en oeuvre. On regrettera également que les informations sur la qualité des données fassent leur apparition au niveau identification seulement. C'est que l'empilement des machines abstraites est étanche : le programmeur n'a pas accès aux couches inférieures. En plus l'identification du contexte nécessite la définition préalable de surfaces de situation de référence. Cela semble assez compliqué pour des cas simples comme par exemple, savoir s'il fait chaud.

Au bilan, le modèle de *Schmidt* présente un fondement mathématique fort, et introduit la notion de qualité de service, et il structure le problème en trois niveaux d'abstraction étanches et fortement couplés, mais il peut sembler complexe et lourd pour le traitement de cas simples.

2.4.4. Gaia

Gaia est une infrastructure intergicielle pour la gestion des espaces intelligents. Où plusieurs actions peuvent être exécutées comme par exemple : contrôler les lumières, ouvrir/fermer des portes d'une salle de classe, exécuter une application pour visualiser des transparents, etc **[Román 02]**. Dans un espace intelligent Gaia, les utilisateurs ont la possibilité d'interagir avec les équipements disponibles mais aussi d'intégrer leurs équipements personnels en tant que ressources additionnelles de cet espace. Les utilisateurs peuvent accéder à des données (applications) qui peuvent être situées dans des espaces distants. Gaia offre aussi aux applications des mécanismes d'adaptation à des changements de l'environnement ubiquitaire.

Gaia s'appuie sur un modèle de contexte basé sur des prédicats du type : Contexte (*TypeContexte*, *Sujet*, *Prédicat*, *Objet*) ou 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.

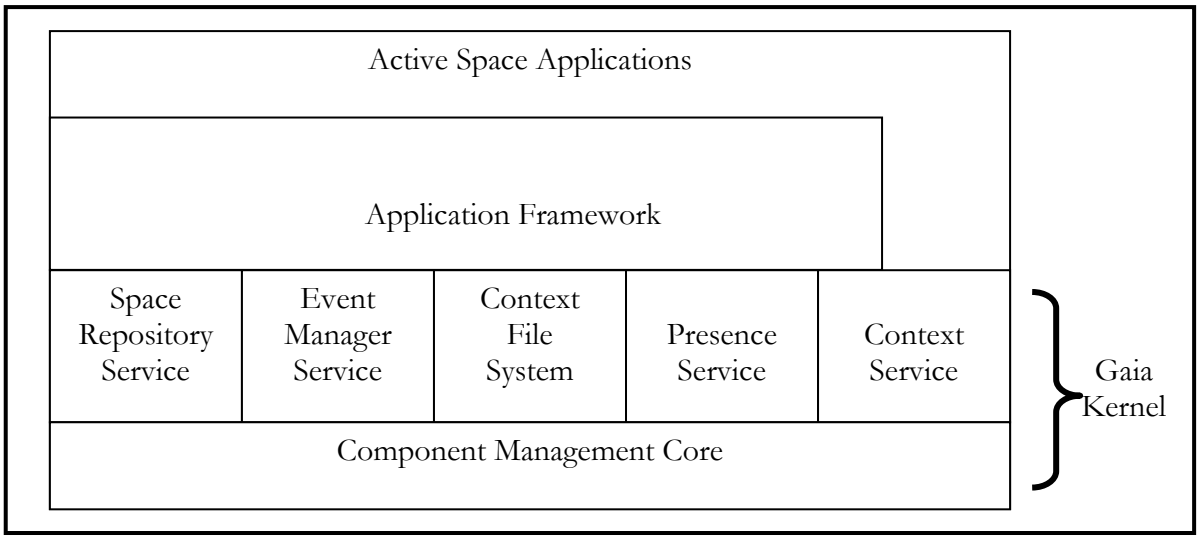

Figure 2.6: Archiecture intergicielle de GAIA [Román 02]

2.4.5. SOCAM

SOCAM (*Service Oriented Context-Aware Middleware*) **[Tao Gu 04, 05]** est un intergiciel pour la sensibilité au contexte qui fournit une infrastructure pour la création de services sensibles au contexte. Il offre aux concepteurs de services le modèle d'ontologie CONON pour la description du contexte.

CONON (*CONtext ONtology*) **[Hang Wang 04]** est une ontologie de contexte extensible écrite en utilisant le langage OWL-DL. Elle permet de décrire le contexte auquel des services peuvent être sensibles dans un environnement ubiquitaire. Cette ontologie est structurée en deux niveaux. Le premier niveau de l'ontologie CONON permet de décrire des concepts généraux communs à toutes les applications sensibles au contexte. Ces concepts sont constitués d'informations de localisation, d'informations sur l'utilisateur, d'informations sur des activités et des entités de calcul. Le second niveau de l'ontologie est extensible et permet d'ajouter à CONON des ontologies pour des domaines d'applications spécifiques comme l'illustre la figure 2.7.

L'ontologie CONON ne prend pas en compte la description des règles d'interprétation ni des politiques d'adaptation. Elle ne modélise pas non plus les capteurs à partir desquels les informations de contexte sont collectées. CONON ne décrit que le contexte et certaines relations existantes entre les informations de contexte en ajoutant la notion de qualité des données.

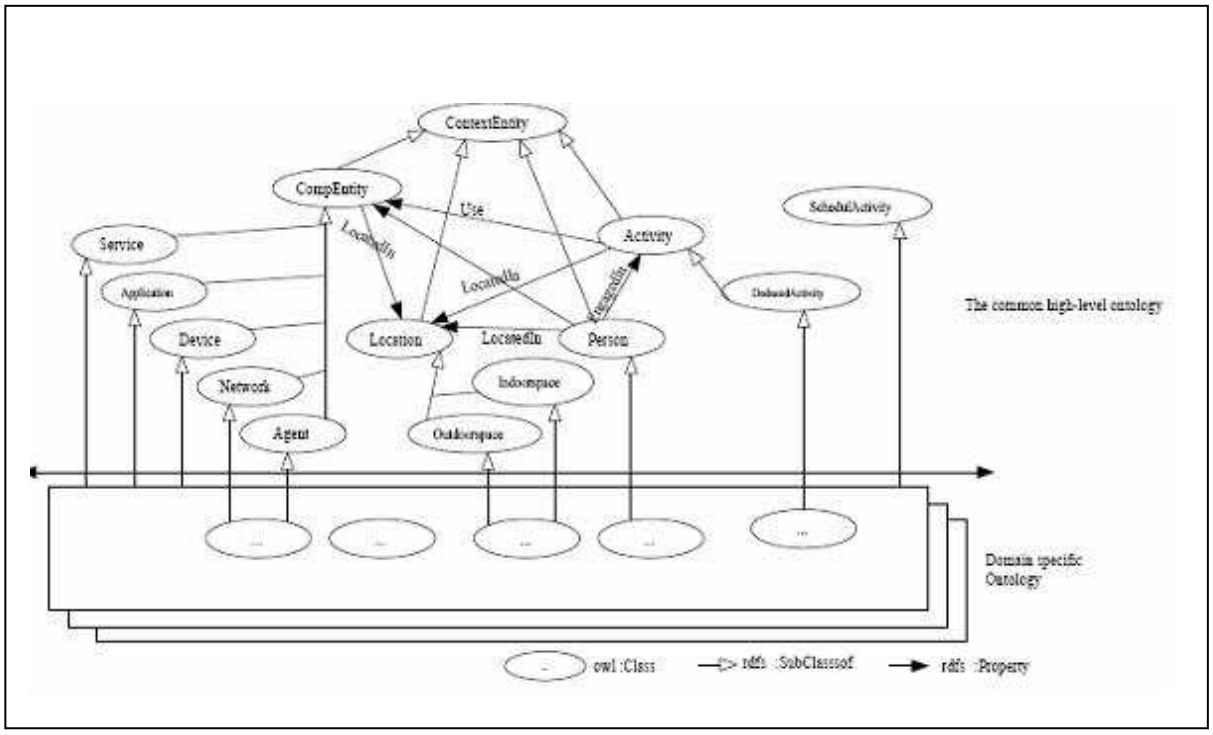

Figure 2.7: Ontologie de contexte CONON [Hang Wang 04]

Comme l'illustre la figure 2.8, l'architecture de l'intergiciel SOCAM est constituée d'un ensemble de composants qui lui permettent de gérer automatiquement la collecte et l'interprétation du contexte. Le *Context provider* se charge de collecter le contexte auquel chaque service est sensible et de le fournir au Context interprété. Il existe deux types de Context provider : les Context providers externes et les Context providers internes. Les Context providers externes se chargent de collecter le contexte à partir de sources de données externes à la machine comme la température de la pièce par exemple. Les Context providers internes se chargent de collecter des données sur l'état interne de la machine comme la valeur de la bande passante par exemple. Le Context interprété est un composant qui fournit un service de raisonnement sur les informations de contexte dans le but de les interpréter. Le composant Service de découverte des services est équivalent aux pages blanches. Il est utilisé par le Context Interpreter et le Context Provider pour s'enregistrer et par les services sensibles au contexte pour trouver des services d'interprétation du contexte et de collecte des données.

Les services sensibles au contexte sont soit des applications soit des agents. Ils utilisent deux catégories de contextes observables : le contexte de bas niveau et/ou le contexte interprété. Ils adaptent leurs comportements selon les valeurs observées du contexte. Pour cela, chaque service s'enregistre auprès du Context provider pour acquérir le contexte auquel il est sensible (contexte pertinent) et fournit au Context interprété des règles d'interprétation qui permettent à l'intergiciel d'interpréter le contexte. À la réception d'une observation du contexte le service se charge d'analyser les données reçues et dans le cas échéant d'appliquer l'adaptation nécessaire.

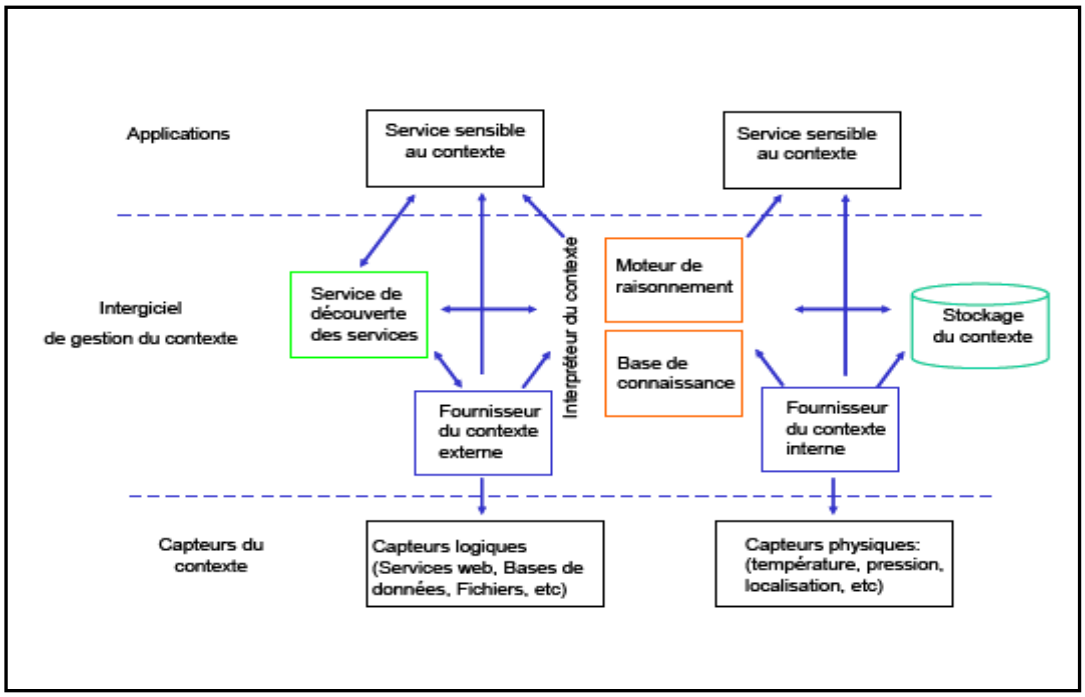

Figure 2.8: Architecture intergicielle de SOCAM [Hang Wang 04]

L'intergiciel SOCAM décharge les développeurs de services sensibles au contexte de la tâche de collecte et d'interprétation des données, il leur fournit un modèle pour décrire le contexte à observer. Mais les développeurs doivent implémenter l'analyse et l'adaptation de chaque service. Cet intergiciel n'offre aucun moyen aux développeurs pour choisir les capteurs que l'intergiciel doit utiliser afin de collecter le contexte.

2.4.6. CARISMA

CARISMA (*Context Aware Reflexive Middleware for Mobile Applications*) **[Capra 00]** est un intergiciel réflexif qui permet aux développeurs de créer des applications sensibles au contexte. Cet intergiciel offre une grammaire basée sur XML pour modéliser les situations pertinentes et les actions d'adaptation associées. Cela permet à l'intergiciel d'adapter les services de l'application lors des variations pertinentes de l'environnement. Des API réflexives permettent de changer le profil de l'application au cours de son exécution.

L'intergiciel est vu par l'application comme un fournisseur de services dynamiquement configurable. Cette configuration est effectuée grâce au profil de l'application décrit dans le modèle **[Capra 02]**. Un profil d'application est constitué de deux parties distinctes **[Capra 03]**, la partie réactive et la partie proactive. La partie réactive permet au développeur de modéliser les situations pertinentes devant être notifiées à l'application pour qu'elle réagisse. La partie proactive permet au développeur de modéliser les adaptations proactives qui représentent l'adaptation des services de l'application en fonction du contexte. Chaque service que l'application veut adapter peut être fourni avec différentes politiques, chaque politique étant associée à une situation pertinente.

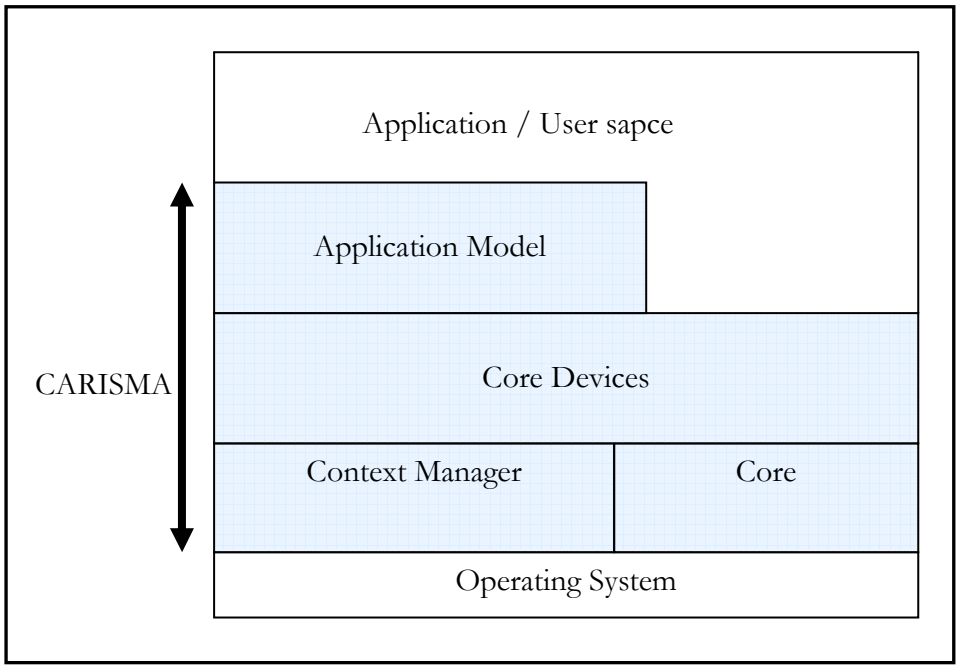

Figure 2.9: Architecture intergicielle de CARISMA [Capra 02]

Comme l'illustre la figure 2.9, l'intergiciel CARISMA est constitué de cinq entités qui se chargent de gérer les applications sensibles au contexte. L'entité *Core* fournit des fonctionnalités de base comme la gestion de la communication dans un environnement distribué et la découverte de service. L'entité *Context Manager* se charge d'interagir avec les capteurs pour collecter les informations de contexte auxquels chaque application est sensible (contexte pertinent), et de surveiller le changement de leurs valeurs. L'entité *Core Services* se charge de gérer les informations de contexte en activant l'entité Context Manager. L'entité *Context Manager* se charge de collecter le contexte, d'intercepter les invocations de l'application, et de les adapter. L'entité *Application Model* offre un support pour créer et exécuter des applications sensibles au contexte.

Lorsque plusieurs applications s'exécutent dans une instance de l'intergiciel, les profils de ces applications peuvent entrer en conflit. La détection et la résolution de ces conflits ne peuvent pas se faire statiquement, puisque les profils de l'application peuvent être changés par l'utilisateur en cours d'exécution de l'application. CARISMA propose une approche dynamique de résolution de conflit à chaque fois qu'ils se produisent. Cette approche est calquée sur un modèle économique de vente aux enchères. Les différentes applications sont considérées comme des consommateurs qui sont en concurrence pour obtenir les services fournis par le producteur (l'intergiciel).

L'intergiciel CARISMA utilise la réflexivité comme technique d'adaptation pour faciliter le développement des applications sensibles au contexte. Cet intergiciel prend en compte la collecte du contexte et l'analyse des données collectées. Il se charge de notifier les applications lors de la détection des situations pertinentes associées aux adaptations réactives, et d'adapter les appels de services lors de la détection des situations pertinentes associées aux adaptations proactives.

CARISMA utilise le profil de l'application pour prendre les décisions de notification et d'adaptation appropriées. Cet intergiciel offre la possibilité aux utilisateurs de l'application de changer leurs profils en cours d'exécution grâce à des APIs réflexives. Cependant, il n'offre aucun moyen aux développeurs de décrire ni de choisir les capteurs à utiliser pour collecter le contexte. L'intergiciel CARISMA interagit avec des capteurs qui ne peuvent collecter que des informations

locales à la machine où est installée l'application, ce qui limite considérablement le type d'applications à développer à l'aide de cet intergiciel. CARISMA utilise des informations de contexte de bas niveau. Aucun mécanisme d'interprétation n'est intégré à l'intergiciel.

2.4.7. RCSM

RCSM (*Reconfigurable Context Sensitive Middleware*) **[SFW 02]** est un intergiciel orienté objet qui permet la création et la gestion d'applications sensibles au contexte. Pour faciliter le développement de ce type d'applications, RCSM demande aux développeurs d'écrire une interface sensible au contexte qui décrit les situations pertinentes auxquelles l'application est sensible et les règles d'adaptation de cette application. Cette interface est décrite à l'aide du langage CA-IDL (*Context Aware-Interface Description Language*). RCSM utilise cette interface pour générer le code d'adaptation de l'application.

Le langage CA-IDL fige les types de contextes observables que l'intergiciel peut surveiller. Ces types sont répartis dans trois catégories de contexte : la catégorie *DeviceSpecificContext* pour les contextes spécifiques à la machine sur laquelle s'exécute l'application, la catégorie *EnvironmentSpecificContext* pour les contextes associés à l'environnement, et la catégorie *UserSpecificContext* pour le profil de l'utilisateur. RCSM considère une application comme un ensemble d'objets sensibles au contexte. Chaque objet est structuré en deux parties : une interface sensible au contexte et une implémentation indépendante du contexte. L'interface encapsule la description de la sensibilité au contexte, alors que l'implémentation reste indépendante du contexte.

La compilation d'une interface sensible au contexte permet à l'intergiciel de générer le code d'adaptation de l'objet auquel l'interface est associée sous la forme de conteneur d'objet. Le conteneur d'objet se charge d'enregistrer l'objet auprès du R-CAP (*Context Aquisition and Processing framework*) pour le contexte auquel il est sensible. Le R-CAP est une entité appartenant à l'intergiciel RCSM qui se charge de collecter le contexte, de l'analyser et de lancer les actions d'adaptation nécessaires quand une situation pertinente est détectée. À chaque catégorie de contextes est associé un ensemble de capteurs qui se chargent de collecter le contexte auquel l'application est sensible.

L'intergiciel RCSM facilite le développement des applications orientées objets sensibles au contexte en offrant un langage de description d'interfaces sensibles au contexte et en déchargeant le développeur d'application de la tâche de collecte, d'analyse du contexte et d'adaptation de l'application. Cet objectif est atteint grâce à la génération de code d'adaptation à partir des interfaces sensibles au contexte décrites par le développeur d'application. Ces adaptations sont des adaptations comportementales de nature réactives. Cependant, RCSM ne prend pas en compte l'interprétation du contexte et n'offre aucun moyen de choisir, pour chaque contexte, le capteur à partir duquel les données seront collectées. L'ajout ou la suppression d'un capteur nécessite la modification de la grammaire du CA-IDL et de la plateforme RCSM.

2.4.8. CASS

CASS (*Context-Awareness Sub-Structures*) **[Fahy 04]** est un intergiciel pour la sensibilité au contexte qui permet aux développeurs de gérer leurs applications. Cette gestion concerne l'interaction avec les capteurs pour collecter les informations de contexte, la sauvegarde des données collectées dans une base de données et leur interprétation.

Comme l'illustre la figure 2.10, l'intergiciel CASS est structuré en un ensemble de composants pour gérer le contexte. Le *SensorListner* se charge d'interagir avec des noeuds de capteurs pour collecter le contexte et de le sauvegarder dans la base de données. Le composant *Interpreter* se charge de calculer des informations de contexte de haut niveau en utilisant le *RuleEngine* et le *ContextRetriever* se charge de retrouver les observations du contexte dans la base de données. L'intergiciel se charge de détecter le changemement du contexte observé et de notifier les applications sensibles à ces contextes. En effet, chaque application se charge d'analyser ce nouveau contexte et d'appliquer l'adaptation nécessaire.

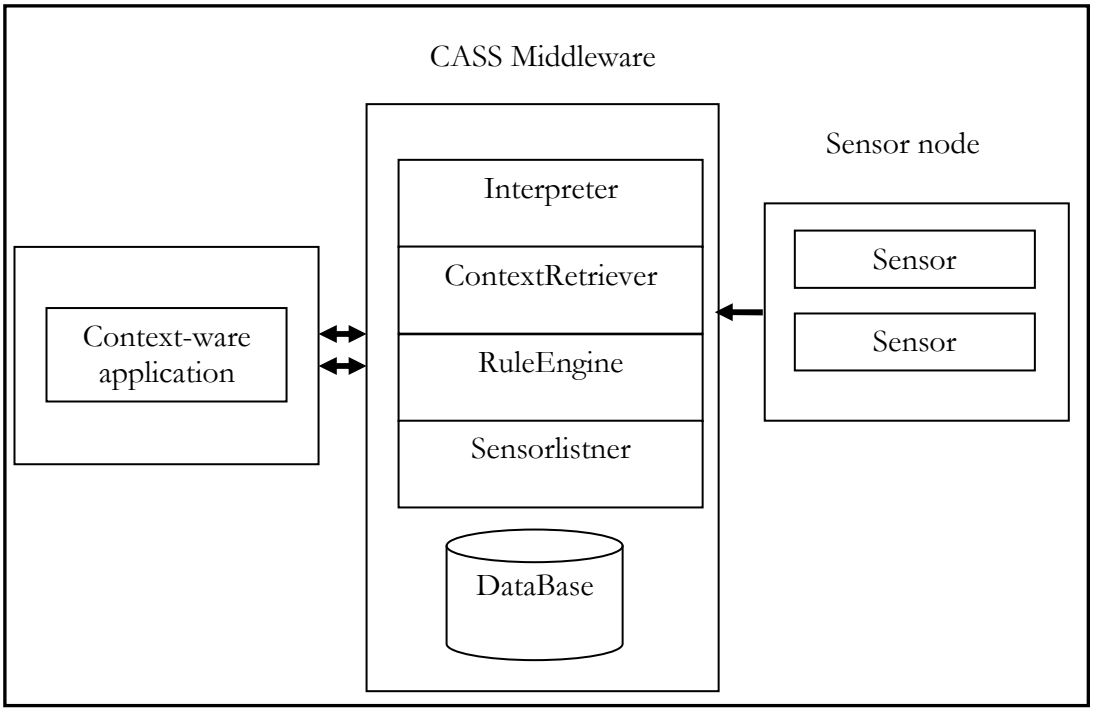

Figure 2.10: Architecture intergicielle de CASS

L'intergiciel CASS facilite l'interaction avec les capteurs et l'interprétation des données en intégrant dans son architecture des composants qui se chargent d'effectuer ces tâches. Les applications sensibles au contexte doivent s'enregistrer auprès de cet intergiciel pour être notifié à chaque fois qu'une variation de contexte a été détectée. Le développeur de l'application se charge de programmer l'analyse de ces données ainsi que l'adaptation de l'application. Ce dernier ne peut pas choisir les capteurs que l'intergiciel doit utiliser pour collecter les informations de contexte.

2.4.9. CoBrA

L'architecture de CoBrA *(Context Broker Architecture)* **[Chen 04a]** s'appuie sur système multi agents centré autour d'un agent de courtage *(Context Broker Agent)*. L'agent de courtage est un serveur central de contexte qui maintient une base de connaissances commune à tous les agents **[Chen 04b]**.

Les agents peuvent être des applications qui s'exécutent localement, des services offerts par des équipements de l'environnement mobile ou des services web de présence des personnes, etc. Le rôle de l'agent COBRA se résume aux tâches suivantes (Figure 2.11 et 2.12) :

(i) Acquisition du contexte depuis les différentes sources de l'environnement à travers différents types de capteurs, inférence de nouvelles connaissances contextuelles,

(ii) Vérification de la cohérence de la base de connaissances du contexte et protéger la vie privée des utilisateurs, par la mise en place des mécanismes d'accès et de sécurité.

Les équipements mobils comme les téléphones portables et les PDAs n'ont pas assez de ressources pour faire des calculs et des traitements complexes sur le contexte capturé, d'où la nécessité d'utilisation d'une architecture centralisée basée sur des agents courtiers qui a pour objectif d'introduire un serveur doté de ressources de calcul suffisantes pour exécuter des fonctions complexes de traitement du contexte.

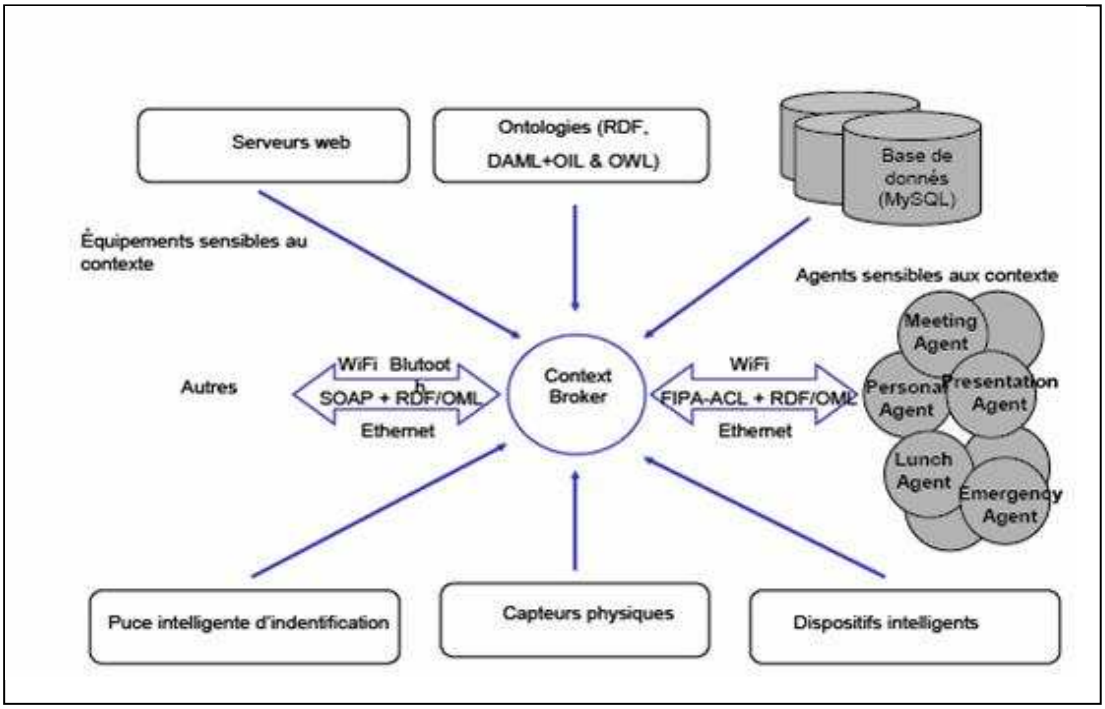

Figure 2.11: Architecture d'un espace actif basé sur l'agent CoBra (a) [Chen 04b]

L'architecture de l'agent Broker est composée de trois modules **[Chen 03a]** :

1- Une base de connaissances qui contient les différentes connaissances contextuelles acquises.

2- Un moteur d'inférence qui 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. Les règles d'inférence sont basées sur le langage FLORA.

3- Un gestionnaire de politiques associées au contexte : Ce module, permet à l'agent CoBrA d'analyser les politiques de gestion du contexte définies par les utilisateurs et de décider des actions correspondantes.

L'agent CoBrA repose sur un modèle du contexte appelé Soupa (*Standard Ontology for Ubiquitous and Pervasive Applications*) **[Chen 04c]**. Ce dernier est composé d'un ensemble d'ontologies écrites en langage OWL, qui permettent à des applications hétérogènes de partager la même interprétation du contexte à travers un vocabulaire commun.
SOUPA est composée de deux ensembles d'ontologies : le noyau (*core*) et les extensions. Le noyau comporte des ontologies de base, indépendantes des applications, qui sont écrites par extension de plusieurs ontologies existantes (*FOAF, DAML-TIME, RCC*, etc.). Les ontologies d'extensions (extensions *ontologies*) comprennent des classes spécifiques au domaine d'application.

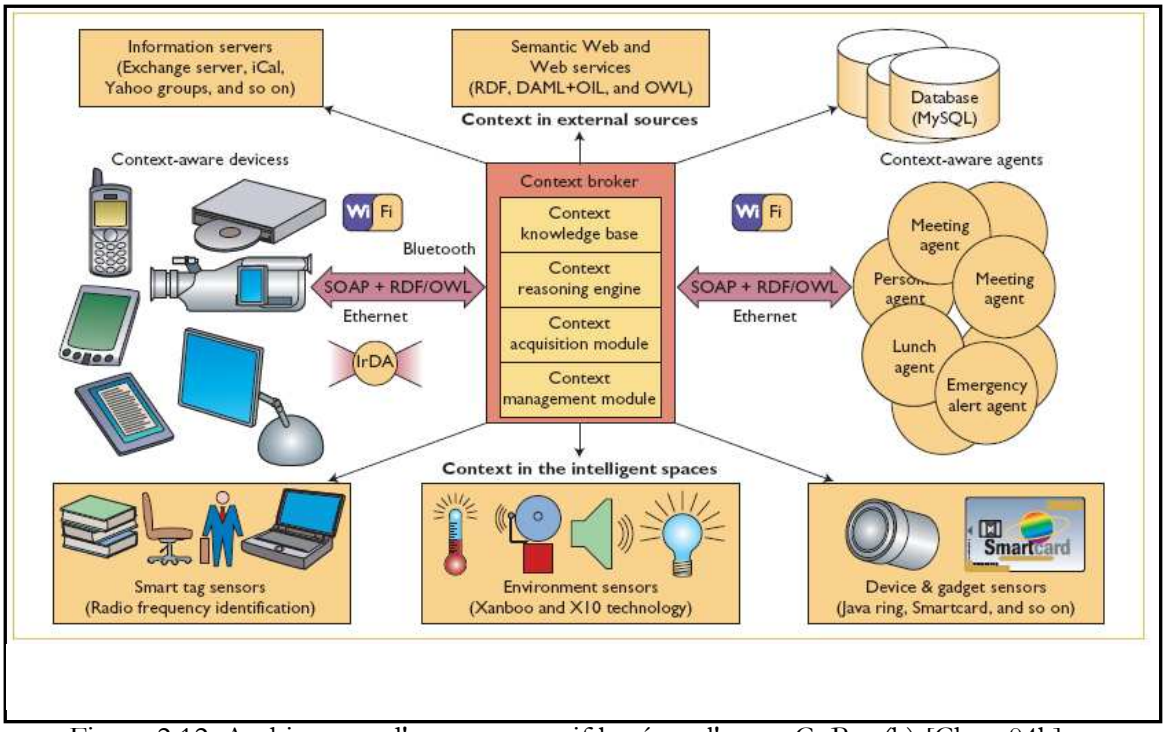

Figure 2.12: Architecture d'un espace actif basé sur l'agent CoBra (b) [Chen 04b]

CoBrA présente deux inconvénients majeurs d'un point de vue développement. Les agents qui interrogent le serveur de contexte doivent disposer d'un modèle global (ensemble des ontologies) du contexte, nécessitant des requêtes complexes et par conséquent des temps de traitement relativement importants. Cependant, d'un point de vue généricité, ces requêtes sont difficilement réutilisables pour d'autres agents.

2.4.10. MyCampus

Le projet MyCampus 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 CMU **[Gandon 03]**. Comme exemples de services, nous pouvons citer le service de recommandation de restaurants présents sur le campus, le service d'organisation de réunions entre utilisateurs, etc. MyCampus repose sur une architecture multi agents qui s'articule autour d'un agent central appelé *e-Wallet* **[Gandon 03]**. Ce dernier a pour rôles

- L'acquisition et le traitement des connaissances contextuelles. le respect des règles de la vie privée de l'utilisateur (Figure 2.13).

Dans l'environnement MyCampus, un agent e-Wallet permet de gérer le contexte de plusieurs utilisateurs.

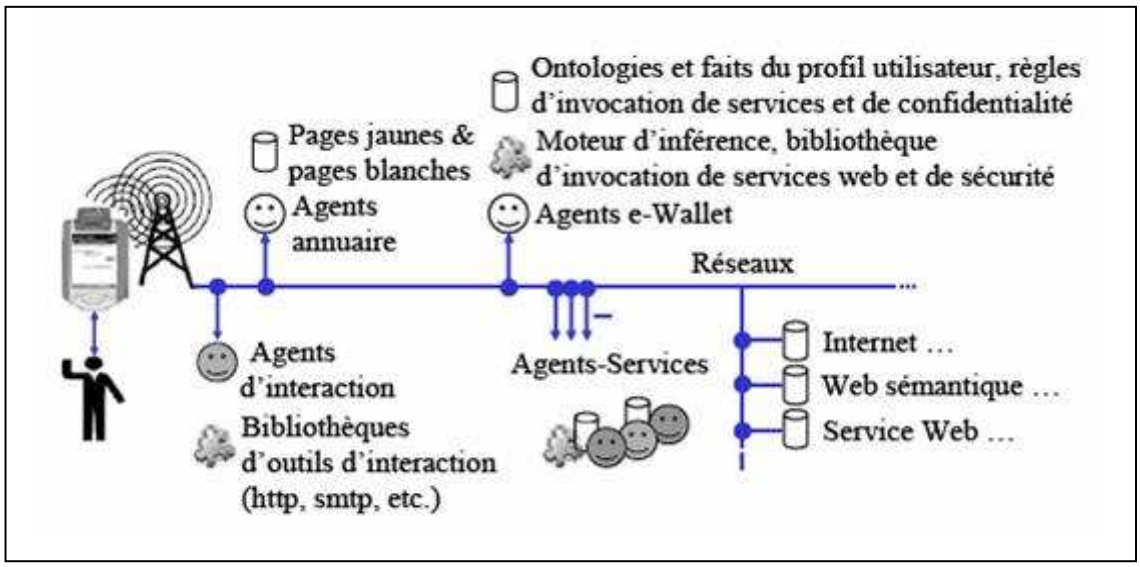

Figure 2.13: Architecture itergicielle deMyCampus [Gandon 03]

L'agent e-Wallet est structuré en trois couches (noyau, services, confidentialité).

- 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.

L'architecture du système MyCampus comporte trois autres types d'agents : les agents d'interaction, les agents annuaires et les agents de services. Les agents d'interaction gèrent les échanges avec l'utilisateur et les interactions avec les autres agents. Les agents annuaires permettent à des agents/utilisateurs à la recherche d'un service d'obtenir la liste des agents fournissant ce service. Les agents de services sont gérés par l'intermédiaire de l'agent e-Wallet.

L'agent e-Wallet repose sur un modèle du contexte à base d'ontologies web sémantique exprimées en langage OWL **[Gandon 04]**. Dans ce modèle, on distingue d'une part, une ontologie OWL permettant de décrire les données personnelles de l'utilisateur et son contexte courant, et d'autre part, des ontologies spécifiques qui sont une extension du langage OWL pour décrire respectivement, des règles de déduction, d'appel de services, de confidentialité et d'accès au contexte.

2.5. Synthèse

Dans les différents projets qui ont été présentés ci-dessus, les architectures intergicielles de gestion du contexte sont généralement composées de plusieurs couches hiérarchiques d'abstraction du contexte, permettant de séparer les fonctions de perception, de décision et d'action contextuelles des aspects fonctionnels des applications. En global les architectures intergicielles de gestion du contexte peuvent être classées en deux catégories :

- Les intergiciels à base de capteurs de contexte tels que Context Toolkit **[Dey 01]** et TEA **[Schmidt 02]**. Ce type d'infrastructure répond à la problématique de l'acquisition et de traitement du contexte à partir de capteurs hétérogènes disséminés dans l'environnement.

- Les intergiciels à base de serveur de gestion centralisée du contexte. Un serveur de gestion du contexte est une entité logicielle à l'image de l'agent CoBrA ou de l'agent E-Wallet, qui fournit des connaissances contextuelles de haut niveau et des services intelligents aux utilisateurs **[Chen 04a]**.

Bien que le problème d'hétérogénéité des sources du contexte a été bien traité, celle de la gestion des sources contextuelles dans un environnement ubiquitaire n'a pas été suffisamment traitée dans les projets présentés dans ce chapitre. La gestion des sources de contexte nécessite de disposer d'un mécanisme de découverte dynamique de ces sources, adapté à la nature dynamique et non déterministe de l'environnement ubiquitaire (mobilité, fluctuation des connexions réseaux, disponibilité des équipements). Parmi les travaux présentés, seuls les projets Socam et MyCampus proposent un service de découverte des sources contextuelles. Dans Socam, le service de découverte des composants fournisseurs crée dynamiquement des liens entre les applications et les composants fournisseurs disponibles, qui peuvent changer pendant d'exécution des applications. L'absence d'un mécanisme flexible de découverte est un inconvénient majeur des architectures présentées car les sources contextuelles sont supposées disponibles pendant le temps d'exécution de l'application, ce qui est loin d'être le cas dans un environnement réel. Ainsi, le fait qu'un ou plusieurs capteurs de contexte puissent devenir indisponibles peut altérer le fonctionnement d'une application sensible au contexte.
Le tableau suivant résume les principales caractéristiques des précédents integiciels.

Intergiciel	Architecture	Modélisation du contexte	Entités d'acquisition du contexte	Entités de traitement du contexte	Vérification du contexte	Découverte de services	Historique du contexte	Sécurité / confidentialité
Context Toolkit	Intergiciel à objets capteurs répartis	XML	Objets Widget	Objets interpreter	Non	Non	Non	Non
Projet TEA	Intergiciel à objets capteurs embarqués	Fonctions Mathématique	Objets Cues	Tuple space	Non	Non	Non	Non
Gaia	Serveur centralisé (Agents)	DAML	Agent provider	Non	Non	Non	Oui	Non
SOCAM	Serveur centralisé (composants)	OWL	Composants, Capteurs	Composant interpréteur	Oui	Oui	Non	Non
CARISMA	Intergiciel réflexif	XML	Context Manager	Non	Non	Oui	Non	Non
RCSM	Intergiciel à objets capteurs répartis	CA-IDL	R-CAP	Non	Non	Non	Non	Non
CoBrA	Serveur centralisé (Agents)	OWL	Agent CoBra	Agent CoBrA	Oui	Non	Non	Oui
MyCampus	Serveur centralisé (Agents)	OWL	Agent eWallet	Agent eWallet	Oui	Oui	Non	Oui

Chapitre 3 :

La Représentation des Connaissances : état de l'art des travaux

3.1. Introduction

À sa création par Tim Berners Lee, au début des années 1990, le web était exclusivement destiné à partager des informations sous forme de pages html, affichables par un logiciel «navigateur web », et généralement destinées à être lues par un utilisateur humain.

Très rapidement, on s'est rendu compte que cette conception du web était bien trop limitée, et ne permettait pas un réel partage du savoir.

L'arrivée de *XML*, en 1998, a donné un cadre à la structuration des connaissances, rendant ainsi possible la création de nouveaux langages web destinés non plus à un rendu graphique à l'écran pour un utilisateur humain, mais à un réel partage et à une manipulation des savoirs, ce qui donne naissance au web sémantique.

Le Web sémantique n'est pas un Web séparé, mais une extension du Web actuel dans lequel l'information est munie d'une signification bien définie permettant aux ordinateurs et aux personnes de mieux travailler en coopération **[Berners-Lee 01]**. Cette notion du Web sémantique, due à Tim Berners-Lee au sein du W3C, fait d'abord référence à la vision du Web de demain comme un vaste espace d'échange de ressources entre êtres humains et machines permettant une exploitation, qualitativement supérieure, de grands volumes d'informations et de services variés. Les utilisateurs déchargés d'une bonne partie de leurs tâches de recherche, de construction et de combinaison des résultats, grâce aux capacités accrues des machines à accéder aux contenus des ressources et à effectuer des raisonnements sur ceux-ci. Le Web sémantique est d'abord une infrastructure pour permettre l'utilisation de connaissances formalisées en plus du contenu informel actuel du Web. Cette infrastructure doit permettre d'abord de localiser, d'identifier et de transformer des ressources de manière robuste et saine tout en renforçant l'esprit d'ouverture du Web avec sa diversité d'utilisateurs. Elle doit s'appuyer sur un certain niveau de consensus portant, par exemple, sur les langages de représentation ou sur les ontologies utilisés. Elle doit contribuer à assurer, le plus automatiquement possible, l'interopérabilité et les transformations entre les différents formalismes et les différentes ontologies. Elle doit offrir des mécanismes de protection (droits d'accès, d'utilisation et de reproduction), ainsi que des mécanismes permettant de qualifier les connaissances afin d'augmenter le niveau de confiance des utilisateurs.

La convergence actuelle d'Internet vers une vision sémantique et l'apparition des langages de contenus pour le web font une réponse prometteuse aux besoins de l'informatique ubiquitaire, tout en garantissant une certaine interopérabilité sémantique des connaissances et le partage d'une représentation standard et intelligible de la sémantique des connaissances contextuelles tout en renforçant l'esprit d'ouverture et de sécurité de ces derniers.

3.2. Ontologie : Définitions

Le concept d'ontologie est utilisé depuis très longtemps, notamment en linguistique et en traitement des langages naturels, une ontologie définit les termes utilisés pour décrire et représenter un champ d'expertise. Les ontologies sont utilisées par les personnes, les bases de données et les applications, qui ont besoin de partager des informations relatives à un domaine bien spécifique comme, la médecine, la fabrication d'outils, la gestion de finances, etc. Les ontologies associent les concepts de base d'un domaine précis et les relations entre ces concepts, tout cela d'une manière compréhensible par les machines. Elles encodent la connaissance d'un domaine particulier ainsi que les connaissances qui recouvrent d'autres domaines, ce qui permet de rendre les connaissances réutilisables.

Autrement dit, une ontologie est un modèle d'organisation des connaissances dans un domaine donné. On trouvera dans l'ontologie les classes d'objets à organiser (personnes, étudiant, professeur, thèse...), les types d'attributs pouvant être attachés aux objets (référence, description, adresse, nom...) et les types de relations entre les objets (un objet "étudiant " peut être relié par une relation "supervisé par" à un objet de type "professeur"), etc...

Les ontologies informatiques sont des outils qui permettent de représenter précisément un corpus de connaissances sous une forme utilisable par une machine. Elles représentent un ensemble structuré de concepts. Les concepts sont organisés dans un graphe dont les relations peuvent être des relations sémantiques et/ou des relations de composition et d'héritage (au sens objet). Ce terme est utilisé aussi dans le domaine de la gestion des connaissances. Dans ce dernier, une ontologie permet de fournir le « sens » des symboles utilisés pour construire un modèle du monde (On parle parfois de méta modèle). On peut citer l'exemple d'une carte géographique qui représente un modèle (une abstraction) du monde réel.

On distingue généralement deux entités globales au sein d'une ontologie. La première, à objectif terminologique, définit la nature des éléments qui composent le domaine de l'ontologie en question, un peu comme la définition d'une classe en programmation orientée objet définit la nature des objets que l'on va manipuler par la suite. La seconde partie d'une ontologie explicite les relations entre plusieurs instances de ces classes définies dans la partie terminologique. Ainsi, au sein d'une ontologie, les concepts sont définis les uns par rapport aux autres (modèle en graphe de l'organisation des connaissances), ce qui autorise un raisonnement et une manipulation de ces connaissances.

3.3. Architecture du web sémantique

La proposition du W3C s'appuie au départ sur une pyramide de langages dont seulement les couches basses sont aujourd'hui relativement stabilisées. La figure 3.1 montre une des versions [**Hyvönen Eero 02]** de l'organisation en couches proposée par le W3C. Deux types de bénéfices peuvent être attendus de cette organisation. (1) Elle permet une approche graduelle dans les processus de standardisation et d'acceptation par les utilisateurs. (2) Par ailleurs, si elle est bien conçue, elle doit permettre de disposer du langage au bon niveau de complexité, celle-ci étant fonction de l'application à réaliser.

Un aspect central de l'infrastructure est sa capacité d'identification et de localisation des diverses ressources. Elle repose sur la notion d'URI (*Uniform Resource Identifier*) qui permet d'attribuer un identifiant unique à un ensemble de ressources, sur le Web bien sûr mais aussi dans d'autres domaines (documents, téléphones portables, personnes, etc.). Cette notion connaît

aujourd'hui de nombreuses extensions, en cours de standardisation, à d'autres entités que les URLs. Elle est à la base même des langages du W3C.

Une autre caractéristique de tous ces langages est d'être systématiquement exprimables et échangeables dans une syntaxe XML. Ceci permet de bénéficier de l'ensemble des technologies développées autour d'XML : XML Schemas, outils d'exploitation des ressources XML (bibliothèques JAVA, etc.), bases de données gérant des fichiers XML.

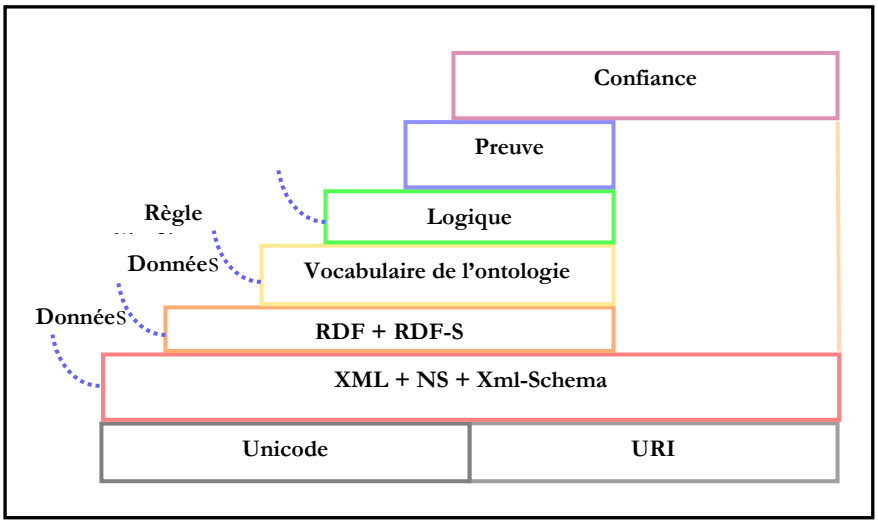

Figure 3.1: Architecture en couches du web sémantique [Hyvönen Eero 02]

3.4. Les langages du W3C

Les travaux visant la réalisation du Web sémantique se situent à des niveaux de complexité très différents. Les plus simples utilisent des jeux plus ou moins réduits de méta-données dans un contexte de recherche d'information ou pour adapter la présentation des informations aux utilisateurs. Dans ce cas, des langages de représentation simples sont suffisants. Dans les travaux plus complexes mettant en oeuvre des architectures sophistiquées, pour permettre par exemple l'exploitation de ressources hétérogènes, des langages plus expressifs et plus formels issus des travaux en représentation et en ingénierie des connaissances, sont nécessaires.

3.4.1. RDF

Le premier des langages du web sémantique est RDF (*Resource Description Framework*) **[RDF]** auquel s'est ajouté rapidement RDF Schema (RDFS). Les objectifs initiaux de RDF étaient la représentation et une meilleure exploitation des méta-données. Mais, de manière plus générale, RDF permet de voir le Web comme un ensemble de ressources reliées par les liens étiquetés «*sémantiquement*». RDF a permis aussi d'exprimer de larges vocabulaires, surtout quand il est complété avec RDFS qui permet d'offrir un niveau supérieur de structuration. Les énoncés RDF sont des triplets (*ressource - attribut - valeur*), la valeur est une ressource ou chaîne de caractères. Une ressource doit disposer d'une URI. Les triplets sont interprétables comme (sujet – prédicat – objet) **[RDF 04a]** (Figure 3.2), le sujet représente la ressource à décrire, le prédicat représente un type de propriété applicable à cette ressource et l'objet représente une donnée ou une autre ressource.

La simplicité du modèle, critiquable pour certains, peut-être une des clés de son acceptation et de la relative simplicité de la réalisation d'outils. Certains ajouts comme la possibilité de considérer un énoncé (*triplet*) RDF comme un noeud du graphe lui-même, peuvent augmenter l'expressivité du langage, particulièrement dans un contexte discursif ou de méta-données.

Un document RDF ainsi formé correspond à un multi-graphe orienté étiqueté. Chaque triplet correspond alors à un arc orienté dont le label est le prédicat, le nœud source est le sujet et le nœud cible est l'objet.

La sémantique d'un document RDF peut être exprimée en théorie des ensembles et en théorie des modèles en se donnant des contraintes sur le monde qui peuvent être décrites en RDF. RDF hérite alors de la généricité et de l'universalité de la notion d'ensemble. Cette sémantique peut être aussi traduite en formule de logique du premier ordre, positive, conjonctive et existentielle:

{Sujet, objet, prédicat} ⇔ prédicat(objet, sujet)

Ce qui est équivalent à : ∀ objet, ∃ sujet tq predicat(objet, sujet).

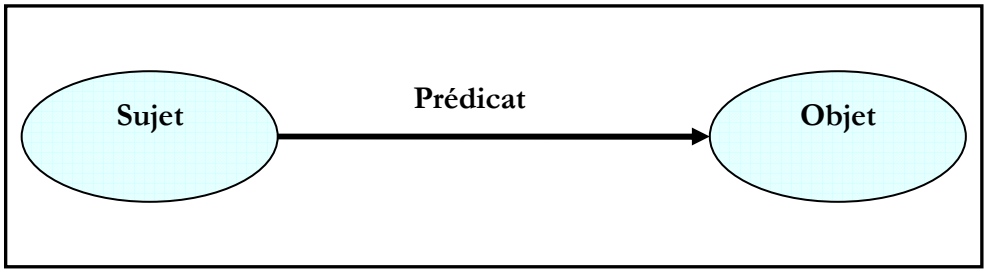

Figure 3.2: Un triplet RDF (ressource , attribut , valeur)

Le W3C a prévu un mécanisme d'inférence pour la sémantique de RDF déduisant exclusivement et intégralement les conséquences des prédicats, sans que ce mécanisme ne fasse l'objet d'une recommandation.

3.4.2. RDF-S

RDFS (*Resourse Description Framework schema*) est une extension sémantique du RDF avec la possibilité de définir des hiérarchies de classes et de propriétés dont l'applicabilité et le domaine de valeurs peuvent être contraintes à l'aide des attributs *rdfs : domaine* et *rdfs : range* **[RDFS 04b]**. Par exemple, si une propriété représente le nom d'une personne, on peut exiger que les valeurs de cette propriété soient une référence à une personne et non pas une à voiture ou à une maison. On peut aussi restreindre les propriétés s'appliquant à une ressource. Par exemple, cela n'a probablement aucun sens d'autoriser une propriété «date de naissancee» à être appliquée à un morceau de musique. La figure 3.3 illustre l'exemple de la propriété fullName qui a comme valeur une référence à une personne.

```
<rdf:property rdf:about="http://localhost/profile#fullName">
<rdfs:range rdf:resource="http://www.schema.org/TR/rdf-schema#Person>
</rdf:property>
```

Figure 3.3: Représentation du nom d'un personne en RDF

À chaque domaine applicatif peut être ainsi associé un schéma identifié par un préfixe particulier et correspondant à une URI. Les ressources instances sont ensuite décrites en utilisant le vocabulaire donné par les classes définies dans ce schéma. Les applications peuvent alors leur donner une interprétation opérationnelle. On peut noter que RDFS n'intègre pas en tant que tel de capacités de raisonnement.

Pour résumer, XML peut être vu comme la couche de transport syntaxique, RDF comme un langage relationnel de base. RDFS offre des primitives de représentation de structures ou primitives ontologiques.

3.4.3. DAML-OIL

DAML+OIL (*Drapa gent Markup Language / Ontology Inference Layer*) est un langage d'ontologies qui a été proposé pour répondre aux utilisateurs désirant plus de facilités d'expressivité et de raisonnement que celles qui sont fournies par RDF et RDF-S sur les classes, les instances, les relations entre classes, les types, les énumérations, etc. La dernière version de DAML+OIL, intègre le typage de données basé sur le langage W3C XML Schema Definition Language (XSDL) **[Horrocks 02]**.

3.4.4. OWL

OWL (*Ontology Web Language*) s'appuie sur le langage DAML+OIL, produit de la combinaison de l'américain DAML et OIL provenant de projets européens. Le langage OWL est actuellement construit sur RDFS, et apporte ainsi aux langages du Web sémantique, l'équivalent d'une logique de description tout en disposant aussi d'une syntaxe XML. Sans être exhaustif, il ajoute à RDF la possibilité de définir des classes de manière plus complexe correspondant aux connecteurs de la logique de description équivalente (intersection, union, restrictions diverses, etc.), les classes disjointes, les propriétés inverses ou transitives ou bien encore les restrictions de cardinalité sur les propriétés. Ces descriptions peuvent être utilisées ensuite par un raisonneur comme FaCT (*Fast Classification of Terminology*), pour inférer par exemple la subsumption de concepts. En se fondant sur une logique de description, un tel langage a une sémantique formelle claire, ce qui permet de le doter de services inférentiels.

Le langage OWL offre trois sous-langages d'expression croissante conçus pour des communautés de développeurs et d'utilisateurs spécifiques **[OWL 04]**.

Le langage *OWL Lite* concerne les utilisateurs ayant principalement besoin d'une hiérarchie de classifications et de mécanismes de contraintes simples. Par exemple, quoique OWL Lite gère des contraintes de cardinalité, il ne permet que des valeurs de cardinalité de 0

ou 1. Mettre en œuvre des outils pour OWL Lite devrait être plus simple que pour ses parents d'expression plus grande.

Le langage *OWL DL* concerne les utilisateurs souhaitant une expressivité maximale sans sacrifier la complétude de calcul (toutes les inférences sont sûres d'être prises en compte) et la décidabilité (tous les calculs seront terminés dans un intervalle de temps fini) des systèmes de raisonnement. Le langage OWL DL comprend toutes les structures de langage de OWL avec des restrictions comme la séparation des types (une classe ne peut pas être en même temps un individu ou une propriété, une propriété doit être un individu ou une classe). OWL DL se nomme ainsi pour sa correspondance avec la *logique de description*, un champ de la recherche portant sur un fragment décidable particulier de la logique de premier ordre. Le langage OWL DL, conçu pour gérer le secteur existant de la logique de description, offre les propriétés de calcul souhaitées pour les systèmes de raisonnement.

Le langage *OWL Full* est destiné aux utilisateurs souhaitant une expressivité maximale et la liberté syntaxique de RDF sans garantie de calcul. Par exemple, dans OWL Full, on peut simultanément traiter une classe comme une collection d'individus et comme un individu à part entière. Le langage OWL Full permet à une ontologie d'augmenter la signification du vocabulaire prédéfini (RDF ou OWL). Un système de raisonnement ne pourra probablement pas mettre en œuvre toutes les caractéristiques de OWL Full.

Chaque sous-langage langage représente une extension par rapport à son prédécesseur plus simple, à la fois en ce qu'on peut légalement exprimer et en ce qu'on peut conclure de façon valide.

Les développeurs d'ontologies adoptant OWL devraient évaluer quel sous-langage convient le mieux à leurs besoins. Le choix entre OWL Lite et OWL DL dépendra de la mesure où les utilisateurs auront besoin des structures de restriction plus expressives de OWL DL. Les moteurs de raisonnement OWL Lite auront les propriétés de calcul souhaitables. Les moteurs de raisonnement OWL DL, tout en ayant un sous-langage décidable, seront soumis à une complexité dans l'éventualité du pire plus élevée. Le choix entre OWL DL et OWL Full dépendra principalement des besoins des utilisateurs en ce qui concerne les facilités de Méta modélisation offertes par le schéma RDF (par exemple, pour définir des classes de classes). Entre une utilisation de OWL Full comparée à OWL DL, la prise en charge du raisonnement est moins prévisible.

3.5. Outils disponibles pour le web sémantique

3.5.1. Editeur d'ontologies "Protégé"

Protégé **[STA 2005]** est un éditeur d'ontologies distribué en open source par l'université d'informatique médicale de Stanford. Protégé n'est pas un outil spécialement dédié à OWL, mais un éditeur hautement extensible, capable de manipuler des formats très divers. Le support d'OWL, comme de nombreux autres formats, est possible dans Protégé grâce à un plugin dédié.

3.5.2. Framework Jena

Jena **[JENA 05]** est un framework écrit en Java, dont l'objectif est de fournir un environnement facilitant le développement d'applications dédiées au web sémantique. Jena

permet de manipuler des documents RDF, RDFS et OWL, et fournit en plus un moteur d'inférences permettant des raisonnements sur les ontologies.

3.5.3. OWL validator

Une fois qu'un document OWL est écrit, que ce soit à la main ou à l'aide d'un éditeur tel que Protégé, c'est bien de s'assurer de sa validité et de la cohérence des concepts qu'il exprime. D'une manière plus générale, le respect d'un standard ou de la définition d'un format favorise l'interopérabilité, en permettant au développeur de s'assurer de l'intégrité des données contenues dans le document.

Tout comme le W3C qui propose un validateur HTML (*http://validator.w3c.org/*), il existe différents validateurs d'ontologies OWL. Certains valident uniquement la syntaxe du document, tandis que d'autres vérifient également la cohérence des informations contenues dans l'ontologie.

a) Validateur RDF du W3C

Le validateur RDF du W3C (*http://www.w3.org/RDF/Validator/*) permet de valider des documents RDF. Il permet donc également de s'assurer qu'un document OWL respecte la syntaxe de RDF, ce qui donne déjà une première indication de la validité d'une ontologie.

b) WonderWeb OWL Ontology Validator

Le validateur *WonderWeb OWL Ontology Validator* **[BEC 05]** a été développé par Sean Bechhofer et Rafael Volz dans le cadre du projet WonderWeb (http://wonderweb.semanticweb.org/).

D'autres outils en relation avec le web sémantique sont proposés par WonderWeb ; ils sont disponibles à l'adresse http://phoebus.cs.man.ac.uk :9999/OWL. En outre du convertisseur d'ontologies OWL (http://phoebus.cs.man.ac.uk:9999/OWL/Converter), on notera surtout l'outil de génération de documentation «OWL Ontology HTML Presentation» (http://phoebus.cs.man.ac.uk:9999/OWL/Presentation), qui permet de créer des documentations agréables à consulter à partir de toute ontologie valide.

c) Validateur vOWLidator

Le validateur *OWL vOWLidator* **[RAG 2005]** effectue la validation de documents OWL en s'appuyant sur le framework Jena. Publié en open source, vOWLidator est disponible comme exécutable (Java est naturellement requis). Une version en ligne (*http://owl.bbn.com/validator//*) du validateur est également disponible et même si elle ne semble plus maintenue depuis Octobre 2003, elle fournit un moyen de valider rapidement des ontologies en ligne.

3.6. Agents de services web sémantique

Le pouvoir véritable du Web sémantique sera atteint quand les gens créeront de nombreux programmes qui collecteront les contenus du Web à partir de sources diverses, qui traiteront l'information et échangeront les résultats avec d'autres programmes. L'efficacité de ces agents logiciels croîtra de manière exponentielle au fur et à mesure que seront disponibles des contenus du Web lisibles par des machines et des services automatisés (comprenant d'autres agents). Le Web sémantique promeut cette synergie : même les agents qui ne sont pas

expressément fabriqués pour travailler ensemble peuvent transférer des données entre eux si les données sont accompagnées de sémantique.

Un aspect important du fonctionnement des agents sera l'échange de "*preuves*" écrites dans le langage unifié du Web sémantique (langage qui exprime les inférences logiques en utilisant des règles et des informations comme celles qui sont spécifiées par les ontologies).

3.7. Web Sémantique et Web Services

La notion de services correspond à une approche spécifique des ressources disponibles sur le Web qui met l'accent sur les fonctionnalités offertes par tel ou tel logiciel en termes d'un processus métier. Suivant la définition usuelle, un service permet à un utilisateur, non seulement d'obtenir de l'information, mais aussi d'effectuer des changements sur l'état du monde. Le commerce électronique est, bien sûr, un exemple privilégié de ces approches, mais l'exposition de tout processus fonctionnel sur le Web, comme la souscription d'une police d'assurance, relève également de cette problématique. Cette notion de services est aujourd'hui au coeur des stratégies d'un certain nombre des principaux acteurs du monde du logiciel comme Microsoft ou IBM.

De nombreuses techniques et plusieurs langages sont proposés comme des standards actuels ou futurs pour découvrir ou localiser les services Web (typiquement à travers un annuaire de services), pour invoquer ou activer ces services et les faire interopérer. De manière plus ambitieuse et moins aboutie aujourd'hui, il est proposé en plus de surveiller ou de suivre leur exécution (identifier les échecs, donner des traces) et surtout de les composer (sélection automatique, enchaînement et interopération). Parmi ces propositions, les plus connues sont :

- SOAP *(Simple Object Acess Protocol)* qui décrit les modalités de l'échange d'information entre services en termes d'enveloppes d'échange, de fichiers contenus et principalement de routage,
- Les annuaires UDDI (*Universal Description, Discovery and Integration*).
- WSDL (*Web Services Description Language*) qui décrit, de manière fonctionnelle et opérationnelle, comment utiliser un service.
- WSFL (*Web Services Flow Language*) pour décrire des enchaînements de services comme un processus.

Les approches services Web et celles du Web sémantique partagent le but commun de rendre l'information sur le Web plus accessible aux machines. Un certain nombre de recherches se proposent de les coupler. Une des idées partagées par ces travaux est de répondre aux limites de WSDL par l'ajout d'une couche sémantique (à base de marqueurs) au-dessus de WSDL décrivant le quoi et le pourquoi, pas seulement le comment.

3.7.1. Les services web

L'accès aux systèmes d'information s'appuie aujourd'hui de plus en plus sur des technologies Internet. Les efforts de standardisation dans ce contexte ont accentué l'engouement des personnes et des organisations (aussi bien académiques, qu'industrielles, commerciales, ou institutionnelles) pour l'utilisation de l'Internet et a permis l'émergence des services Web comme support de développement des applications accessibles par Internet.

Ainsi, les technologies associées aux services Web sont devenues incontournables pour le développement d'applications interagissant les unes avec les autres par le biais de l'Internet.

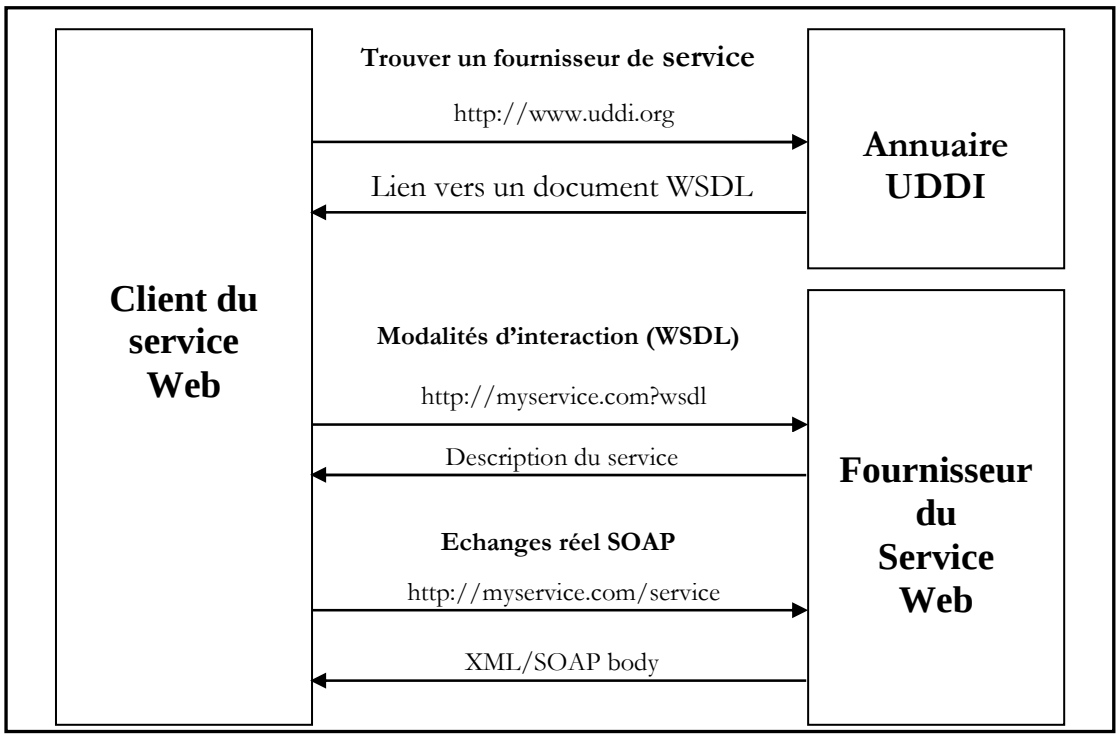

Figure 3.4: Utilisation d'un service Web [Théodoloz]

La figure 3.4 et 3.5 schématisent l'utilisation d'un service Web. Le client recherche un service Web dans un registre UDDI. Il reçoit ensuite l'URL d'un fournisseur du service souhaité, puis une description (WSDL) de la manière d'invoquer ce service.

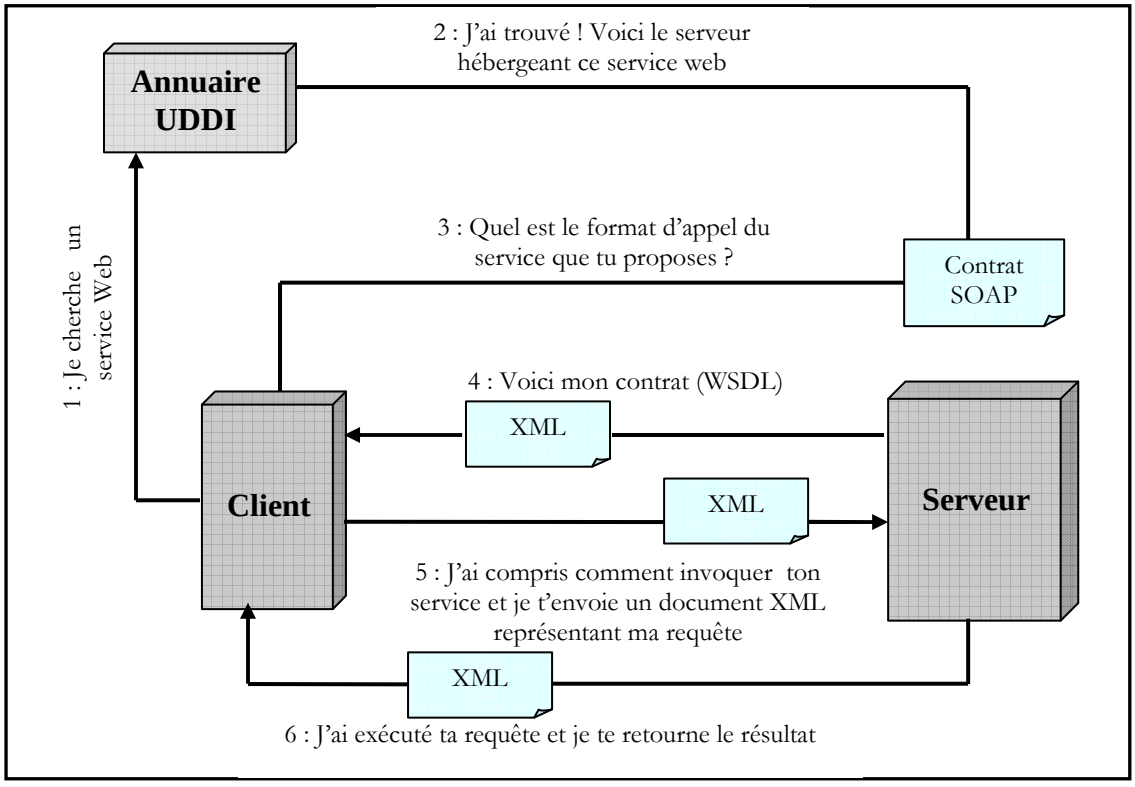

Figure 3.5: Cycle de vie d'utilisation [Jeremy]

3.7.2. SOAP

SOAP (*Simple Object Acces Protocol*) est une spécification de communication entre services Web par échange de messages en XML. Ces messages voyagent d'un émetteur vers un récepteur, les récepteurs peuvent former une chaîne. Ainsi, SOAP peut être utilisé suivant le modèle classique client serveur via HTTP ou suivant un modèle d'échange à sens unique. SOAP est indépendant des langages de programmation et des systèmes d'exploitation utilisés.

SOAP s'inspire des générations précédentes de RPC (*Remote Procedure Call*) et de RPC objet (IIOP de CORBA, DCOM). Le double objectif de SOAP est de servir d'une part de protocole de communication pour l'intégration d'applications d'entreprise existantes et d'autre part, de permettre la communication entre applications et services Web.

Un message SOAP est une enveloppe contenant un en-tête (header) facultative et un corps (body) du message :

- L'entête contient des entrées qui permettent d'ajouter des extensions au message, telles que la prise en charge de transactions ou d'éléments de sécurité.
- Le corps contient les données destinées au récepteur du message, comme un appel de procédure à distance ou une réponse à un appel.

Les messages SOAP sont transportés généralement à l'aide de requêtes HTML de type Post, mais SOAP peut également utiliser d'autres protocoles de transport tels que SMTP ou FTP.

Par exemple, l'appel à la méthode suivante :float GetLastTradePrice (string tickerSymbol) serait codé de cette manière :

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="some URL"
SOAP-ENV:encodingStyle="some URL">
<SOAP-ENV:Body>
        <m:GetLastTradePrice xmlns:m="Some URL">
                        <tickerSymbol>symbol</tickerSymbol>
        </m:GetLastTradePrice>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 3.6: Codage d'une méthode avec SOAP

3.7.3. WSDL

WSDL (*Web Services Description Language*) est un langage de la famille XML permettant de décrire les types de données supportés et les fonctions offertes par un service Web. L'objectif est de fournir la description, en XML, des services indépendamment de la plateforme et du langage utilisés et sous une forme que des personnes ou des programmes peuvent interpréter. Les descriptions WSDL sont en fait l'équivalent des interfaces IDL (*Interface Definition Language*) de CORBA par exemple.

Dans le langage WSDL, un service est vu comme une collection de messages pour les échanges et d'une collection de points d'entrée. Un point d'entrée consiste en la description abstraite d'une interface et de son implantation. La description abstraite contient : (i) La définition des messages qui sont consommés et générés par le service (les entrées et les sorties), et (ii) la signature des opérations offertes par le service. La mise en correspondance (implémentation binding) entre l'interface et son implantation est fournie. Elle contient essentiellement l'indication du protocole utilisé pour échanger des messages avec le service (par exemple SOAP au-dessus d'HTTP) et les associations entre la description de l'interface abstraite du service et les types de messages supportés par le protocole de communication sous-jacent (par exemple SOAP).

3.7.4. UDDI

UDDI (*Universal Description Discovery and Integration*) est une spécification qui définit une manière de publier et de découvrir des services Web. Elle recommande d'utiliser WSDL pour la description des services et SOAP comme couche de transport.

Un annuaire UDDI est constitué de pages blanches (noms et adresses d'entreprises), de pages jaunes (services Web classés par catégories industrielles) et de pages vertes (informations d'implémentation des services Web proposés). Ces annuaires sont des fichiers XML hébergés par des entreprises (opérateurs UDDI) dont la liste est disponible sur le site *http://www.uddi.org*.

La spécification UDDI fournit également une API permettant la recherche et la publication de services Web selon la spécification SOAP.

La publication d'un service dans UDDI requiert la création d'un fichier XML qui décrit le Web Service suivant les spécifications UDDI **[CCMW 01]**. Les principaux éléments de ce fichier sont :

- ✓ *BusinessEntity* : chaque service appartient à une organisation identifiée par un nom, une description, des personnes de contact, une ou plusieurs catégories, etc.
- ✓ *Business service* : inclut les informations non techniques relatives au service comme son nom et sa description.
- ✓ *BindingTemplate* : définit ou et comment accéder au service.
- ✓ *TModel* : comprends des liens vers les informations techniques du service, comme par exemple un fichier WSDL décrivant les différentes méthodes disponibles pour l'utilisation du service.

3.8. La recherche et la découverte de services

La mobilité des utilisateurs, la nature dynamique d'un environnement ubiquitaire et l'hétérogénéité des équipements impliqués dans un scénario de communication et de fourniture de service, nécessite le déploiement des mécanismes de découverte dynamique et de fourniture de ces services. Plusieurs mécanismes de découverte de services (on parle également de protocoles) ont été proposés ces dernières années (SLP, JLS, SSDP, Salutation, SDP, etc.).

Ces mécanismes de découverte de service sont particulièrement indispensables pour l'informatique ubiquitaire caractérisé non seulement par la mobilité des équipements, mais par la mobilité de l'utilisateur lui-même. En effet, les équipements dans ce type d'environnement peuvent se déplacer d'un réseaux à un autre sans connaître les infrastructures utilisées et la possibilité de se connecter à des environnements initialement inconnus, empêchant ainsi les utilisateurs et les équipements de tirer profit de certains services proposés et d'interagir avec eux. Il est donc nécessaire de déployer des mécanismes qui permettent aux utilisateurs de découvrir automatiquement ces services sans pour autant reconfigurer leurs terminaux à chaque accès. De plus, ces mécanismes doivent prendre en considération le contexte de la découverte de services afin de ne proposer à l'utilisateur que les services qui répondent au mieux à ses besoins.

La découverte de service peut être classée selon le niveau de connaissance initiale du client sur le service, et elle peut être divisée en deux catégories principales. Pré-configuré où les entités qui découvrent les services possèdent des informations sur le fournisseur du service concerné ou sur l'entité qui permette de récupérer ce genre d'information. Non-configurée ou les entités ignorent le contexte des services proposés.

Les deux catégories peuvent être, à leur tour, classées selon le nombre d'entités impliquées dans le processus de découverte de service. Deux modes, le mode sensible à la localisation et immédiat sont caractérisés par une relation directe entre le client et le fournisseur de service, tandis que le mode médiateur doit faire appel à une entité intermédiaire «le médiateurr» pour assurer la découverte d'information pour le compte des fournisseurs de services. Dans le mode immédiat, le client peut diffuser une requête dans le réseau pour retrouver un service, et les fournisseurs appropriés répondent, ce mode est le mode passif, dans le cas contraire, les clients n'envoient pas des requêtes. Ils écoutent les messages d'annonce diffusés par les fournisseurs de services, et dans ce cas, c'est un mode de communication passif.

Dans le mode à médiateur, on trouve le mode non-transparent, si le client utilise intentionnellement le médiateur pour retrouver les services, et le mode transparent, si le client pense qu'il interagit avec un fournisseur de service ordinaire alors qu'en réalité, il passe par un médiateur.

3.9. Protocole de découverte de services

Les protocoles de découverte de services fournissent des mécanismes pour la découverte spontanée et dynamique de services disponibles dans le réseau, les protocoles les plus représentatifs sont présentés dans la section suivante.

3.9.1. Service Location Protocol

Le protocole de localisation de services SLP (*Service Location Protocol*) **[Guttman 99]** de Sun Microsystems est un standard de l'*IETF* **[IETF]** pour la découverte spontanée de services dans les réseaux IP. Il est basé sur une infrastructure utilisant trois entités de base :

- ✓ *L'Agent Utilisateur (UA, User Agent)* qui envoie les requêtes de recherche de services pour le compte d'un client (utilisateur ou application).
- ✓ *L'Agent Service (SA, Service Agent)* qui annonce et affiche la localisation des services et leurs caractéristiques pour les comptes des services.
- ✓ *L'Agent Répertoire (DA, Directory Agent)* qui collecte dans sa base de données, les adresses des services et les informations reçues des SAs. Il répond également aux requêtes des UAs.

Lorsqu'un nouveau service se connecte au réseau, le SA contacte le DA pour annoncer son existence (Enregistrement Service). De même, si un client a besoin d'un service particulier, l'UA envoie une requête de recherche de services au DA (Requête Service).

SLP propose trois méthodes pour la découverte de DA : statique, active et passive. Dans la découverte statique, les agents SLP obtiennent l'adresse du DA grâce au protocole DHCP (*Dynamic Host Configuration Protocol*).

Dans la découverte active, les UAs et les SAs envoient les requêtes à une adresse de diffusion allouée à SLP. Dans le cas de la découverte passive, les DAs diffusent périodiquement les messages de présence sur le réseau.

La découverte de services s'effectue selon deux modes : mutlicast et unicast. Le mode unicast est utilisé en cas de présence d'un agent DA. Les agents UA à la recherche de services s'adressent directement à l'agent DA qui enregistre dans son annuaire les annonces de services des agents SA. Le mode multicast est utilisé quant à lui dans le cas où il n'y aurait pas d'agent DA présent sur le réseau. Les agents UA diffusent dans ce cas et de manière continuent leurs requêtes de services sur le réseau ; les agents SA, qui sont en écoute, détectent ces requêtes et répondent directement aux agents UA concernés.

SLP supporte la recherche de services basée sur les mots-clés. Il compare ces derniers aux attributs de chaque service disponible dans le réseau. Si le service est conforme à la requête, l'URL du service est envoyée au client.

SLP offre également aux UAs la possibilité de formuler des requêtes expressives en utilisant des opérateurs booléens (OR et AND), des comparateurs (<, > =), … ou des sous-chaînes.

3.9.2. Le service lookup de Jini

Jini de Sun Microsystems est un environnement Java distribué qui supporte la découverte spontanée de services **[JINI]**. Les services Jini peuvent être des équipements matériels, des applications et des programmes logiciels ou une combinaison des deux. Ces derniers peuvent s'organiser en communautés d'objets et accéder aux services du réseau de manière flexible.

Jini utilise trois entités de base : les *services* à proposer aux clients, les *lookup Services* (LSs) qui cataloguent les services disponibles et, les *clients* qui utilisent les services. Pour enregistrer un nouveau service ou découvrir les services disponibles, les clients et les services doivent d'abord localiser un des LSs présents dans le réseau.

Le service de découverte proposé dans Jini, appelé service de consultation (*Jini Lookup service JLS*), est basé sur un service d'annuaire, similaire en partie à l'agent DA de la norme SLP. Notons que le service JLS est nécessaire au fonctionnement de chaque plateforme Jini et qu'il est considéré comme l'unique moyen dont disposent les objets clients pour découvrir les services recherchés.

Une requête de recherche de services envoyée par le client est une instance de la classe *ServiceTemplate* de Jini. Le client peut spécifier l'identificateur du service à rechercher, son type ou les valeurs de ses attributs. Une fois le service découvert et sélectionné, le LS retourne au client le *proxy* correspondant. Le client télécharge ensuite le code du proxy si ce dernier n'existe pas sur sa machine et la communication entre le client et le service se fait par l'invocation d'une méthode spécifique du proxy.

3.9.3. Universal Plug and Play (UPnP) et Simple Service Discovery Protocol (SSDP)

UPnP **[CL 99]** est une architecture proposée par Microsoft pour déployer une infrastructure de réseau permettant la communication entre différents appareils de marques et de technologies différentes. UPnP vise à déployer un réseau de communication dans les maisons et dans les locaux d'entreprise où les membres communicants ne sont pas forcément des PCs.

UPnP permet aux dispositifs de joindre dynamiquement un réseau, obtenir une adresse IP, transmettre des services et connaître les équipements déjà connectés ainsi que leurs services.

La découverte des services dans UPnP est basée sur le protocole SSDP (*Simple Service Discovery Protocol*) **[GCPLA99]**.

Le protocole SSDP est considéré comme un protocole léger qui peut fonctionner avec ou sans présence d'un annuaire de service. Quand un service désire intégrer le réseau, il envoie au préalable un message pour s'annoncer auprès des dispositifs présents. Toute annonce peut être envoyée en mode multicast pour être détectée par tous les dispositifs présents. Si l'annuaire de services est présent, il enregistrera cette annonce. Alternativement, l'annonce peut être envoyée en mode unicast directement à l'annuaire de service. Lorsqu'un client veut découvrir un service, il peut soit envoyer un message en multicast sur le réseau, soit contacter directement le service via son adresse URL récupérée suite à l'annonce de sa présence. Dans le premier cas la réponse à la requête est envoyée en unicast par le service lui-même.

3.9.4. Le protocole Salutation

Salutation est un protocole de découverte de service développé par le consortium Salutation **[Salutation 99]**. C'est un standard ouvert indépendant des systèmes d'exploitation, des protocoles de communication et des plateformes physique.
Salutation a été créé pour résoudre les problèmes de découverte et d'utilisation de services dans les environnements caractérisés par l'hétérogénéité et la mobilité de leurs dispositifs. Salutation assure l'interopérabilité entre des entités réseau ayant des environnements d'exécution hétérogènes et utilisant des protocoles de transport différents.

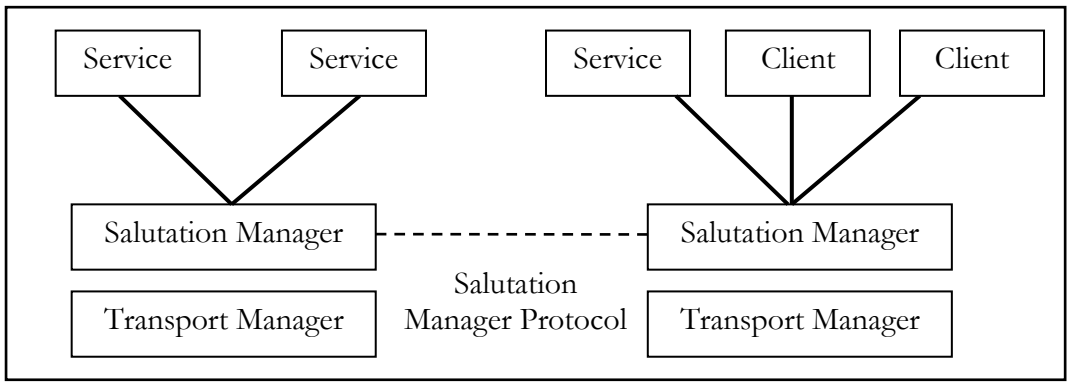

Figure 3.7: Architecture de Salutation [Salutation 99]

Salutation est composé de deux principaux composants (Figure 3.7) : le SLM (*Salutation Manager)* et le TM *(Transport Manager)*. Le SLM est le cœur de l'architecture. C'est l'annuaire des descriptions de services disponibles. Il fonctionne au-dessus du TM qui assure des communications fiables indépendamment des protocoles de transport réseau utilisés.

Un même SLM peut présenter un ou plusieurs clients et/ou services. Il peut découvrir d'autres SLMs et les services qu'ils proposent. La coopération entre les SLMs forme conceptuellement un service lookup similaire à celui de Jini. La seule différence est qu'il est distribué dans le réseau. La communication entre les SLMs se fait via RPC *(Remote Procedure Call)* qui permet, même à des clients et à des services ne possédant pas de SLM, d'utiliser des SLMs distants **[Rekesh 99]**.

3.9.5. Secure Service Discovery Service

SSDS **[CZH 99]** est une architecture de déploiement et de découverte de services à grande échelle. Elle inclut principalement des serveurs, un gestionnaire des informations sur les services disponibles dans les domaines qu'il couvre.
Les principaux composants du système SSDS et leur rôle dans le processus de découverte de services sont les suivants :
Serveur SDS (*Service Discovery Service*) : les serveurs SDS jouent le rôle d'annuaires de services. Ils sont organisés en domaines hiérarchiques où chaque domaine spécifie une étendue dans le réseau. Les serveurs SDS disponibles diffusent périodiquement en multicast des messages authentifiés contenant leur adresse URL.
Services : les services écoutent les messages des serveurs SDS pour récupérer leurs adresses ; ils diffusent ensuite, en multicast, les descriptions des services chiffrées et authentifiées.

Client : les clients découvrent le serveur SDS de leur domaine en écoutant sur une adresse SDS connue. Ils utilisent un protocole RMI authentifié **[CZH 99, Wel99]** pour contacter le serveur SDS et pour envoyer les requêtes de découverte de services en format XML.

Autorité du certificat CA (*Certificate Authority*) : le SDS utilise des certificats signés par le CA pour l'authenticité des liens entre les composants du système SSDS et leurs clés publiques.

Gestionnaires des capacités CM (*Capability Manager*) : le SDS utilise les capacités (des clés privées appropriées) comme mécanisme de contrôle d'accès pour permettre aux services de contrôler l'ensemble des utilisateurs autorisés à découvrir leur existence.

Comparé aux autres protocoles de découverte de services, SSDS permet d'apporter des améliorations considérables par rapport à la fiabilité, le passage à l'échelle et la sécurité.

3.9.6. Bluetooth SDP

Bluetooth SDP (*Service Discovery protocol*) **[SDP 03]** est une technologie sans fil qui permet aux utilisateurs de faire des connexions faciles, sans fil et instantanées entre des dispositifs de communication divers comme les téléphone portables, les ordinateurs et les PDAs.

La pile protocolaire de Bluetooth contient le protocole SDP (Figure 3.8) de découverte de services fournis par les terminaux Bluetooth **[SDP 99]** ;

SDP permet à un client SDP d'accéder à des informations sur des services proposés par des serveurs SDP. Un serveur SDP peut être n'importe quel terminal Bluetooth qui fournit un service à utiliser par un autre terminal Bluetooth. Un client SDP peut être n'importe quel terminal Bluetooth pouvant utiliser les services. Un terminal Bluetooth peut être à la fois, un client SDP et un serveur SDP.

Contrairement aux autres protocoles de découverte de services, le SDP de Bluetooth est spécifique aux terminaux Bluetooth. Il ne fournit pas de mécanisme permettant d'accéder aux services et d'annoncer ou d'enregistrer un nouveau service. Pour pouvoir utiliser un service découvert par le protocole SDP, le terminal doit faire appel à un autre protocole de haut niveau pour effectuer toutes les tâches nécessaires à l'accès et à l'utilisation du service, donc il ne dispose ni d'un annuaire de services, ni d'un système de gestion d'abonnement. Pour combler ces lacunes, le protocole SDP est associé à Salutation ou Jini dans certaines applications.

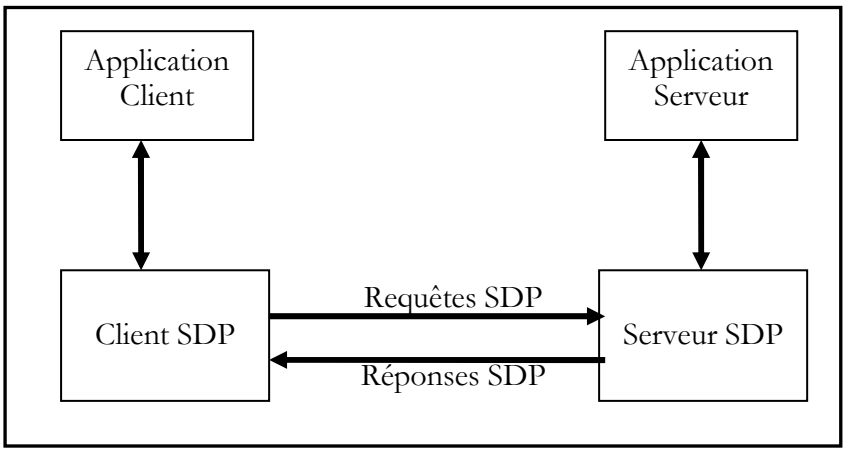

Figure 3.8: Protocole SDP dans Bluetooth [SDP 03]

3.10. Limites des systèmes existants pour les environnements mobiles

Les protocoles de découverte de services présentés dans les sections précédentes, sont bien utilisés dans les systèmes distribués, mais ils possèdent des limitations qui les rendent non exploitables dans les environnements mobiles.

L'un des limitations des ces protocoles est qu'ils n'exploitent pas les informations concernant le contexte de découverte de services telles les préférences de l'usager et les capacités des ressources disponibles (réseau et dispositif).

En plus, ces protocoles de découverte de services n'utilisent pas des langages expressifs de description de services qui permettent de représenter convenablement les différentes fonctionnalités d'un service.

Et enfin, les infrastructures de découverte de services existants n'utilisent pas de formalismes ou de standard pour la description de leurs services tels que les ontologies.

Chapitre 4 :

Architecture intergicielle sémantique basée sur le standard OSGi

4.1. Introduction

Avant de présenter le modèle de l'intergiciel pour la mise en œuvre d'applications sensibles au contexte que nous proposons, une introduction au paradigme des agents et des systèmes multi agents est nécessaire, afin de montrer l'intérêt d'un tel système dans la conception et la mise en œuvre des systèmes complexes en général et des environnement ubiquitaires intelligents en particulier.

Un environnement ubiquitaire est perçu comme un environnement multi agents, intégrant de façon transparente des agents intelligents, communicants, et autonomes.

Dans la suite de ce chapitre, nous présentons le modèle de l'architecture de l'intergiciel sensible au contexte basé sur le standard OSGi inspiré du travail de *Chibani* **[Chibani 06]**, et nous présentons son application dans le domaine de la robotique, ainsi que le modèle de son architecture multi agents.

4.2. L'intelligence artificielle distribuée

L'un des objectifs de l'Intelligence Artificielle (*IA*) est la définition de systèmes capables de représenter des connaissances, de raisonner, de planifier des actions afin de résoudre des problèmes pouvant être très complexes. Elle cherche à obtenir des résultats comparables à ce que feraient des êtres humains dans des cas similaires, mais sans forcément utiliser les mêmes moyens.

Mais, pour résoudre ce problème de complexité, il est devenu nécessaire de décomposer un système en plusieurs sous-systèmes moins complexes capables d'interagir entre eux, afin de résoudre le problème global. En d'autres termes, l'*IAD* postule que la conception des systèmes intelligents peut se faire grâce à un ensemble d'entités (sous-systèmes) plus simple où chacune s'occupe d'une partie du problème, tout en ayant au final une complexité globale moindre. L'IAD est donc la solution collaborative à des problèmes globaux par un groupe d'entités distribuées. Les agents peuvent être des éléments d'exécution simples ou bien des entités complexes exhibant un comportement rationnel.

Trois axes de distribution de l'intelligence artificielle sont apparus :

4.2.1. Axes de recherche de l'IAD

Nous distinguons trois axes fondamentaux dans la recherche en IAD :

- Les systèmes multi agents (*SMA*) : Il s'agit de faire coopérer un ensemble d'agents dotés d'un comportement intelligent et de coordonner leurs buts et leurs plans d'actions pour la résolution d'un problème.

- La résolution distribuée des problèmes (*RDP*) : Elle s'intéresse à la manière de diviser un problème particulier sur un ensemble d'entités distribuées et coopérantes. Elle s'intéresse aussi à la manière de partager la connaissance du problème et d'en obtenir la solution.
- L'intelligence artificielle parallèle (*IAP*) : Elle concerne le développement de langages et d'algorithmes parallèles pour l'IAD. L'IAP vise l'amélioration des performances des systèmes d'intelligence artificielle sans, toutefois, s'intéresser à la nature du raisonnement ou au comportement intelligent d'un groupe d'agents. Cependant, il est vrai que le développement de langages concurrents et d'architectures parallèles peut avoir un impact important sur les systèmes d'IAD.

4.3. Le concept d'agent

4.3.1. Définitions

Il n'existe pas, actuellement, une définition de l'agent qui fasse foi dans le monde de l'intelligence artificielle distribuée. Il est donc nécessaire, pour avoir une bonne vision de ce concept, de confronter plusieurs de ces définitions.

Jacques Ferber **[Ferb 95]** définit un agent comme étant une entité physique ou virtuelle évoluant dans un environnement dont il n'a qu'une représentation partielle et sur lequel il peut agir (Figure 4.1). Il est capable de communiquer avec d'autres agents et est doté d'un comportement autonome. Cette définition aborde une notion essentielle : l'autonomie. En effet, ce concept est au centre de la problématique des agents. L'autonomie est la faculté d'avoir ou non le contrôle de son comportement sans l'intervention d'autres agents ou d'êtres humains. Une autre notion importante abordée par cette définition concerne la capacité d'un agent à communiquer avec d'autres.

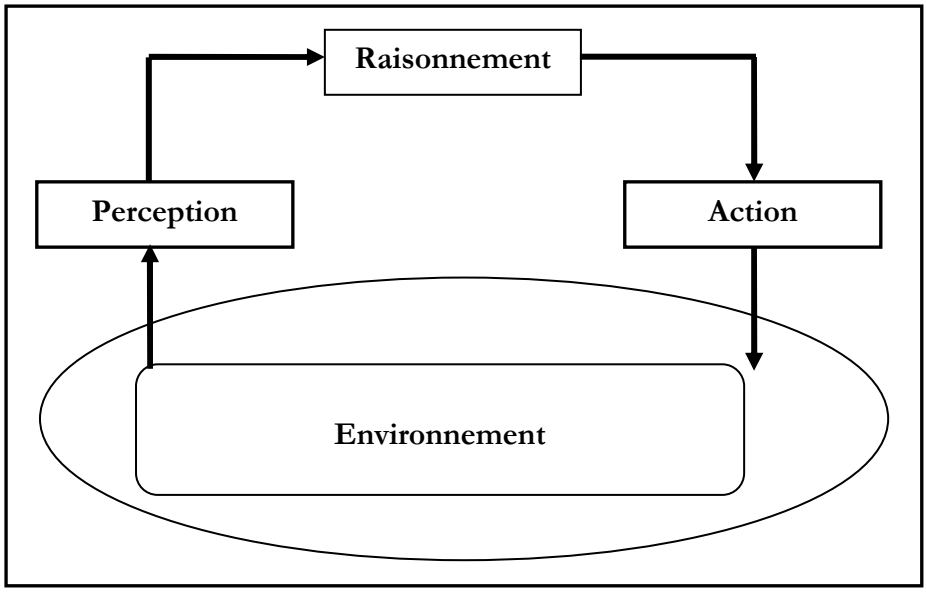

Figure 4.1: Fonctionnement d'un agent

Yves Demazeau **[Demazeau]** définit l'agent comme toute entité réelle ou virtuelle dont le comportement est autonome, évoluant dans un environnement qu'il est capable de percevoir et sur lequel il est capable d'agir, et d'interagir avec les autres agents. Cette définition introduit

l'interaction (la communication, l'échange d'informations et l'action sur le monde) qui est le moteur des systèmes multi agents. En effet, l'interaction suppose la présence d'agents capables de se rencontrer, de communiquer, de collaborer et d'agir.

Quand au *Ferber* **[Ferb 95]**, Un agent est une entité située, réelle ou virtuelle, agissant dans un environnement, capable de le percevoir, d'agir sur celui-ci et d'interagir avec les différents composants l'entourant. Une entité est un agent si elle est capable d'exercer un contrôle local sur ses processus de perception, de communication, d'acquisition de connaissances, de raisonnement, de prise de décision et d'exécution.

Et enfin *Wooldridge* **[Wool 02]** définit l'agent comme un système informatique capable d'agir de manière autonome et flexible dans un environnement changeant. Par flexibilité on entend :

- ✓ *Réactivité* **:** un système réactif maintient un lien constant avec son environnement et répond aux changements qui y surviennent,
- ✓ *Pro-activité* **:** un système proactif (aussi appelé téléonomique) génère et satisfait des buts, son comportement n'est donc pas uniquement dirigé par des événements,
- ✓ *Capacités sociales* **:** un système social est capable d'interagir ou coopérer avec d'autres systèmes.

À partir de ces définitions, nous pouvons définir un agent comme :

- une entité autonome qui peut offrir des services,
- Une entité dont le comportement est la conséquence de ses objectifs, de sa perception, de ses représentations, de ses compétences et des communications qu'elle peut avoir avec les autres agents,
- Une entité qui possède des ressources,
- Une entité qui est apte à agir sur l'environnement du système auquel il appartient,
- une entité qui peut communiquer avec les autres agents,
- Une entité qui est capable de se reproduire.

4.3.2. Principales caractéristiques d'un agent

Un agent doit posséder les caractéristiques suivantes :

- ✓ Il est autonome ou semi autonome : il participe dans la résolution du problème avec peu de connaissances de ce que les autres agents font. Chaque agent a une partie du problème, soit il produit les résultats lui-même, ou bien les ramène d'un autre agent de l'organisation.
- ✓ Il est situé : On appelle agent purement situé **[Ferber 95]**, une entité physique (ou éventuellement informatique si elle est simulée) qui se situe dans un environnement, qui est mue par une fonction de survie, qui possède des ressources propres, sous la forme d'énergie et d'outils, qui est capable de percevoir (mais de manière limitée) son environnement, qui ne possède pratiquement aucune représentation de son environnement, qui possède des compétences, dont le comportement tend à satisfaire sa fonction de survie, en tenant compte des ressources, des perceptions et des compétences dont elle dispose.

- ✓ Il est interactionnel : les agents forment une collection d'individus coopérants pour accomplir une tâche particulière. Dans ce sens, on peut les considérer comme une société qui présente des aspects comme la connaissance, la compétence et la responsabilité qu'on peut les associer à chaque individu de cette société.
- ✓ L'agent est réactif : peut percevoir son environnement, via des capteurs, et peut agir sur son environnement via des effecteurs ou actionnaires.
- ✓ L'agent est proactif : capacité de l'agent à percevoir son environnement et interagir avec lui (prendre de l'initiative).
- ✓ La société des agents est structurée : dans la plupart des cas de résolutions des problèmes par agents, un agent, même s'il présente un état unique et de propres compétences, il va coordonner avec les autres agents dans la totalité de la résolution. Ainsi, le résultat final sera à la fois collectif et coopératif.
- ✓ L'intelligence est émergente : même si l'agent possède individuellement des connaissances et des compétences, la totalité de la résolution obtenue est supérieure à la somme des efforts individuels. L'intelligence est vue comme un phénomène résident dans la société des agents et émerge d'elle, et elle n'est pas seulement une propriété de l'agent.

4.3.3. Typologie des agents

Il existe en fait plusieurs types d'agents, qui selon les capacités qu'ils possèdent, seront qualifiés de réactifs, cognitifs ou hybrides.

4.3.3.1. Les agents réactifs

Les défenseurs de cette approche partent du principe suivant : dans un système multi agents, il n'est pas nécessaire que chaque agent soit individuellement « *intelligent* » pour parvenir à un comportement global intelligent. En effet, des mécanismes simples de réactions aux évènements peuvent faire émerger des comportements correspondant aux objectifs poursuivis.

Cette approche propose la coopération d'agents de faible granularité (fine grain) mais beaucoup plus nombreux. Les agents réactifs sont de plus bas niveau, ils ne disposent que d'un protocole et d'un langage de communication réduits, leurs capacités répondent uniquement à la loi stimulus/action.

Les premiers travaux relatifs à cette approche ont été réalisés en 1986 par *R. Brooks* **[Broo 89]**. D'après lui, plusieurs milliers de micro-robots identiques, d'une taille aussi petite que possible, travaillant ensemble sur une tâche donnée pourront être plus efficaces qu'un seul gros robot spécialisé.

4.3.3.2. Les agents cognitifs

Un système cognitif comprend un petit nombre d'agents **[Dieng90]** dont chacun est assimilable, suivant le niveau de ses capacités, à un système expert plus ou moins sophistiqué, on parle d'agent de forte granularité.

Ils regroupent plusieurs sous-types d'agents définis de la façon suivante :

- Les « *agents intelligents* » **[TrEsp 99]** combinent les trois caractéristiques (autonomie, coopération, adaptativité) à leur plus haut niveau. Ils sont capables de négocier avec d'autres

agents et d'acquérir ou de modifier leurs connaissances. Ainsi, ils planifient leurs actions. Le terme agent intelligent est souvent utilisé pour caractériser des agents dotés de la capacité d'apprentissage.

- Les « *agents collaborants* » **[Nwa 96]** : ce sont des agents cognitifs non apprenants. Ils sont donc à la fois fortement autonomes et coopérants, mais leur adaptativité ne va pas jusqu'à l'adaptation et à l'acquisition des connaissances. Les agents collaborants sont surtout utilisés dans les domaines qui nécessitent une décentralisation comme par exemple la maintenance de réseau, ou encore pour simuler le comportement d'organisations humaines ou animales **[Bourr]**.
- Les « *agents interfaces* » **[Koda 96]** sont souvent utilisés dans les logiciels de bureautique. Leur fonction consiste à capturer les actions de l'utilisateur (le plus souvent les actions sur le clavier, la souris, la capture de la voix ou de l'expression du visage par utilisation d'une webcam, caméra dédié aux communications Internet). Ces agents possèdent en général une capacité de coopération limitée à l'échange d'informations concernant les actions de l'utilisateur.
- Les « *agents informations* » **[Rho00, RGK97]** sont dédiés à la recherche d'information, principalement sur l'Internet. Ces agents (tels que *softbot* **[Etz94]**] possèdent une grande autonomie ; ils agissent seuls, soit en fonction d'un calendrier, soit en fonction d'un manque d'information, soit en fonction d'une nouvelle disponibilité d'information. Ils sont capables d'adapter leurs fonctionnements en fonction du besoin de l'utilisateur ou de la quantité ou pertinence de l'information (par exemple, si un nouveau site propose des informations plus pertinentes, ce site sera alors privilégié pour les recherches futures). Toutefois, ces agents agissent le plus souvent de manière isolée, ce qui peut entraîner un problème de redondance d'informations.

Le tableau 4.A résume les différentes propriétés des deux précédentes approches :

Systèmes d'agents cognitifs	Systèmes d'agents réactifs
Représentation explicite de l'environnement	Pas de représentation explicite
L'agent peut tenir compte de l'historique passé	Pas de mémoire de son historique
Agents complexes	Fonctionnement stimulus/réponse
Petit nombre d'agents	Grand nombre d'agents

Tableau 4-A: Les agents cognitifs et réactifs

4.3.3.3. Les agents hybrides

Les agents hybrides sont des agents ayant des capacités cognitives et réactives. Ils conjuguent en effet la rapidité de réponse des agents réactifs ainsi que les capacités de raisonnement des agents cognitifs. Cette famille regroupe donc des agents dont le modèle est une compromise autonomie/coopération et efficacité/complexité.

4.3.4. Architecture d'agents

L'architecture de l'agent est le mappage des composants internes de l'agent : ses structures de données, les opérations qu'il peut effectuer sur ces structures, et le contrôle de flux entre eux.

Ces composants sont définis selon le comportement de l'agent et en fonction de l'interaction de l'agent avec son environnement.

D'une façon générale, les agents ont une architecture qui leur permet d'être réactifs, cognitifs ou coopérants (Figure 4.2).

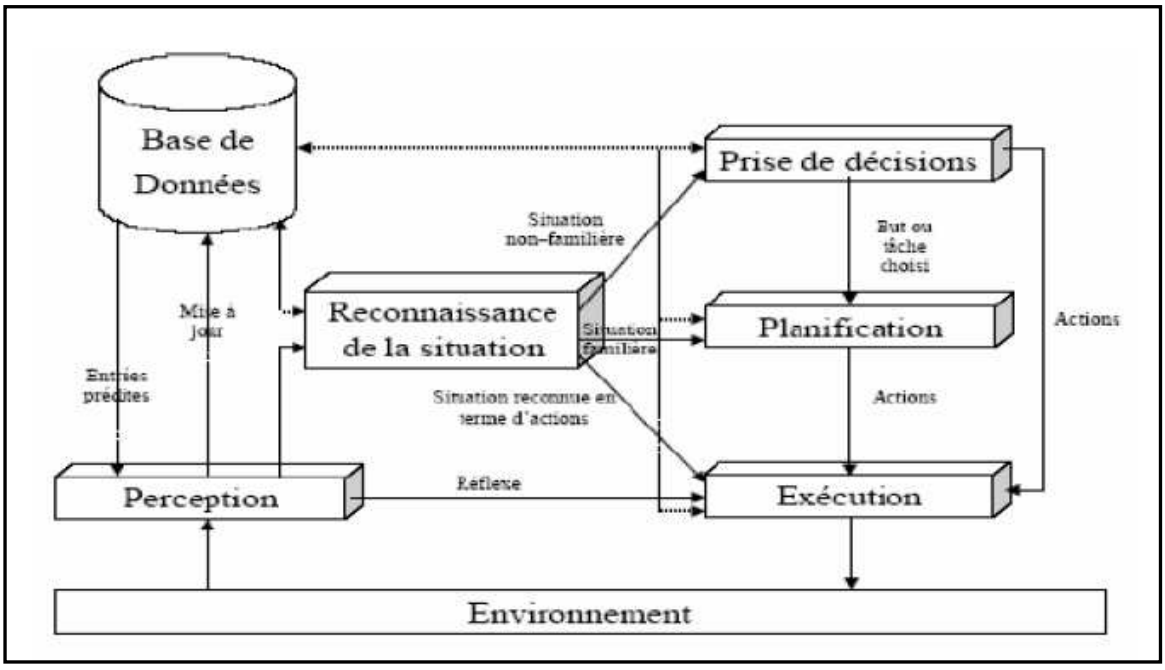

Figure 4.2: Architecture d'agent [Marc 99]

Si l'agent perçoit des situations qu'il connaît très bien, alors, on parle d'un processus réactif. Par contre, on dit qu'un processus est cognitif, lorsque l'agent rencontre ou perçoit une situation familière et qu'il lui suffit un simple raisonnement pour la résoudre. Dans certains cas, il peut toutefois s'avérer qu'un agent ne peut pas résoudre un problème (situation non familière), l'agent aura donc besoin d'engager un processus de coopération pour demander de l'aide aux autres agents.

En se basant sur cette structure, *Russell* et *Norvig* **[Russell 95]**, estiment qu'il existe quatre principales architectures d'agents. Ces architectures se distinguent par le niveau de complexité de l'agent.

4.3.4.1. Agent réflexe simple

Ce type d'agents ne fait que réagir à des stimulus provenant de son environnement. Ces agents n'ont aucune représentation de leur environnement, aucune base de connaissances et aucune forme de mémoire. Les trois composants de cette architecture sont :

- Les capteurs (*sensors*) qui servent à percevoir les changements de leur environnement.

- Un ensemble de règles et de conditions (ou une fonction) déterminant les actions à effectuer.
- Les effecteurs qui permettent aux agents d'agir sur leur environnement.

4.3.4.2. Agent réflexe à états

Ce type d'agents est une version améliorée de l'architecture précédente. Ces agents possèdent aussi des capteurs, des effecteurs et des règles (ou fonctions). De plus, ce type d'agents possède différents états et connaît les résultats des actions qu'il peut exécuter. Ces agents peuvent donc mieux déterminer quelle action poser en fonction des stimulus qu'ils perçoivent de leur environnement via leurs capteurs.

4.3.4.3. Agent à base de buts

Les agents à base de buts possèdent tous les composants des deux architectures précédentes. De plus, ces agents connaissent les conséquences de leurs actions sur leur environnement. Ce type d'agents agit en fonction de l'atteinte d'un but. De cette façon, si un agent détermine que l'exécution d'une action lui permettra d'atteindre son but, alors l'agent posera assurément cette action (lorsque c'est possible).

4.3.4.4. Agent à base d'utilité

Contrairement à l'architecture à base de buts où les agents posent des actions aléatoires lorsqu'ils n'étaient pas en mesure d'atteindre leur but, les agents à base d'utilité peuvent déterminer l'action (ou les actions) à effectuer pour se rapprocher le plus possible du but à atteindre. Cela signifie que lorsqu'un agent ne peut atteindre son but en posant une action, alors il sélectionnera l'action qui le rapprochera le plus possible du but à atteindre ou l'action qui lui permettra d'atteindre ce but le plus rapidement possible. C'est à cette étape que l'on applique la fonction d'utilité pour déterminer l'action à poser.

4.4. Système Multi Agents (SMA)

Les systèmes multi agents s'appuient sur le principe suivant : au lieu d'avoir un seul agent en charge de l'intégralité d'un problème, on considère plusieurs agents qui n'ont chacun en charge qu'une partie de ce problème. La solution du problème initial est alors obtenue au travers de l'ensemble des comportements individuels et des interactions, c'est-à-dire par une résolution collective.

4.4.1. Définitions

Un système multi agents est un ensemble d'agents qui évoluent dans un environnement commun. *Ferber* **[Ferb 95]**, définit un système multi agents de la façon suivante (figure 4.3) :

On appelle système multi agents (ou SMA), un système composé des éléments suivants :

- ✓ Un environnement *E,* c'est-à-dire un espace disposant généralement d'une métrique.
- ✓ Un ensemble d'objets *O* situé dans *E*. Ces objets sont passifs, c'est-à-dire qu'ils peuvent être perçus, créés, détruits et modifiés par les agents.
- ✓ Un ensemble *A* d'agents, qui représentent les entités actives du système.

- ✓ Un ensemble de relations *R* qui unissent des objets (et donc des agents) entre eux.
- ✓ Un ensemble d'opérations *Op* permettant aux agents de *A* de percevoir, produire, consommer, transformer et manipuler des objets de *O*.
- ✓ Des opérateurs chargés de représenter l'application de ces opérations et la réaction de l'environnement envers les tentatives de modification.

Selon *Christophe*, il existe deux grands types de systèmes multi agents :

- **Système multi agents ouvert :**

Un système multi agents ouvert **[Christophe 05]**, partage les caractéristiques des systèmes ouverts. Pour ce qui concerne les entités élémentaires du système que sont les agents, ils n'ont pas la possibilité d'avoir une représentation complète de l'environnement. Pour ce qui est du système dans sa globalité, il doit être modulaire et extensible. La modularité concerne le fait que le système multi agents est composé de plusieurs sous-systèmes mis en relation. Ces sous-systèmes ont chacun leur propre mode de fonctionnement. L'extensibilité se traduit par le fait que le système multi agents supporte l'ajout et le retrait dynamique d'éléments.

À l'inverse, le qualificatif fermé signifie que l'ensemble des agents qui compose le système reste le même.

- **Système multi agents homogène/hétérogène**

Un système multi agents homogène est composé d'agents homogènes. Deux agents ont cette particularité s'ils sont identiques du point de vue de leur modèle et de leur architecture.

Le qualificatif hétérogène est utilisé pour préciser que le système multi agents est composé d'agents différents du point de vue de leurs modèles et de leurs architectures.

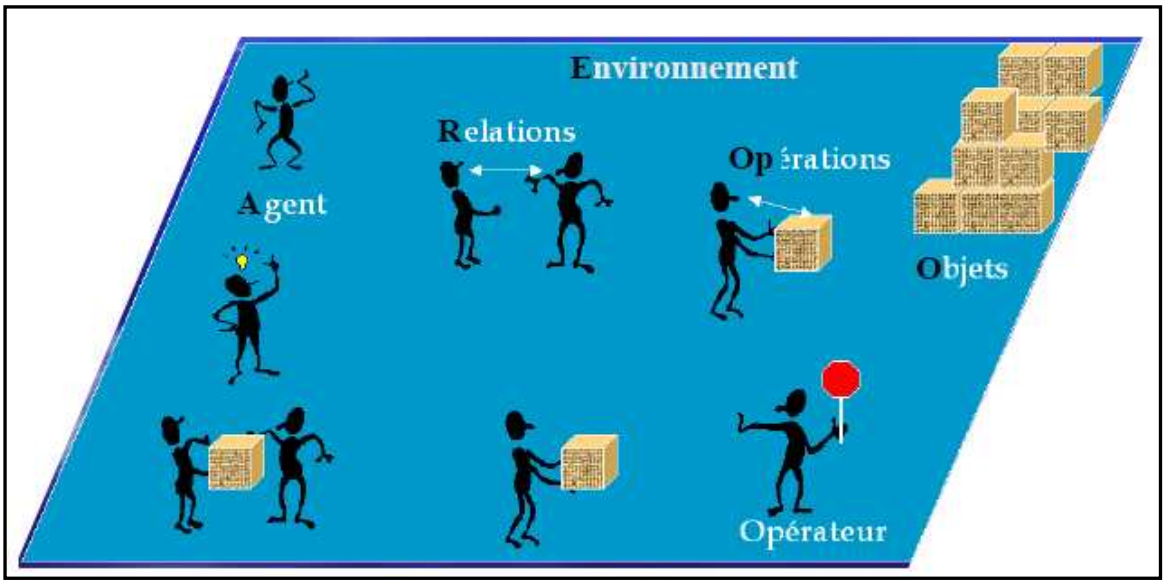

Figure 4.3: Un exemple de Système Multi agents [Ferb 95]

4.4.2. Les caractéristiques d'un SMA

Nous pouvons citer les caractéristiques suivantes :

- Autonomie des agents (Entre eux, incluant les utilisateurs),
- Décentralisation du contrôle : métaphore des feux de circulations (Asynchronisme),
- Différents types de distributions qui peuvent être prises en compte :
  - ✓ Distribution physique (traitements, données),
  - ✓ Distribution de la compétence des agents : faire mieux à plusieurs que seul,
  - ✓ Distribution des rôles et des buts des agents.
- Hétérogénéité/Homogénéité :
  - ✓ Hétérogénéité : les SMA constitué d'agents différents on l'avantage pour faciliter l'intégration d'agents non initialement prévu (Ouverture), ce type est plutôt pour des agents cognitifs et en petite quantité,
  - ✓ Homogénéité : SMA est constitué d'agents identiques, Plutôt pour les agents réactifs et en grande quantité.
- Utilisateurs : les utilisateurs peuvent être intégrés au SMA. Les interfaces servent à intégrer l'utilisateur dans l'environnement défini dans le SMA (Si l'environnement n'est pas physique), un agent représente un utilisateur dans un environnement (généralement) virtuel
  - ✓ Personnalisation de l'agent qui trace le profil de l'utilisateur pour s'adapter aux besoins de celui-ci.
  - ✓ Connaissances dans les SMA : les connaissances et leur traitement ne sont pas un objet d'étude central des SMA.

4.4.3. Types d'architectures des SMA

Les SMA peuvent être conçus selon plusieurs types d'architectures. Le choix de l'architecture dépend du niveau de complexité de la modélisation, du développement et surtout de l'implémentation du système. Souvent, les architectures qui tendent vers la décentralisation se modélisent plus difficilement. Par contre, les systèmes adoptant une architecture plus centralisée (ou hiérarchisée) sont relativement plus simples à concevoir. En effet, il existe plusieurs autres architectures hybrides empruntant quelques caractéristiques à plusieurs architectures.

4.4.3.1. Systèmes centralisés (blackboard systems)

L'architecture *Blackboard* est appliquée pour résoudre les problèmes qui nécessitent la coordination de multiples processus ou sources de connaissances. Un Blackboard (tableau noire) est une base de données centrale et globale, pour la communication entre des sources de connaissances indépendantes et asynchrones, qui travaillent sur certains aspects d'un problème particulier.

Parmi ses caractéristiques, on distingue :

- L'indépendance de l'expertise : chaque source de connaissances focalise sur un aspect du problème, ainsi elle contribue dans la solution avec une expérience indépendante de celles des autres sources.
- La diversité des techniques de résolution : le mécanisme de résolution interne d'une source n'est pas le même d'une autre source.

- Représentation flexible de l'information sur le blackboard : le système ne pose pas des restrictions pour la représentation de l'information.
- Langage d'interaction commun : la représentation des informations sur le blackboard doit être comprise par toutes les sources.
- Activation par événement : les sources sont activées avec le déclenchement des événements (les changements des informations sur le blackboard).
- Génération de solution incrémentale : Les sources contribuent dans la génération de la solution, elles initialisent le travail, raffinent, et parfois contredisent la solution trouvée.

4.4.3.2. Les systèmes hiérarchisés

Ces systèmes se basent sur une structure où les entités répondent à leurs supérieurs hiérarchiques. Ce type de système est relativement simple à implémenter. Plusieurs systèmes utilisent cette architecture pour la réalisation de SMA. Cette architecture comporte des faiblesses, parmi lesquelles, nous pouvons citer :

- Le peu de tolérance aux fautes,
- La limite imposée par les capacités de son supérieur hiérarchique, etc.

4.4.3.3. L'approche totalement distribuée

Cette approche est de loin la plus puissante mais aussi la plus complexe à développer et à implémenter. L'approche consiste à diviser le SMA en sous-systèmes indépendants effectuant chacun une partie du travail. Chaque sous-système est constitué d'un ou plusieurs agents pouvant effectuer une ou plusieurs tâches. Ces agents peuvent avoir un but auxiliaire ou tout simplement attendre des requêtes provenant des autres agents leurs demandant d'effectuer une ou des tâches en particulier.

Parmi les avantages de cette approche, nous pouvons citer :

- Très grande tolérance aux fautes,
- Le partage équitable du travail entre les sous-systèmes et/ou agents,
- L'utilisation plus uniforme des ressources,
- L'indépendance et l'autonomie des sous-systèmes,
- La répartition des tâches,
- L'efficacité du modèle concurrent et/ou parallèle, etc.

4.4.4. L'interaction dans les SMAs

L'interaction est généralement toutes sortes d'échanges qui existent entre les agents eux-mêmes, ou entre l'agent et l'environnement. Les agents peuvent communiquer directement (échange direct des informations ou communication *verbale*), ou indirectement (via l'environnement).

Nous pouvons définir deux formes d'interaction : L'interaction faible et l'interaction forte. Par exemple, on peut dire qu'un individu qui écrit une lettre et qui l'envoie effectue un acte d'interaction faible. En revanche, l'interaction forte couvrira non seulement le fait qu'un

expéditeur envoie une lettre, mais également le fait que le destinataire la reçoive, la lise et en comprenne le contenu.

4.4.4.1. Types d'interaction

a) Coordination

La coordination est une propriété d'un système d'agents qui réalisent des activités dans un environnement partagé. La coordination détermine quelles sont les règles de bon fonctionnement du système auquel les agents appartiennent.

Selon la nature des agents et des interactions, plusieurs formes de coordination sont possibles : la coopération, qui est une coordination entre les agents non antagonistes, et la négociation, qui se trouve entre les agents compétitifs.

b) Coopération

La coopération peut être considérée comme une attitude adoptée par les agents qui décident de travailler ensemble. *Durfee* et ses collègues **[Durfee 89]** ont proposé quatre buts génériques pour établir la coopération dans ce groupe d'agents :

- Augmenter le taux de finalisation des tâches grâce au parallélisme,
- Augmenter le nombre de tâches réalisables grâce au partage de ressources (information, expertise, dispositifs physiques, etc.),
- Augmenter les chances de finaliser des tâches en les dupliquant et en utilisant éventuellement des modes de réalisation différents,
- Diminuer les interférences entre tâches en évitant les interactions négatives.

c) Communication

La communication est un outil d'interaction indispensable pour l'agent quelle que soit sa capacité. La communication peut être définie sur plusieurs niveaux : selon les capacités de l'agent, on parle de communication de haut niveau, et celle de bas niveau.

Pour intéresser les uns les autres, les agents participent dans un dialogue d'une manière active, passive ou les deux, qui les rend maîtres, esclaves, ou pairs.

Plusieurs types de communications existent :

- *Communication par envoi de messages*

Les agents sont en liaison directe et envoient leurs messages directement et explicitement au destinataire. La seule contrainte est la connaissance de l'agent destinataire, comme le montre la Figure 4.4.

Il y a deux types de messages : assertions et requêtes.

Chaque agent, que se soit actif ou passif, doit avoir la capacité d'accepter des informations. Dans le cas le plus simple, cette information sera communiquée par une source extérieure sous forme d'assertion.

Pour assumer un rôle passif dans un dialogue, un agent doit répondre aux questions : accepter des requêtes et répondre à ces requêtes par des assertions. Pour assumer un rôle actif, un agent doit envoyer des requêtes et produire des assertions. Avec ces capacités, il peut contrôler potentiellement des agents en les faisant répondre aux requêtes ou accepter des assertions. Deux agents qui travaillent en pair doivent assumer à la fois les rôles actifs et passifs (produire des requêtes et des assertions).

Les systèmes fondés sur la communication par envoi de messages relèvent d'une distribution totale à la fois de la connaissance, des résultats et des méthodes utilisées pour la résolution du problème.

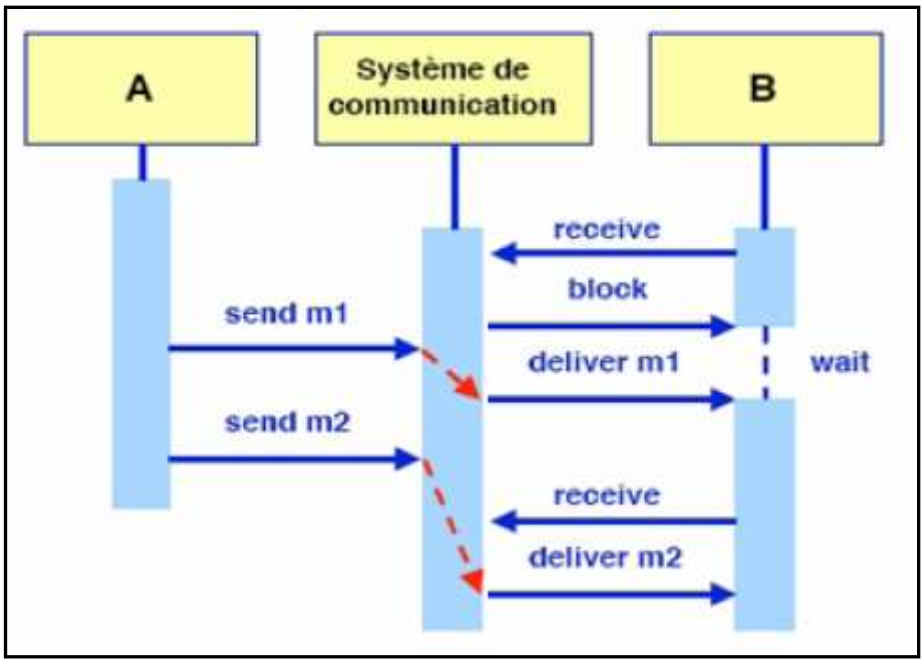

Figure 4.4: Communication par envoi de messages

- *Echange d'informations grâce à un « Blackboard »*

La technique du Blackboard est très utilisée en intelligence artificielle pour spécifier une mémoire partagée par divers systèmes. Dans un SMA utilisant un tableau noir, les agents peuvent écrire des messages, insérer des résultats partiels de leurs calculs et obtenir de l'information (Figure 4.5).

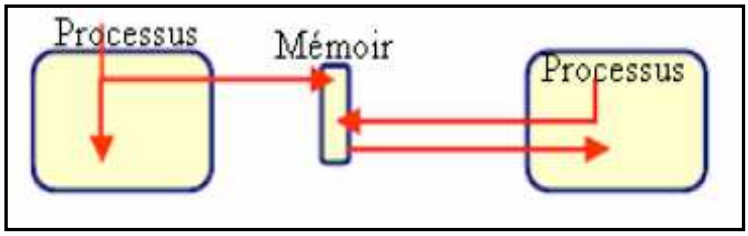

Figure 4.5: Echange d'information "blackboard"[Haiping 03]

- *Actes de discours*

La communication verbale humaine est utilisée comme modèle de communication entre agent. Une théorie de base d'analyse de discours humain est la théorie des actes de discours, qui voie le langage naturel de l'homme comme un ensemble d'actions, comme les requêtes, les suggestions, et les réponses. Par exemple lors qu'on demande quelque chose, on *crée* la demande.

Dans cette approche, on considère que les agents sont dotés de structures de données appelées « états mentaux » à partir desquelles ils peuvent raisonner. Diverses catégories d'états mentaux ont été retenues par les chercheurs dont notamment les croyances, les désirs, les intentions ou buts, les capacités, etc. Ce sont ces mêmes états mentaux qui caractérisent les modèles des agents cognitifs en intelligence artificielle distribuée. Par exemple, lorsqu'un agent X communique avec un autre agent Y, c'est pour influencer les états mentaux de Y. Ainsi, X peut transférer à Y une information qui va changer les croyances d'Y ou bien lui proposer d'adopter un but que Y va transformer en une de ses intentions. Il est donc pertinent de s'intéresser à la façon dont les messages échangés peuvent être structurés pour refléter ces mécanismes d'influence sur les états mentaux des agents.

- *Langage de requêtes et de manipulation de connaissances (KQML)*

C'est un protocole d'échange d'informations et de connaissances. L'élégance de *KQML* est que l'information de compréhension de message est incluse dans la communication.

- *Format d'inter-change de connaissances (KIF)*

La logique symbolique est un outil mathématique qu'on peut utiliser pour la description. Un peu de logique simple (1er ordre) est capable de décrire les faits simples concrets, les définitions, abstractions, inférences, contraintes, et Méta connaissances. KIF, un langage logique particulier, a été proposé comme standard de description dans les systèmes experts, les bases de données et les systèmes multi agents. C'est un langage comprit par la machine et par l'homme.

4.4.5. Conception des SMAs

Les SMAs sont souvent des systèmes complexes qui demandent de longs efforts de développement. Du début des années 80 jusqu'au milieu des années 90, les efforts des chercheurs se sont surtout concentrés sur l'exploration de nouveaux concepts, la mise en place de théories, la réalisation de divers prototypes de SMAs. Cependant, lorsque la technologie atteint plus de maturité et de stabilité, il devient nécessaire de développer des méthodes de conception qui permettent aux entreprises de développer de façon systématique des systèmes opérationnels et des outils pour aider à leur réalisation.

4.4.5.1. Des méthodes

Concevoir des SMA nécessite une méthode. Du fait de la proximité des paradigmes agent et objet, la première idée serait d'utiliser, à quelques adaptations près, les méthodes orientées objet. Mais cela n'est pas possible, parce que :

- Les objets ont une granularité beaucoup trop fine par rapport aux agents ;
- Les interactions entre les objets sont trop rigidement définies (la possibilité d'un agent de refuser ou de négocier une tâche) ;

- Il n'y a pas de mécanisme capable de gérer les structures organisationnelles intrinsèques à la plupart des SMA.

Il est alors nécessaire de créer une méthode orientée agent. Une méthode fondée sur un modèle particulier ou une architecture particulière (modèles d'agent, d'environnement, d'interaction, d'organisation ou d'espace, architectures d'agent ou de communication, etc.).

Les méthodes de conception peuvent être classées en plusieurs groupes, nous pouvons citer :

- Les méthodes utilisant la notation *AUML* **[Bamo99]** : c'est la migration d'*UML* vers *Agent UML*, les méthodes organisationnelles, fondées sur les notions de rôle et d'organisation ;
- Les méthodes formelles, qui utilisent des formalisations mathématiques poussées ;

4.4.6. Les systèmes multi agents pour l'informatique ubiquitaire

L'association des nouvelles technologies de l'information comme les services web et le web sémantique au paradigme des SMA, font de ces derniers une solution bien adaptée et efficace pour concevoir des environnements ubiquitaires intelligents. Ainsi, un environnement ubiquitaire peut être perçu comme un environnement multi agents qui intègre d'une façon transparente des agents intelligents, communicants, et autonomes capables de fournir des service dites sensibles au contexte ; on parle alors d'agents de services sensibles au contexte. Ces services prennent leur pleine signification pour l'assistance aux personnes dans diverses contextes d'utilisation : mobilité (déplacements), maison intelligente et dans différents scénarios d'application : commerce électronique, organisation des voyages, collaboration, divertissement, services, robotiques de services et en particulier la robotique d'assistance, etc.

4.5. OSGi et l'informatique ubiquitaire

4.5.1. Introduction

L'approche à base de services logiciels représente une voie très prometteuse pour la construction d'applications dynamiques. Les services intègrent en effet l'aspect modulaire et autonome des composants logiciels et un ensemble de mécanismes favorisant le dynamisme. Plus précisément, un service peut apparaître et disparaître de façon imprévisible. De ce fait, la liaison avec un service doit pouvoir être réalisée à n'importe quel moment et avec n'importe quel service qui répond aux attentes (fonctionnelles et non fonctionnelles). Cette propriété inhabituelle se révèle très utile pour une importante classe d'applications actuelles, mais soulève aussi de nombreux problèmes.

Parmi les applications pouvant bénéficier de l'approche à services, on trouve toutes les applications ubiquitaires utilisant ou agissant sur des ressources distribuées et fortement évolutives. Pour répondre aux besoins de ces domaines, il faut être capable de traiter l'apparition et la disparition de sources de données sans pour autant arrêter les applications utilisatrices. Pareillement, il faut également être capable de mettre à jour dynamiquement certaines parties des applications pour s'adapter à de nouveaux contextes sans pour autant les stopper.

Open Service Gateway Initiative (OSGI), est un ensemble de spécifications qui définissent un environnement standardisé destiné à faciliter la mise en place de services en réseaux. Ces spécifications concernent les passerelles situées entre un réseau local et l'Internet. Elles

définissent : une architecture logicielle, des services, et la façon dont se déroule l'accès aux dispositifs (Figure 4.7).

OSGi s'est imposé comme le standard de fait pour l'exécution et l'administration des plateformes Java de services opérés et déportés. Cependant, le champ d'application d'OSGi s'est élargi et OSGi pourrait bien être demain la plateforme de référence pour construire des intergiciels pour l'informatique ubiquitaire, des intergiciels dynamiques et flexibles.

Le couplage d'une telle plateforme avec une architecture à base de systèmes multi agents encouragés par les caractéristiques des nouvelles applications ubiquitaires, notamment les applications domotiques, ses avantages sont doubles. D'une part, le modèle de programmation est souple, grâce au découpage d'applications en bundles et en service. Et d'autre part, un outillage avancé est intégré, avec la gestion du déploiement et du cycle de vie des applications.

Avec le développement de la technologie, les équipements domestiques sont de plus en plus numériques et interconnectés en réseau dans la maison. La construction d'un framework pour ces applications est maintenant essentielle. Elle peut nous aider à gérer les dispositifs, à communiquer avec des réseaux externes, notamment à gérer et mettre à jour les services téléchargeables d'un réseau à grande échelle comme Internet.

Les avantages principaux de la plateforme OSGi sont :

- Des services peuvent être dynamiquement chargés sur un réseau à grand secteur (comme Internet) et être accessibles avec une passerelle de services.
- Des services peuvent être consultés par tous les dispositifs reliés par les réseaux dans un environnement ubiquitaire.
- L'installation et la mise à jour des services sont contrôlées de manière dynamique et extensible.
- Des services peuvent être partagés avec les autres services et s'exécutent sur l'environnement d'une ressource limitée comme un dispositif embarqué.

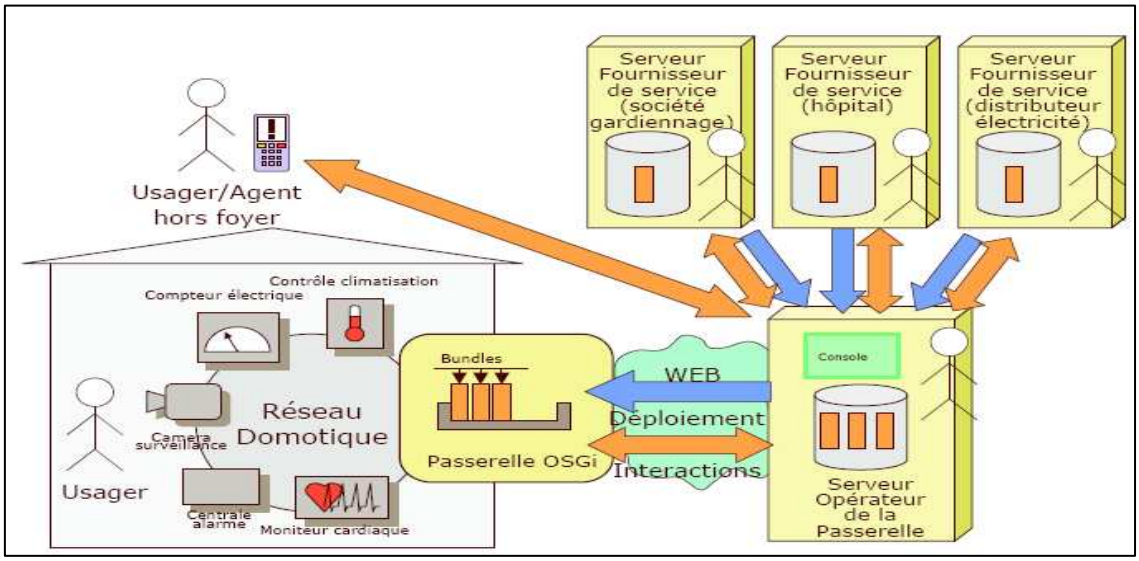

Figure 4.6: Modèle d'administration de passerelles OSGi [OSGi 07]

4.5.2. La plateforme OSGi

La plateforme OSGi est une plateforme d'exécution d'applications Java colocalisées au sein de la même machine virtuelle Java **[Gong 01]**. Chaque application qui est attachée à un fournisseur de services peut éventuellement collaborer avec les applications des autres fournisseurs hébergées sur la plateforme. La plateforme est généralement représentée comme une surcouche au Java Runtime Environnement (Figure 4.7).

Dans ce contexte d'hébergement multi-fournisseur, la spécification OSGi s'est focalisée sur deux points importants : le conditionnement et le déploiement des applications, et la collaboration entre les applications s'exécutant au sein de la même machine virtuelle.

Les applications sont déployées sur la plateforme sous la forme d'unités de conditionnement et de livraison appelées *bundles* (Figure 4.8). Le déploiement avec OSGi couvre l'ensemble des opérations de déploiement **[Carzaniga 98]** c'est-à-dire le chargement et l'installation des unités mais également l'activation, la mise à jour, l'arrêt et la désinstallation. Les opérations de déploiement sont réalisables dynamiquement sans nécessairement redémarrer complètement la plate-forme.

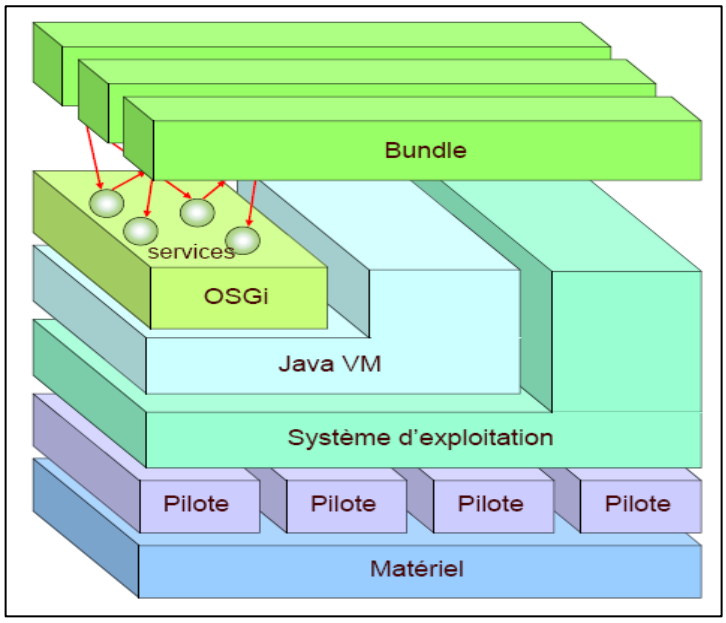

Figure 4.7: Architecture multi-couches d'une plateforme OSGi [OSGi 07]

4.5.2.1. Eléments de la plateforme OSGi

4.5.2.2. Bundle

Le *Bundle* est l'unité de déploiement dans la plateforme OSGi. C'est un fichier *JAR* contenant le code binaire des classes, des ressources comme des fichiers de configuration ou des images. Le fichier de *manifeste* du JAR est augmenté d'un certain nombre d'entrées propres à OSGi. Ces entrées décrivent, entre autres, la classe servant de point d'entrée pour l'activation des services *(Bundle-Activator)*, des informations factuelles, les paquetages importés *(Import-Package)* et les paquetages exportés *(Export-Package)* (Figure 4.9).

manifest.mf

```
Bundle-Name: Robot Bundle Impl 1.0
Bundle-Description: provides a simple OSGi Robot service.
Bundle-UpdateLocation: file:D:\WorkSpace_Eclipse\Robot_bundle.jar
Bundle-Vendor: SEBBAK Faouzi
Bundle-Version: 1.0
Bundle-ContactAddress: Sebbak faouzi (sefaouzi@yahoo.fr)
Bundle-Activator: my_robotpackage.impl.Activator
Export-Package: my_robotpackage
Export-Service: my_robotpackage.RobotMove,my_robotpackage.RobotCamera
```

Figure 4.8: exemple d'un fichier de manifeste du JAR

Le bundle suit le cycle de vie représenté par le diagramme d'état de la Figure 4.9.

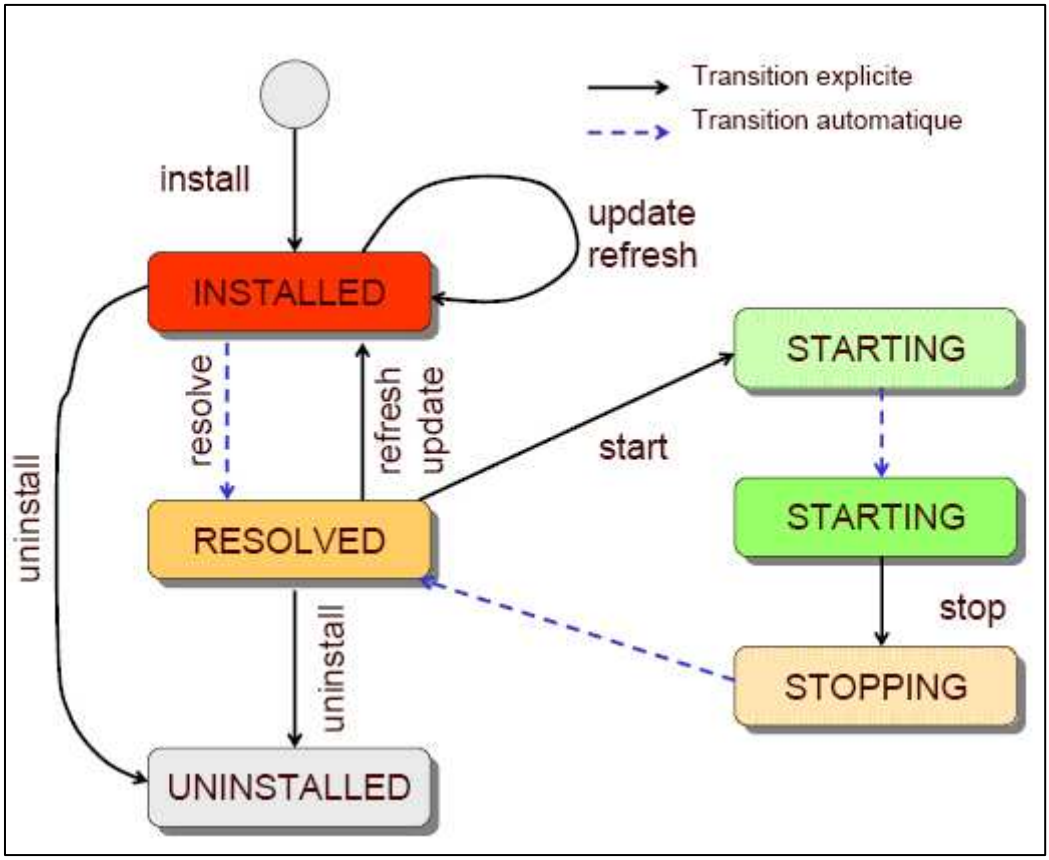

Figure 4.9: Cycle de vie d'un bundle OSGi [OSGi 07]

L'installation consiste à télécharger et stocker le fichier *JAR* du bundle dans le système de fichiers de la plateforme (appelé aussi bundle cache). Le bundle est alors dans son état initial *INSTALLED*.

Par exemple : Install file:c:\my_intergiciel\my_bundle.jar

Install http://localhost/my_application/my_first_bundle.jar

Le bundle passe à l'état *RESOLVED* quand les paquetages qu'il importe (listé dans *Import-Package*), sont tous exportés par des bundles installés et résolus sur la plateforme. Une fois, dans cet état, le *bundle* est prêt pour exporter les paquetages listés dans l'entrée *Export-Package* du manifeste.

L'activation d'un *bundle* devient alors possible. L'activation consiste pour la plateforme à instancier un seul objet de la classe donnée par l'entrée *Bundle-Activator* du manifeste. La classe doit implémenter l'interface *BundleActivator* qui régit le cycle de vie du bundle. La méthode *start(BundleContext)* est alors invoquée avec en paramètre un objet *BundleContext* qui représente à la fois le contexte du bundle et le contexte de la plateforme.

L'arrêt d'un bundle déclenche la méthode *stop(BundleContext)* de l'interface *BundleActivator*. Cette méthode doit dés enregistrer les services fournis, relâché les services utilisés et arrêté les threads démarrés. Le bundle repasse à l'état *résolu*. La mise à jour quand à elle provoque l'arrêt, la réinstallation, la résolution puis la réactivation en continu.

Enfin, la désinstallation du bundle provoque la suppression du JAR du bundle dans le système de fichier local. Cependant, certaines classes restent chargées en mémoire tant que tous les bundles qui en dépendent (c.a.d. qui les importent) sont actifs (c.a.d. dans l'état *ACTIVE*).

4.5.2.3. Service

OSGi suit le paradigme de la programmation orientée services dynamiques **[Bieber 02]**. La programmation orientée services dynamiques considère de plus que les services utiles à un client peuvent apparaître à tout moment et que les services en cours d'utilisation peuvent disparaître à tout moment. Il convient que le client soit sensible à ces changements. Dans la plateforme OSGi, les services sont le moyen pour les bundles de coopérer entre eux.

L'OSGi Alliance a progressivement enrichi les spécifications successives de services standard. Ceux-ci sont généralement optionnels par rapport au noyau OSGi (*core*) qui ne dépend d'aucun d'entre eux.

Parmis ces services : le Http Service pour l'administration à distance via le Web, le Wire Admin Service pour les applications basées sur la collecte de mesures depuis un réseau de capteurs et l'UPnP Device Driver dans le contexte des services à l'habitat.

4.5.3. Modèle conceptuel de l'architecture sémantique basé sur le standard OSGi .Vue d'ensemble

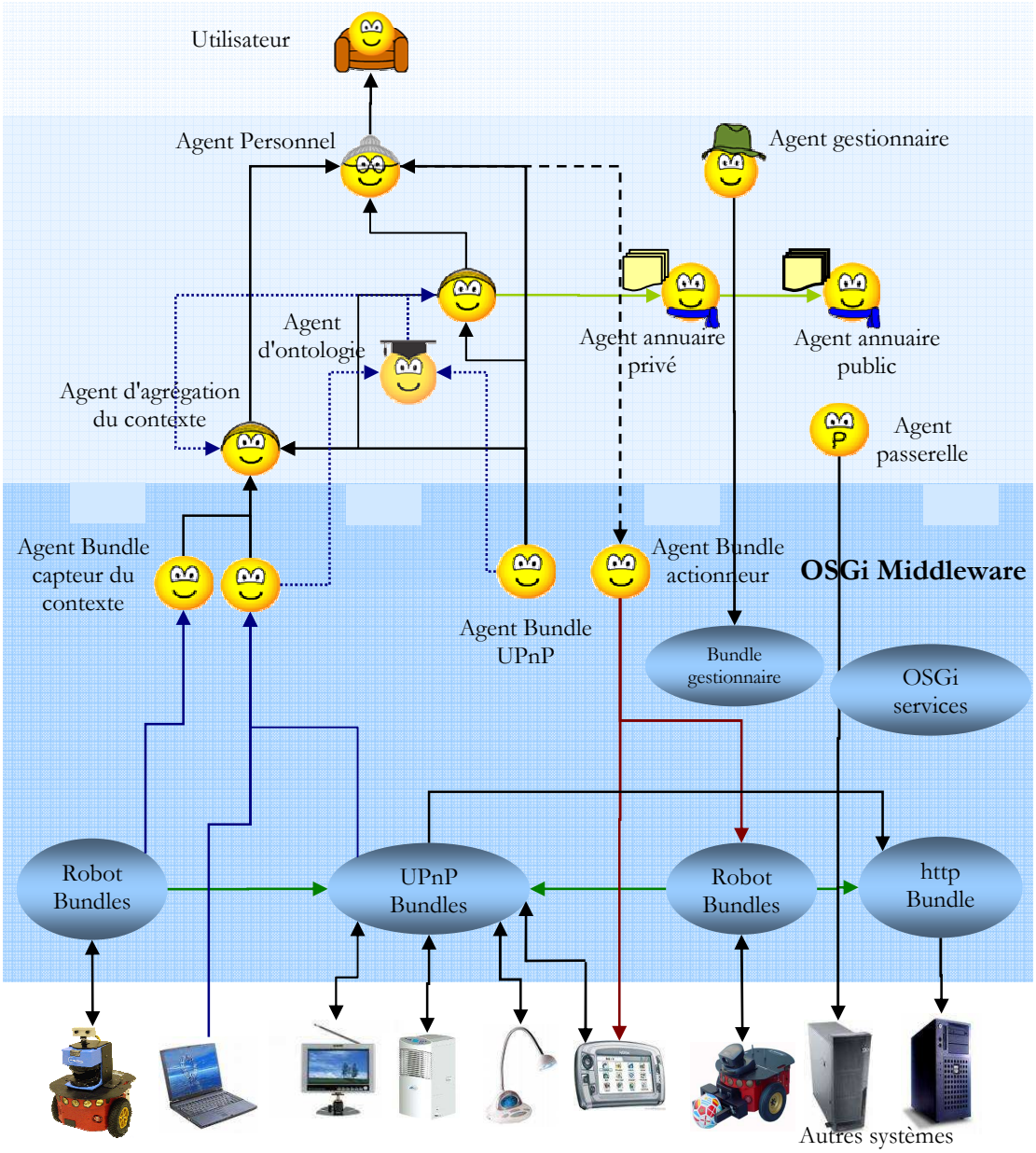

Figure 4.10: architecture sémantique pour la gestion de services robotiques ubiquitaires basée sur le standard OSGi

Dans le modèle de l'intergiciel que nous proposons, nous adoptons l'approche multi agents, décentralisée pour la gestion du contexte proposé par *Chibani* **[Chibani 06]**, afin d'affranchir les contraintes et les problèmes liés à l'utilisation d'un serveur central tels que l'architecture Gaia, SOCAM et MyCampus. Du point de vue des performances, une telle approche permet de s'affranchir de la nécessité de parcourir de la base de données volumineuse du serveur de contexte.

L'intergiciel est composé de deux couches horizontalement superposées, la couche inférieure est la couche OSGi, c'est la couche responsable de l'acquisition du contexte depuis l'environnement ubiquitaire, ainsi que l'application de différents types de contrôle et de commande à travers ces différents bundles, cette couche contient plusieurs unités de bundles, chaque bundle est spécialisé par source de contexte, ce mécanisme de spécialisation permet de partager les tâches liées à l'architecture entre plusieurs entités, de façon à simplifier la conception et le développement d'un tel système.

Un bundle peut être aperçu comme un agent de capture de contexte, dans le cas du bundle robot, ce dernier joue le rôle d'un agent capteur de contexte, puisqu'il donne aux couches supérieures des informations de contexte liées au robot et à l'environnement ubiquitaire tels que la température, la pression, le niveau du gaz CO2 et les positions des victimes dans un environnement de désastre. Ce même bundle offre plusieurs services qui peuvent être utilisés par les agents actionnaires de la couche supérieure, parmi ces services on distingue le service de mouvement du robot et le service du mouvement de la caméra.

La couche qui se situe au-dessus de la couche OSGi forme l'intergiciel multi agents sensibles au contexte; on distingue deux types d'agents de contexte, comme pour le cas des bundles, chaque agent de contexte de l'integiciel est spécialisé par source de contexte, on trouve un agent de température, un agent de mouvement pour robot, etc. L'agent capteur de contexte peut utiliser deux sources différentes de contexte. La première source est la source bundle de la couche inférieure, l'agent peut récupérer directement des informations de contexte à travers l'invocation des services de contexte exporté par ce bundle, et la deuxième source c'est à partir d'un capteur tel un lecteur RFID, une Webcam, etc, et ça dans le cas d'absence de bundle qui exporte ces informations de contexte comme service.

L'agent actionnaire quand a lui, utilise directement les services qu'offrent les bundles de la couche inférieure pour mener ces actions dans l'environnement ubiquitaire.

Cette couche sémantique contient encore, les agents d'agrégation, les agents annuaires, les agents d'ontologie et les agents applicatifs (ou agent assistants personnels) sensibles au contexte. Le comportement de ces derniers est régi par la boucle : perception du contexte, raisonnement et action. Les fonctions de perception du contexte et d'action sur l'environnement ubiquitaire sont assurées respectivement par les agents de collecte du contexte et les agents actionnaires.

Notre intergiciel s'appuie sur les standards des services web pour permettre d'une part, à d'autres systèmes hétérogènes de découvrir et d'accéder aux services des agents et de bundles de l'infrastructure et d'autre part, aux agents de l'infrastructure de découvrir et d'accéder à des services externes disponibles sur le web. Dans ce cas, l'infrastructure joue le rôle de passerelle entre les services et les agents.

4.5.4. Modèle d'interaction agents-bundles

Il existe deux façons différentes pour une interopérabilité entre l'architecture OSGi et les systèmes multi agents à travers la plateforme JADE. La première est que JADE soit construite au-dessus de l'architecture OSGi, et la deuxième est que la plateforme Jade soit enregistrée comme un bundle dans la plateforme OSGi. La première solution semble d'être complexe et très difficile, et nécessite une grande charge de travail, la deuxième semble plus facile à réaliser, et nécessite moins de travail.

Les deux architectures JADE et OSGi peuvent exister dans le même domaine, comme ils peuvent exister sur différents domaines. La Figure 4.11 montre les deux architectures OSGi et JADE co localisé au sein de la même machine virtuelle.

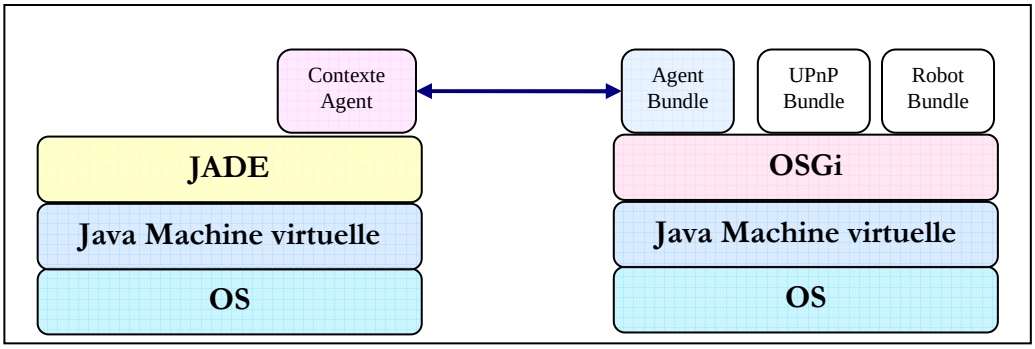

Figure 4.11: Colaboration entre JADE et OSGi [Nam-Ho]

L'architecture sémantique est ainsi devisée en deux parties, une partie OSGi et une partie JADE. Dans la partie JADE, on implémente un agent de contexte, cet agent peut accéder aux services OSGi et obtient ainsi les contextes des services à invoquer.

Dans la partie OSGi, on implémente un bundle (*Agent Bundle*), ce qui représente un agent selon la norme JADE, ce bundle envoie des messages ACL à l'agent de contexte de la plateforme JADE, son rôle est donc est la mise en communication des services JADE avec OSGi, il importe les services disponibles dans JADE, enregistre les services dans le registre des services OSGi, et les invoque quand la communication avec JADE est nécessaire.

Cette proposition **[Nam-Ho]** présente en faite les mêmes inconvénients d'un système centralisé. Si l'agent bundle coté OSGi est hors service, alors l'architecture sera découpée en deux parties qui ne communiquent pas, l'une basée sur des bundles OSGi qui exportent des services non accessibles par les agents de l'autre partie JADE. Le problème est le même si l'agent du contexte coté JADE devient hors service.

Notre alternative de conception d'un tel système à base de couple OSGI et JADE sera d'implémenter une partie des agents de la couche sémantique comme des bundles dans la couche OSGi comme le montre la figure 4.12.

Et notre proposition présente les avantages suivants :

- Les Agents Bundle intégrés dans la plateforme OSGi peuvent utiliser directement les services exportés par les autres bundles sans aucune intervention d'un Agent broker comme dans le cas de [Nam-Ho]. Ces agents utilisent les services proposés par les bundles OSGi, et regroupent les agents capteurs de contexte, les agents d'actionnement, et les agents UPnP.
- Si l'un des agents bundles et accidentellement hors service, ce problème ne mis pas en cause la communication JADE-OSGi et la plateforme contenue à offrir ces services, en substituant ce service par un autre service ou juste le déclarer comme absent.
- C'est une proposition basée sur la décentralisation de la communication, et permettent une meilleure interopérabilité entre les deux plateformes JADE et OSGi.
- En encapsulant les agents JADE dans des bundles OSGi nous permet ainsi de profiter des avantages qu'offre OSGi dans des environnements ubiquitaires hétérogènes et de cacher les

détails d'implémentation de bas niveau. Un agent peut donc être démarré, arrêté, et mis à jour comme un simple bundle OSGi.

Les détails d'implémentation des agents JADE comme des bundles dans la plateforme OSGi seront détaillés dans le chapitre de mise en œuvre et validation de l'intergiciel.

4.5.5. Eléments de la couche OSGi

4.5.5.1. Les composants d'un réseau UPnP

Les éléments de base pour pouvoir travailler avec un réseau UPnP sont les équipements *(devices)*, les services, et les points de contrôle *(Control Points, CP)*. Un équipement (qu'on peut également appeler *équipement contrôlé*) se comporte comme un serveur. Il répond aux requêtes des points de contrôle. Plusieurs équipements et points de contrôle peuvent être présents et opérationnels simultanément sur un même réseau. Un équipement UPnP est un élément qui contient des services et éventuellement d'autres équipements imbriqués. Par exemple, un simple magnétoscope est un équipement, mais un combiné TV/magnétoscope est un équipement qui a deux équipements imbriqués (la TV et le magnétoscope).

Un service est la plus petite unité de contrôle dans un réseau UPnP. Il propose des actions et son état est modélisé grâce à ses variables d'état.

Un service se compose de :

- Une table d'état : Elle modélise l'état du service par des variables d'état et les met à jour quand l'état change.
- Un serveur de commande : Il reçoit des demandes d'action (telle que *switch_on_light*), les exécute, met à jour la table d'état et renvoie les réponses.
- Un serveur d'événements : Il envoie des évènements aux abonnés intéressés lorsque l'état du service change. Par exemple, un service de signal d'incendie enverrait un événement aux abonnés intéressés quand son état devient "alerte".

Et finalement, un point de contrôle est un contrôleur capable de découvrir et de contrôler des équipements, et d'utiliser leurs services.

Plusieurs équipements ont été standardisés par les membres du Forum UPnP, chaque catégorie d'équipements standardisés possède un « *Standardized DCP* » qui définit le DCP, ou *Device Control Protocol*. Le DCP impose la manière de décrire un équipement et ce qu'il doit offrir comme services, ceci pour garantir l'interopérabilité d'équipements provenant de constructeurs différents.

Le Forum UPnP a défini un formalisme pour décrire en XML chaque équipement et chaque service. En plus de la description XML qui normalise les échanges entre machines, il est tout aussi important de normaliser la façon de présenter aux développeurs humains les informations sur les équipements et les services. Il existe pour ça deux fichiers templates, l'un pour un équipement et l'autre pour un service qui explique comment rédiger un DCP.

4.5.5.2. Les Bundles UPnP

On peut imaginer une solution ou un bundle dans la plateforme OSGi joue le même rôle qu'un équipement dans un réseau UPnP, dans la mesure où un bundle OSGi peut jouer le rôle

d'un point de contrôle, et un service OSGi peut être exporté sur le réseau UPnP en tant qu'équipement. Enfin, les événements UPnP peuvent être reçus et transmis entre bundles, équipement et points de contrôles.

Chaque équipement UPnP est représenté et géré par un seul bundle, On trouve également un bundle pour la télévision, un autre pour la machine à laver, et d'autres, pour la climatisation, l'éclairage, la température, etc. Ce bundle peut utiliser les services fournis par l'http bundle pour exporter des services accessibles via le web.

4.5.5.3. Les Bundles pour Robot

Ces bundles encapsulent tous les services qu'il peut offrir un robot de services dans un environnement ubiquitaire en général et dans une maison intelligente en particulier, parmi ces services on trouve les services de base de déplacement, de capture d'information de contexte, de détection de la présence d'une personne et enfin les services d'actionnement sur l'environnement.

Le robot utilise plusieurs capteurs embarqués pour collecter les informations contextuelles à partir de l'environnement, ces informations sont immédiatement accessibles aux agents de la couche supérieure pour être utilisée ensuite par la couche sémantique. Ces bundles robots utilisent des services fournis par d'autres bundles de la même couche OSGi comme le UPnP bundles et peut à son tour offrir des services de contrôle des équipements de l'environnement, Les services proposés par ces bundles sont accessibles via l'interface web, puisque ces bundles utilisent les fonctions web exporté par l'http bundle.

Ces bundles jouent le rôle d'un ensemble d'actionnaires dans l'environnement, et exportent plusieurs services d'actions, qui peuvent être actionnés par les agents de la couche sémantique.

4.5.5.4. Le Bundle HTTP

Le bundle HTTP permet de créer facilement une interface web à partir de fichiers contenus dans les autres bundles, de servlets et même d'applets. Cet accès web permet d'offrir une interface web pour contrôler l'ensemble des bundles ou des équipements présent dans l'environnement ubiquitaire.

4.5.5.5. Le Bundle Gestionnaire

Ce bundle est un bundle système qui gère l'enregistrement, le désenregistrement, l'arrivée et le départ des services, et donne ainsi une vue à jour de la plateforme OSGi pour le compte de l'agent gestionnaire de la plateforme JADE. Il gère aussi le cycle de vie des différents bundles installés de la couche OSGi.

En effet, après l'enregistrement d'un service OSGi, le bundle Gestionnaire informe l'agent gestionnaire de l'arrivée d'un nouveau service, ce dernier mis à jour sa base de services, et le service est ainsi accessible aux autres agents de la plateforme JADE.

4.5.6. Modèle comportemental des agents de l'infrastructure intergicielle

On distingue trois types d'agents dans l'infrastructure intergicielle : les agents capteurs, les agents d'agrégation et les agents actionneurs.

4.5.6.1. Agent capteur

L'agent capteur du contexte fournit aux autres agents des connaissances contextuelles acquises à partir des sources de données du contexte ou à partir des services de contexte fourni par les bundles de la couche OSGi. Il offre ainsi une abstraction des données contextuelles brutes et hétérogènes en masquant les détails d'implémentation des capteurs (physiques ou logiciels) associés aux sources du contexte. Les bundles OSGi peuvent à leurs tours cacher ces détails d'implémentation, et n'offrent aux agents de l'infrastructure qu'une abstraction des données à travers des interfaces de services.
En général, les données contextuelles sont acquises dans une forme brute et par conséquent non exploitables directement dans leurs formats d'origine par les autres agents, puisque ces derniers doivent implémenter les codecs correspondants à ces formats.

Pour garantir une plus grande facilité de gestion des connaissances contextuelles, chaque agent capteur de contexte est spécialisé par un seul type de connaissances contextuelles.

Chaque agent capteur du contexte a deux comportements indépendants exécutés en parallèle. Le premier concerne la tâche d'acquisition et de traitement des données, et le deuxième concerne quand à lui la gestion des requêtes d'interrogation émanant des autres agents.

4.5.6.2. Agent d'agrégation

L'agent d'agrégation du contexte a pour rôle d'agréger ou de fusionner plusieurs connaissances contextuelles en une seule, et la mettre accessible aux autres agents de contextes. Ces connaissances contextuelles peuvent être obtenus à partir des autres agents de contexte ou à partir des services exportés par les bundles OSGi. Un agent d'agrégation se distingue d'un agent capteur de contexte par son indépendance vis-à-vis des sources de contexte. Ainsi quand un agent fournisseur de contexte devient indisponible, il est possible de le substituer par un autre agent ou par un service OSGi sans suspendre le comportement de l'agent d'agrégation. De plus, si l'agent d'agrégation de contexte utilise un service d'un bundle OSGi dans son schéma d'agrégation et ce service n'est plus disponible, alors il peut le substituer soit par un autre service OSGi ou par un agent de contexte.

Le traitement des connaissances contextuelles s'appuie sur un modèle d'agrégation du contexte, qui correspond à un schéma de composition de plusieurs agents de contexte.

4.5.6.3. Agents actionneurs

L'agent actionneur offre aux autres agents applicatifs des services permettant l'exécution d'actions. Cet agent offre une couche d'abstraction qui permet de cacher l'hétérogénéité et les détails d'implémentation des actionneurs disponibles dans l'environnement. Par exemple, si un agent applicatif désire déplacer le robot de service vers une victime dans un environnement de désastre, il peut invoquer le service d'un agent actionneur, responsable du déplacement du robot, sans se soucier de l'implémentation de ce dernier, cet agent applicatif peut également invoquer directement le service *MoveTo* du bundle OSGi du robot. Un agent actionneur possède un comportement simple, qui consiste à enregistrer son service auprès d'un agent annuaire puis à répondre aux invocations des autres agents. La réponse à une invocation consiste à traduire le contenu du message d'invocation en une action sur l'environnement ubiquitaire.

4.5.6.4. Les agents annuaires

L'architecture de l'intergiciel est une architecture à base d'agents d'annuaires répartis, qui offrent aux agents de l'infrastructure deux services essentiels : la publication et la découverte des services. La publication des descriptions de services consiste à ajouter ou à supprimer de l'annuaire du contexte la description sémantique d'un service et l'identifiant de l'agent de service correspondant. La découverte de services permet à l'agent annuaire d'informer un agent demandeur sur la disponibilité d'un ou de plusieurs services.

Nous considérons deux types d'agents annuaires : des agents d'annuaires utilisateurs ou privés et des agents d'annuaires publics.

• **Agent d'annuaire utilisateur (privé) :** Un agent d'annuaire utilisateur permet de gérer les services des agents de contexte qui appartiennent au domaine de l'utilisateur. Puisqu'on associe à chaque utilisateur un seul agent annuaire. Lorsqu'un agent d'annuaire privé est instancié, il diffuse dans les annuaires publics un message de présence comportant des informations à caractère public permettant aux autres agents annuaires de l'identifier.

• **Agent d'annuaire public :** La structure d'un annuaire public comprend d'une part, les références des annuaires publics existants et d'autre part, les annonces de services appartenant au même domaine que l'agent annuaire. Dès qu'un agent d'annuaire public est déployé sur une plateforme multi agents, il diffuse un message de présence vers les annuaires publics existants. Ensuite, il se met en attente des requêtes d'annonces des services et des messages de présence des autres agents annuaires publics ou privés qui sont déployés dans l'environnement **[Chibani 06]**.

4.5.6.5. Agent gestionnaire de l'infrastructure

L'agent gestionnaire a pour rôle de gérer le cycle de vie des agents de services déployés dans l'infrastructure intergicielle. Le comportement de l'agent gestionnaire diffère des agents d'annuaires par sa nature proactive. Il vérifie régulièrement la disponibilité des agents de services et prend des mesures correctives (arrêt, suspension, réinitialisation des agents, etc.) lorsqu'il détecte des anomalies. Cet agent est en contact permanant avec le bundle gestionnaire de la couche OSGi, et il maintient une vue cohérente de l'état de l'architecture globale en termes d'absence et de présence de services et d'agents.

4.5.6.6. Agent d'ontologie

L'agent d'ontologie gère le partage des connaissances entre les différents agents de l'infrastructure. Il offre pour cela deux services : le service de publication des ontologies et le service de vérification sémantique des connaissances contextuelles produites par les agents de contexte. Le partage des ontologies est mis en oeuvre à travers un serveur web, qui est accessible par les agents de l'infrastructure.

4.5.6.7. Agent passerelle

Le rôle de l'agent passerelle se décline sous la forme de deux fonctionnalités : permettre à des systèmes hétérogènes d'invoquer les services de l'infrastructure ou inversement permettre à des agents de l'infrastructure d'invoquer des services web de système hétérogènes. Pour se faire, cet agent est constitué d'un module interne qui traduit les messages échangés avec les agents de contexte, conformément au protocole SOAP. Par ailleurs, il convertit les interfaces de services

des agents de contexte en descriptions de services web conformes au langage WSDL, pour la publication de ces services dans des annuaires de services web.

4.5.6.8. Les agents personnels sensibles au contexte

Les agents personnels sont considérés comme des agents proactifs, dans le sens où ils ont une aptitude à planifier leurs actions, à négocier avec d'autres agents et à acquérir ou à modifier leurs connaissances. Plusieurs types d'agents personnels ont été développés dans des projets tels que l'agent personnel d'assistance à l'interaction avec les applications, l'agent personnel d'organisation automatique de rendez-vous, l'agent personnel d'informations sur le web et l'agent personnel du commerce électronique.

L'agent personnel permet d'identifier l'utilisateur à partir de son profile, de son agenda et de son répertoire de contacts, de comprendre ses besoins et de l'assister dans ses activités quotidiennes, en lui permettant de disposer de plus de temps pour l'essentiel de son travail.

L'agent assistant personnel doit être doté d'un mécanisme d'adaptation au contexte de l'utilisateur, l'idée est d'aller au-delà de la notion d'identification de l'utilisateur par son profile, son agenda et son répertoire, en permettant à l'agent de service de tenir compte de tout le contexte dans lequel évolue l'utilisateur.
L'architecture d'un agent personnel sensible au contexte, illustrée dans la figure 4.14, est composée des éléments suivants:

- Module de raisonnement : Il permet à l'agent d'inférer de nouvelles connaissances contextuelles et d'adapter ses décisions en fonction de ces connaissances.
- Base de connaissances : Cette base comprend les connaissances contextuelles et les règles d'inférence qui permettent d'enrichir ces connaissances et d'émettre des décisions.
- Modèle du contexte : Il comprend les ontologies du contexte et les différentes ontologies spécifiques au métier d'application.
- Module d'interaction : Il permet à l'agent personnel d'interagir avec les utilisateurs, les agents de l'infrastructure intergicielle et des systèmes hétérogènes.
- Module de découverte des services de contexte : Il permet d'une part, la découverte des services de contexte et d'autre part, la découverte des services actionneurs permettant d'agir sur l'environnement.

Le comportement de l'agent personnel est organisé autour d'une boucle de type perception du (contexte-décision-action). La perception correspond à la découverte et l'invocation des services de contexte, la décision correspond à l'interprétation des connaissances contextuelles acquises et l'action au résultat de l'interprétation de ces connaissances.

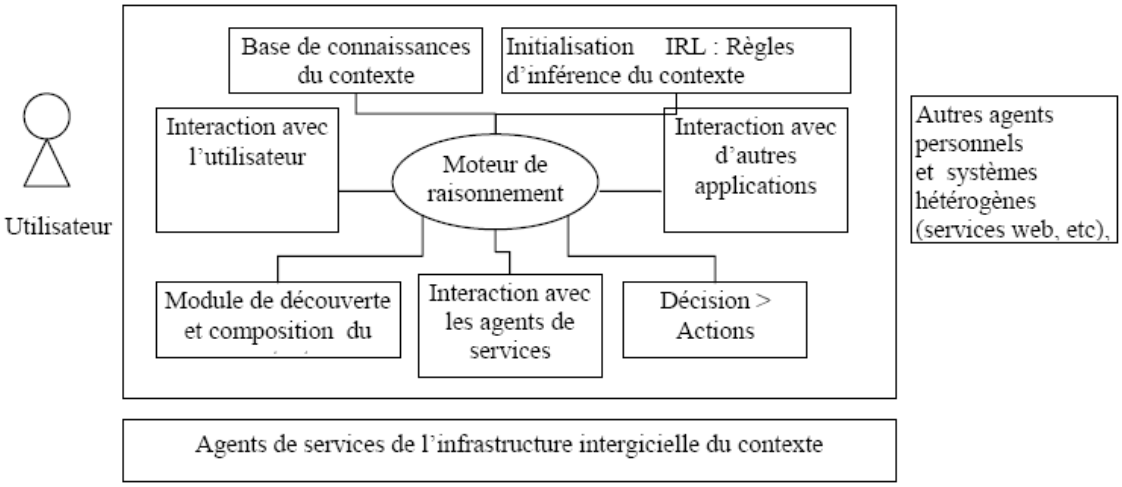

Figure 4.12: Architecture interne d'un agent personnel sensible au contexte [Chibani 06]

4.6. Synthèse

Nous avons montré dans ce chapitre les avantages du paradigme multi agents pour la conception d'un intergiciel sensible au contexte. L'architecture que nous proposons se distingue des approches existantes, par une décentralisation complète de la gestion du contexte partagé entre une couche OSGi est une couche sémantique à base de système multi agents.

Dans le modèle de l'intergiciel, nous adoptons l'approche multi agents, décentralisée pour la gestion du contexte proposé par *Chibani*, et afin d'affranchir les contraintes et les problèmes liés à la nature dynamique d'un environnement ubiquitaire, nous avons ajouté une couche de services OSGi au-dessous de l'intergiciel de Chibani et augmenter ainsi la robustesse et la flexibilité de l'intergiciel. Cette plateforme OSGi présente pour un environnement ubiquitaire beaucoup d'avantages. Premièrement, les services peuvent être dynamiquement chargés et être accessibles avec une passerelle de services, de plus, ces services peuvent être consultés par tous les dispositifs reliés par réseaux dans un environnement ubiquitaire, et l'installation et la mise à jour de ces services sont contrôlées de manière dynamique et extensible. Enfin, ces services peuvent être partagés avec d'autres services et peuvent s'exécuter sur l'environnement d'une ressource limitée comme un dispositif embarqué.

La gestion du contexte repose sur les agents de contexte qui utilisent les informations contextuelles provenant des services de capture de contexte qu'offrent les bundles de la couche OSGi. Les agents d'agrégation du contexte ont pour rôle, quant à eux, de fournir des connaissances contextuelles inférées à partir de connaissances fournies par d'autres agents de contexte et services OSGi. Un agent d'agrégation se distingue d'un agent capteur par son indépendance vis-à-vis des sources de contexte. Conjointement aux agents de contexte, les agents actionneurs offrent une couche d'abstraction qui permet de masquer l'hétérogénéité et les détails d'implémentation des actionneurs disponibles dans l'environnement.

Chapitre 5 :

Mise en œuvre et validation de l'intergiciel

5.1. Introduction

Nous avons présenté dans le chapitre précédent le modèle conceptuel de l'intergiciel sémantique basé sur le standard OSGi. Cette architecture est composée d'une couche inférieure basée sur le standard OSGi, et d'autres couches supérieures qui forment un système multi agents. L'architecture proposée bénéficie ainsi des différents avantages qu'offre le standard OSGi dans des environnements ubiquitaires dont la principale caractéristique est la nature dynamique de l'environnement.

Dans ce chapitre, nous présentons la mise en œuvre de cette architecture intergicielle basée sur le standard OSGi. Dans la première partie du chapitre, nous présentons les outils logiciels sur lesquels nous nous sommes appuyés pour cette mise en œuvre. La deuxième partie sera consacrée à l'étude d'un scénario pour la validation du modèle proposé, ce scénario s'appuie sur le développement d'un ensemble de services robotiques OSGi et un ensemble d'agents sémantiques. Le couplage de ces services OSGi avec les agents de la couche sémantique permet à un robot mobile d'explorer un lieu de désastre qui est l'objet de la simulation.

5.2. Modèle d'implémentation de l'intergiciel

Pour le modèle d'implémentation de la couche OSGi, notre choix s'est porté sur un ensemble de technologies existantes. Il s'agit essentiellement de l'environnement OSCAR pour le développement des services robotiques, de l'environnement JADE (*Java Agent Development Framework*) pour le développement du système multi agents en général et des Bundles Agents en particulier, du moteur de raisonnement JESS utilisé conjointement avec les comportements réactifs propres à JADE pour implémenter tous ce qui est comportement délibératif de l'agent, de l'éditeur Protégé pour la construction de l'ontologies OWL, du module beangenerator pour générer l'ontologie pour jade à partir du modèle OWL, du moteur d'inférence en logique de description Racer **[Haarslev 00]** pour tester la validation de l'ontologie proposée. Enfin, le simulateur *USARSim* pour la validation et la simulation des services développés.

OSCAR est développé par l'équipe Adele qui est une implémentation ouverte (open source) du standard OSGi, qui s'est imposé comme la version libre de référence. JADE est un framework logiciel pour les systèmes multi agents entièrement gratuit, implémenté en Java et développé par Italia Telecom. Quand au USARSim, il s'agit d'un simulateur de haute fidélité à trois dimensions développé par l'Institut National des standards (NIST National Institute of Standards).

Les trois environnements de développement seront détaillés dans les sections suivantes.

5.2.1. OSCAR

OSCAR est une implémentation open source du standard OSGi, démarrée en décembre 2000 par l'équipe *ADELE* du laboratoire LSR de l'université de *Joseph Fourier* juste après la publication de la première spécification d'OSGi. Depuis le jour, OSCAR a été constamment en

développement, ce qui a lui permet d'adopter toutes les spécifications du standard OSGi y compris la dernière spécification (la spécification 4). Les fonctionnalités du framework OSGi qu'offre Oscar sont très stables et en cours d'utilisation par un large communauté d'utilisateurs, en plus, suit à son caractère flexible, Oscar peut être facilement intégré dans divers projets ou être ajouté comme une extension.

Oscar et OSGi en général, contenus de jouer un rôle très important dans différents domaines où le caractère dynamique est présent notamment dans des environnements ubiquitaires hétérogènes.

5.2.2. JADE

Le meilleur moyen pour construire un système multi agents (SMA) est d'utiliser une plateforme multi agents. Une plateforme multi agents est un ensemble d'outils nécessaires à la construction et à la mise en service d'agents au sein d'un environnement spécifique. Ces outils peuvent servir également à l'analyse et au test du SMA ainsi créé. Ces outils peuvent être sous la forme d'environnement de programmation (API) et d'applications permettant d'aider le développeur. L'un des outils les plus utilisés dans ce domaine est la plateforme JADE.

JADE est une plateforme crée par le laboratoire *TILAB* **[TILAB]** permettant la construction des systèmes multi agents. Cette plateforme est entièrement implémenté en JAVA, et répond aux spécifications FIPA (*Foundation for Intelligent Physical Agents*) dont l'objectif est de produire des standards pour l'interopération d'agents logiciels hétérogènes. Ainsi, la plateforme JADE fournit un grand nombre de classes qui implémentent le comportement des agents qu'elle crée. Elle possède trois modules principaux (nécessaire aux normes *FIPA* **[FIPA 01]**) (Figure 5.1).

- DF « *Director Facilitor* » fournit un service de « *pages jaunes* » à la plateforme ;
- ACC « *Agent Communication Channel* » gère la communication entre les agents ;
- 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 plateforme.

Cette plateforme multi agents peut être distribuée sur chaque machine (et ce même si elle ne dispose pas nécessairement du même système d'exploitation) et les configurations peuvent être modifiées au démarrage des agents en les déplaçant d'une machine à une autre, ce qui permet une très grande portabilité des agents.

Le choix de la plateforme JADE a été motivé par les raisons suivantes :

- JADE est un environnement logiciel libre, s'appuyant sur le langage Java. Il a pour but de faciliter le développement de systèmes multi agents, conformément au standard FIPA.
- JADE est un environnement intégré qui comporte une bibliothèque de classes Java pour le développement des SMA et une interface graphique pour le déboguage, l'exécution et l'administration des agents.
- La version JADE-LEAP (*Jade Light Weight Agent Plateforme*) a été développée pour être supportée par plusieurs dispositifs mobiles (PDA, Téléphones mobiles, etc.). Autre caractéristique, JADE peut être distribué sur plusieurs plateformes matérielles, qui n'ont pas nécessairement le même système d'exploitation **[Berger 02]**.

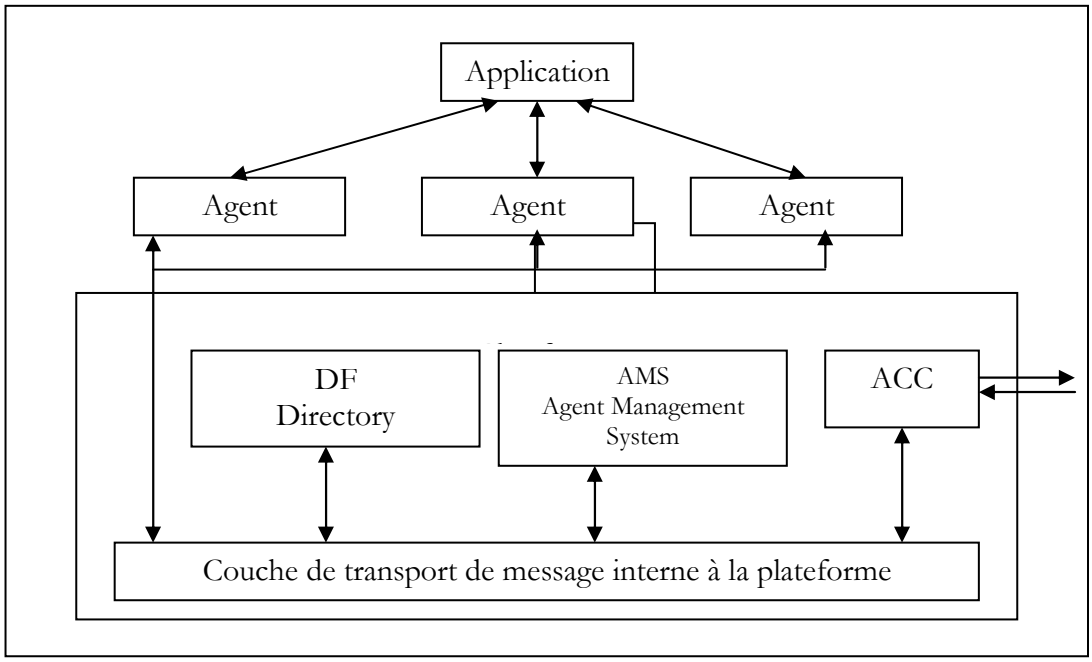

Figure 5.1: Modèle de référence d'une plateforme agent FIPA [FIPA 01]

5.2.3. USARSim

USARSim est un simulateur de haute fidélité à trois dimensions développé par l'Institut National des Standards, dont l'objectif est d'être utilisé dans le domaine de la recherche pour le sauvetage dans un milieu urbain (*USAR Urban Search And Rescue*) pour limiter les dommages causés par les catastrophes naturelles par la fourniture de l'aide à la décision dans des situations urgentes. Il est aussi utilisé dans le domaine de recherche des interactions humain-robots (*HRI*), et dans le domaine de la recherche pour la coordination entre une population de robots **[USARSim]**.

L'institut National des Standards *NIST* fournit des environnements standards qui peuvent être utilisés dans différents scénarios de simulation, parmi ces environnements, trois environnements de désastre ont été développés, le jaune, l'orange et le rouge, ces environnement se distingue par le degré de difficulté affronté par un robot mobile lors d'une opération d'exploration.

USARSim a été construit au-dessus du moteur du jeu *Unreal Tournment*, et profite ainsi de la prise en charge de la plupart des aspects difficiles de la simulation par ce dernier, et il se base sur la modélisation des robots, des systèmes de contrôle, des capteurs, des interfaces, des outils et des environnements. Ces tâches à leurs tours ont été accélérées par d'autres outils intégrés aux seins du moteur du jeu permettant de modéliser une variété de plateformes dans moins de temps.

La version actuelle d'USARSim consiste en différents modèles environmentales, différents modèles des robots expérimentaux et commerciaux et différents modèles de capteurs. L'architecture de ce simulateur est détaillée dans la section suivante :

5.2.3.1. Architecture du simulateur

Le simulateur USARSim utilise l'architecture client/serveur et se divise ainsi en deux parties client et serveur (Figure 5.2). Au-dessus de l'icône réseaux (*Network*) se trouve la partie client,

cette partie contient le Unreal client et le contrôleur ou l'application de l'utilisateur. La partie Unreal client représente une vue de l'environnement de la simulation qui peut être attachée à un spectateur ou à la caméra du robot. La partie serveur (*Unreal server*) quand à elle, contient le Unreal engine qui est le noyau du moteur du jeu *Unreal Tournment*, le *Gamebots*, différentes cartes d'environnements de simulation, et un ensemble de modèles de robots, de victimes et de capteurs, etc.). Le serveur maintient un état cohérant de tous les objets de la simulation, et répond aux requêtes des clients et aux contrôleurs des utilisateurs.

a) Le Unreal engine

Le *Unreal engine* utilisé dans le simulateur USARSim est développé par le groupe *Epic Games* comme un noyau de jeu de combat *Unreal Tournment 2004*. Ce moteur de jeu offre un graphisme 3D et une haute qualité de réalité avec l'adoption d'un moteur physique (the Karma engine). Il offre ainsi un langage de script sous le nom de "*Unreal Script*" ce qui permet aux développeurs des jeux de développer leurs propres jeux. Avec ce langage de script, les développeurs peuvent créer leurs objets toute en contrôlant leurs comportements.
Le *Uneal Editor* est un éditeur 3D intégré avec le moteur Unreal engine, et qui aide les développeurs à la construction de leurs propres cartes, formes géométriques et terrains.

b) Le Gamebots

Le protocole utilisé dans le moteur *Unreal engine* est un protocole propriétaire. Ce qui engendre une difficulté lors de la communication de l'Unreal Tournment avec d'autres applications. Par conséquent, le Gamebots a été construit par les chercheurs dont l'objectif est de faire un pont de communication entre le Unreal engine et les applications extérieures. Il se base sur une communication socket via le protocole TCP/IP. Ainsi, USARSim permet au Gamebots de communiquer avec les contrôleurs développés par les utilisateurs dans un environnement de simulation.

c) Les contrôleurs

Dans la partie client du simulateur USARSim nous trouvons l'élément le plus important pour notre travail, ainsi que pour d'autres projets de recherche en robotique, comme la coopération et la coordination entre un groupe de robots, l'interaction homme-machine et les services robotiques dans un environnement ubiquitaire.

D'une manière générale, le contrôleur doit en premier lieu se connecter au moteur de jeu "Unreal server", et ensuite envoyer une commande à l'USARSim pour ajouter un robot dans la carte de simulation. Après la création du robot dans le simulateur, le contrôleur écoutera les données des capteurs embarqués sur le robot et envoyera des éventuelles commandes pour le contrôler. L'architecture client/serveur du simulateur permet l'ajout de plusieurs robots dans une simulation, et par conséquent, chaque contrôleur doit créer sa propre connexion.

Plusieurs contrôleurs ont été développés par une variété de développeurs tels que : *The Mobility Open Architecture Simulation and Tools* (*MOAST*), *Pyro* et *Player* (voir le chapitre I), dont le but est d'aider les développeurs dans leurs efforts par la fourniture de plusieurs frameworks, outils et interfaces.

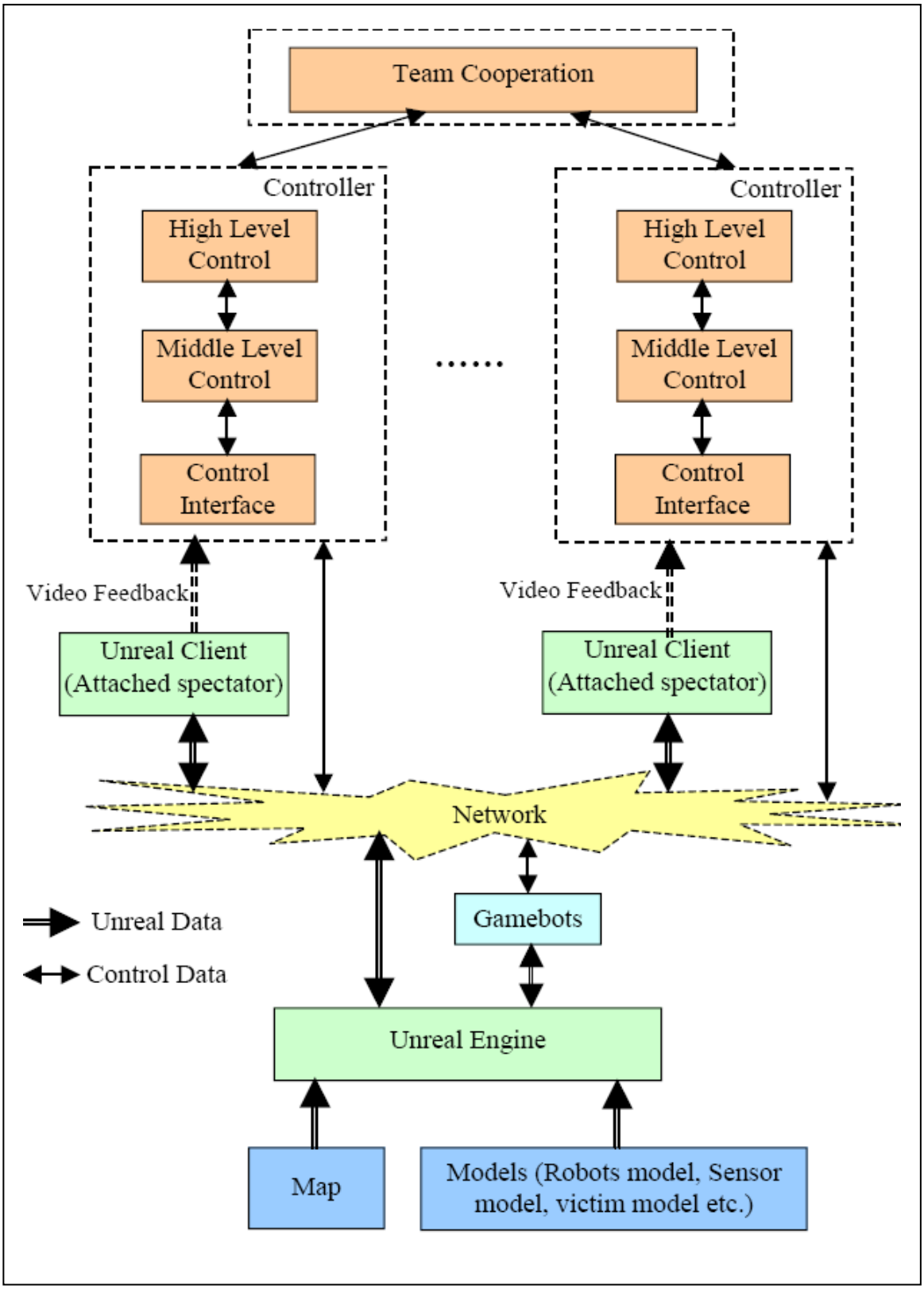

Figure 5.2: Architecture du simulateur USARSim [USARSim]

5.2.3.2. Éléments du simulateur

Les éléments de base du simulateur sont : l'environnement de la simulation, les robots et l'ensemble de leurs capteurs et actionnaires. Ces trois éléments seront déraillés dans les sections suivantes.

a) L'environnement de la simulation

L'environnement joue un rôle très important dans la simulation. Il définit le contexte de la simulation, et sans lui la simulation n'a aucun sens. Plusieurs environnements de simulation ont été distribués pour être utilisés dans USARSim, et les développeurs sont totalement libres en ce qui concerne le contexte d'utilisation de ces environnements.

USARSim est basé sur des environnements de désastres au sein du domaine de la recherche et sauvetage urbaines (*USAR*), autour de cet axe de recherche, le NIST a construit trois environnements qui permettent aux chercheurs d'évaluer les performances de leurs robots.

Parmi ces environnements, trois ont été largement utilisés dans la recherche dans le domaine de la robotique : l'environnement jaune (yellow arena), l'environnement orange (orange arena) et l'environnement rouge (red arena).

La différence entre ces trois environnements réside dans la difficulté rencontrée lors de la navigation par un robot semi ou complètement autonome. Dans l'environnement jaune (Figure 3), qui est l'environnement le plus simple, avec des obstacles comme des murs perpendiculaires, le robot, aura une difficulté de navigation nettement inférieure par rapport à une navigation dans un environnement rouge où l'environnement est composé de deux étages avec plusieurs obstacles physiques comme les escaliers et les ponts, et couvert par plusieurs débris comme des papiers et des blocs cendreront.

Figure 5.3: L'environnement jaune de la simulation [USARSim]

b) Simulation des capteurs et actionnaires

Les capteurs jouent un rôle très important dans le processus de simulation, et représentent l'élément de base pour le contrôle des robots à travers la vérification de l'état de quelques objets ou par une série de calculs dans le Unreal engin.

Dans USARSim trois types de capteurs ont été simulés :

- Les capteurs d'état de robot tel que l'état de la batterie.
- Les capteurs d'estimation de la position tels que les capteurs de position, et les capteurs de vitesse.
- Les capteurs de perception de l'environnement comme les capteurs sonars, les capteurs laser, les caméras, les lecteurs des tags RFID.

Les capteurs configurables que présente USARSim peuvent être facilement montés sur des robots en ajoutant des lignes de code dans leurs fichiers de configuration.

Les actionnaires sont très similaires aux capteurs et peuvent être montés sur un robot, mais au lieu de capter des données de l'environnement, les actionnaires acceptent les commandes des utilisateurs et exécutent les fonctions correspondantes dans l'environnement de la simulation.

c) Simulation des robots

Avec l'utilisation du Krama engine intégré dans le jeu Unreal Tournment 2004, différents types de corps rigides peuvent être simulés et par conséquent, plusieurs robots ont été créés.

Ces robots virtuels simulent tous les comportements des robots réels. Le modèle architectural d'un robot simulé comporte une chassie, une caméra et d'autres objets auxiliaires, connectés à travers des jointures simulés (Figure 5.4)

USARSim fournit 8 différents robots : *P2AT*, *P2DX*, *ATRV-Jr*, *Zerg*, *Tarantula*, *Talon*, *Telemax*, et *Soryu*. En plus, USARSim contient quatre véhicules pour des simulations à l'extérieur (*outdoor*) et fournit quelques algorithmes de test de navigation : *Hummer*, *sedan*, *SnowStorm*, et *Cooper*.

D'autres robots existent encore tels que le *QRIO*, *ERS*, le *Submarine* et le *Helicopter* qui peuvent être utilisés chaqu'un un dans son environnement spécifique.

Figure 5.4: Robot ATRV-Jr (réel / simulé)

5.3. Les services robotiques développés

L'intergiciel proposé dans le chapitre précédent se base sur une couche OSGi, dans laquelle différents types de services ont été implémentés. Ces services permettent aux couches supérieures de l'intergiciel d'acquérir et d'inférer différentes informations de contexte qui peuvent être traitées et analysées par la suite. Ces informations de contexte jouent un rôle très important dans la mesure où les services proposés par l'intergiciel sont sensibles au moindre changement de ce contexte.

Les services robotiques quand à eux, offrent aux utilisateurs applicatifs en général et aux couches sémantiques en particulier des informations sur le contexte de l'environnement tels que la température, le niveau du gaz CO2, la pression, la présence des personnes et des victimes (dans un milieu de désastre). En même temps, ces services offrent des interfaces d'action qui peuvent agir sur le contexte de l'environnement.

Les services robotiques comportent deux grandes classes (Figure 5.5). Les services de base tels que le service de déplacement du robot, le service de déplacement de la caméra, le service du contexte brut, le service d'état et le service de localisation du robot. Les services optionnels quand à eux comportent par exemple les services de loisirs dans une maison intelligente, et les services nécessaires pour explorer un environnement de désastre.

Les services de base sont obligatoire pour tout type de robot, dans le contexte ou un robot immobile ne présente aucun intérêt dans un environnement ubiquitaire. L'absence d'un service de base comme le service de déplacement du robot fait de ce dernier un ensemble de capteurs.

Les services optionnels changent d'un environnement à un autre. Ainsi, les services qu'offre un robot de service dans une maison intelligente ne sont pas forcement les mêmes services qu'offre un robot qui explore un environnement de désastre. Dans le premier cas, les services se résument dans les services de loisirs, d'aide et de contrôle, tandis que pour le second cas, la détection des corps et des victimes, le niveau de gaz du CO2, et la température sont des services obligatoire dans ce genre de situation.

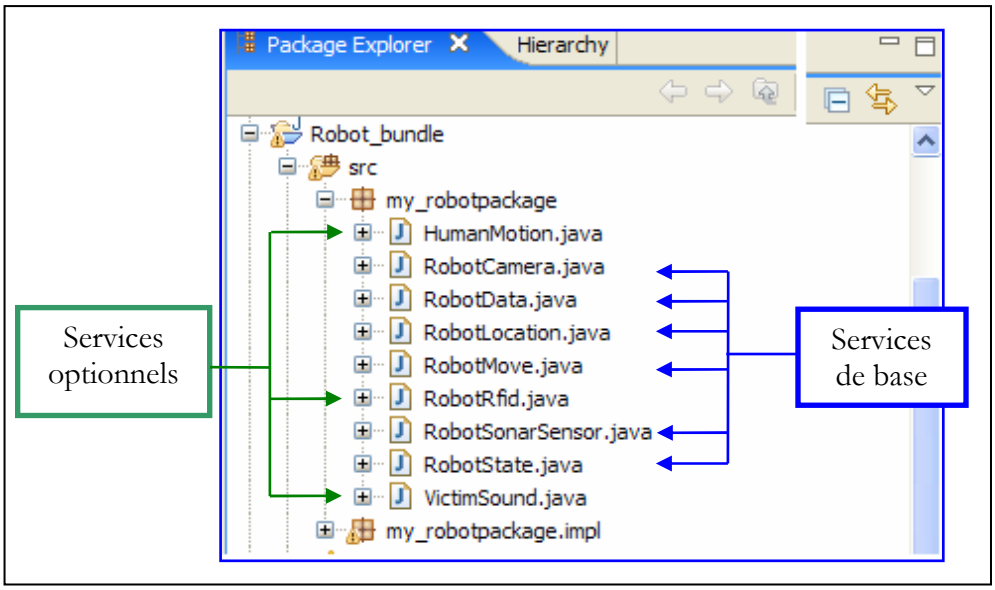

Figure 5.5: Les services robotiques

5.3.1.1. Les services robotiques de base

Dans le présent travail, les services robotiques de base comportent un service de déplacement du robot, un service de déplacement de la caméra, un service de contexte brut, un service de détection d'obstacles, un service de localisation et un service d'état du robot (voir l'*annexe A* pour les détails de chaque service).

a) Le service de déplacement du robot

Le service de déplacement du robot est le service de base dans cet ensemble, à l'aide de ce service, le robot peut offrir ces services tout en déplaçant dans son environnement et offrant d'autres services optionnels. Ce service contient les primitives de base pour la navigation d'un robot dans un environnement donné, les cinq fonctionnalités de base qu'offre ce service sont: le déplacement en avant, le déplacement en arrière, le tournement à gauche et à droite, et le déplacement vers une victime.

b) Le service du mouvement de la camera

Le service du mouvement de la caméra permet quand à lui d'orienter la caméra du robot vers d'autres orientations que celle de l'origine en offrant des primitives de contrôle qui agissent sur l'orientation de la caméra, et par conséquent un agent applicatif peut récupérer plusieurs aperçus sur l'environnement. Le mouvement de cette caméra du robot se résume dans quatre actions : tourner la caméra vers le haut, vers le bas, vers la droite et vers la gauche.

c) Le service de localisation du robot

Le service de localisation du robot fournit l'emplacement exact du robot dans son environnement. Ce service contient une seule fonctionnalité qui retourne les coordonnées et l'orientation du robot. Ces informations peuvent être utilisées par la suite par d'autres services ou par la couche sémantique pour mener d'autres actions dans l'environnement en tenant compte de l'emplacement actuel du robot.

Dans une maison intelligente, et lors de détection d'une chute libre d'une personne à travers un accéléromètre, un agent (agent applicatif) de la couche sémantique doit savoir la localisation du robot avant tous éventuels déplacements, cet agent peut ensuite déplacer le robot vers la personne tout en invoquant une série de services. Parmi ces services, le service de déplacement du robot joue le rôle le plus important, ensuite le service de détection et d'évitement d'obstacles, et enfin le service du mouvement de la caméra afin de récupérer le flux vidéo et l'envoyer vers le système de surveillance de l'hôpital concerné. Donc la localisation du robot dans cette maison et la brique de base dans un tel et d'autres scénarios de services robotiques.

d) Le service du contexte brut

Dans le simulateur USARSim un robot peut être aperçu comme une agrégation de plusieurs capteurs et actionnaires dont l'objectif est de capter les données et de mener des actions dans un environnement donné. Tout comme le cas d'un robot réel, USARSim envoie chaque 20 millisecondes « durée configurable » toutes les données brutes captées par l'ensemble des capteurs du robot. Ces données sont traitées par le service du contexte brut (RobotData) et rendues disponible pour l'ensemble des services de base et des services optionnels à travers un ensemble de fonctionnalité.

La plupart des services de base et optionnels doivent s'enregistrer auprès de ce service pour récupérer les données des capteurs concernés. Ainsi, le service du mouvement du robot par exemple doit invoquer les fonctions suivantes du service du contexte brut pour récupérer les données du capteur INS (*Inertial Navigation Sensor*) :

- Récupérer la position x du robot (get_Current_posx_From_Ins)
- Récupérer la position y du robot (get_Current_posy_From_Ins)
- Récupérer la position z du robot (get_Current_posz_From_Ins)

Les fonctionnalités qu'offre ce service pour chacun des services de base et optionnels sont détaillées dans l'annexe A.

e) Le service de détection d'obstacles

Un robot de service fournit ses services en explorant un environnement inconnu à l'origine. Cet environnement présente différents types d'obstacles (des murs, des victimes, des robots, et d'autres objets). En fournissant ces services le robot doit contourner les obstacles tout en réalisant les buts d'origines.

Le service de détection d'obstacle utilise les données reçues de la part des capteurs ultrason du robot pour fournir deux primitives de navigation. La première donne un ensemble de valeurs qui correspond aux distances des capteurs par rapport aux obstacles, et la deuxième donne la distance d'un obstacle par rapport à un capteur donné parmi les seize capteurs embarqués sur le robot.

f) Le service d'état du robot

Enfin, et lors de la fourniture de ces services dans un environnement donné, le robot peut accidentellement devenir indisponible, et par conséquent certains de ces services ne seront plus accessibles.

Beaucoup de facteurs entrent en jeu dans ce genre de situation, et la batterie du robot est souvent la source du problème, c'est pourquoi l'état du robot en général et de la batterie en particulier doit être vérifié durant la fourniture des services par le robot.

Dans cet objectif, le service de l'état du robot a été développé autour de quelques fonctionnalités qui vérifient l'état du robot en se basant sur l'état de la batterie et l'état de la lumière émise par le robot.

5.3.1.2. Les services robotiques optionnels

Un robot de service évolue dans un environnement aménagé par l'homme et donc a priori structuré. Toutefois, cet environnement n'est pas conçu de manière à faciliter les tâches du robot. Il est variable et évolutif et nécessite de la part du robot beaucoup de services optionnels pour une autonomie décisionnelle et un fonctionnement robuste. Ces services optionnels changent d'un environnement à un autre et d'une situation à une autre.

Les services robotiques deviennent largement utilisés dans une multitude de disciplines, et les robots doivent en conséquence fournir des services appropriés pour chacun de ces domaines.

Dans une maison intelligente, et lors de la fourniture de ces services, un robot doit être capable de localiser un utilisateur grâce à sa caméra, d'obtenir son profile grâce au RFID, d'avoir

la température d'une pièce et la température de l'eau, d'ouvrir et fermer les volets et les fenêtres, d'allumer et éteindre le système d'air conditionnée et d'allumer et éteindre la lumière et la télévision…etc. Contrairement aux services fournis dans une maison intelligente, un robot de service doit être capable de détecter la présence d'une victime dans un environnement de désastre, de capter le niveau du gaz CO2 et d'identifier les victimes qui présentent un signe de vie.

Dans le présent projet, quelques services optionnels ont été développés pour être utilisé par un robot de services qui explore un environnement de désastre. La tâche principale de ce robot est de localiser et d'identifier les victimes dans cet environnement, ces services ainsi que les précédents services de bases seront utilisés dans un scénario pour valider notre proposition intergicielle.

a) Le service de détection des mouvements des victimes

Dans un environnement de désastre, les victimes d'une explosion ou d'un incendie peuvent donner des signes de vie à travers des mouvements de quelques parties du corps, et par conséquent un robot de service doit être en mesure de les détecter et de les localiser.

Ce service contient deux primitives, la première donne la probabilité que l'objet qui présente des mouvements est une victime, tandis que la deuxième donne la position de cet objet. À travers ces deux primitives, le robot peut décider que le présent objet est une victime ou peut invoquer d'autres services comme la détection des victimes par la voix, et dans les deux cas, un flux vidéo est envoyé au superviseur du système pour une ultérieure décision.

b) Le service de détection de la voix des victimes

Certain des victimes de l'environnement précédent peuvent donner des signes de secours à travers la voix, et par conséquent notre robot de service doit les détecter, et de les localiser. Ce service utilise un capteur de son embarqué sur le robot et présente trois primitives qui donnent : le niveau du son émis par la victime, la durée du son, et la localisation de cette victime. L'erreur qu'engendre ce service par une fausse détection de victime peut être détectée par une confirmation à travers le flux vidéo envoyé par la caméra du robot.

c) Le service de détection des victimes à travers les RFID

La radio-identification, venant de l'anglais *radio frequency identification*, est une méthode pour stocker et récupérer des données à distance en utilisant des marqueurs appelés radio-étiquettes (*RFID tag* en anglais) **[Wikipedia]**. Les radio-étiquettes sont de petits objets qui peuvenre collés ou incorporés dans des produits. Les radio-étiquettes comprennent une antenne associée à une puce électronique qui leur permettent de recevoir et de répondre aux requêtes radio émises depuis l'émetteur-récepteur.

Ce système d'identification est largement utilisé dans le contrôle d'accès des bâtiments sensibles, dans les clés électroniques de certains modèles d'automobiles et dans les badges mains-libres, etc. Combinés avec des capteurs sensibles aux fonctions principales du corps humain, ce système peut être adopté comme une solution de supervision de l'état de santé d'un patient.

Dans le présent travail, le robot est équipé avec un lecteur RFID qui lui permet d'identifier un RFID tag, de lire son mémoire et de l'écrire à travers le service de détection de victimes à base des RFID tag développé.

Ce service comporte deux primitives qui donnent le nombre des tags RFID détectées dans l'environnement et les identificateurs de tous les RFID présent dans le champ du lecteur

RFID du robot, et deux primitives pour la lecture et l'écriture dans ces RFID, ainsi qu'une dernière primitive pour la localisation de ces tags RFID.

5.4. Interopérabilité JADE-OSGi

L'architecture intergicielle proposée repose sur un couplage faible entre les deux plateformes JADE et OSGi, parmi les agents de la plateforme JADE, quelques-uns ont été implémentés comme des bundles dans la plateforme OSGi et profitant ainsi d'une *meilleure interopérabilité* JADE-OSGi.

Parmi les agents implémentés dans la plateforme OSGi, on distingue les agents d'acquisition du contexte, et les agents d'actionnement sur l'environnement. Ce choix et motivé par l'exploitation directe des services exportés par le Bundle Robot et les Bundles UPnP.

Pour implémenter ces agents dans la plateforme OSGi, nous avant développés deux bundles : un *Bundle Agent Loader* et un *Bundle Agent Factory* (Figure 5.6).

Le Bundle Agent Loader démarre la plateforme JADE (création d'un conteneur principal) et exporte le service "*deployAgent*" qui permet le déploiement d'agent par la suite. Si ce bundle détecte qu'il y a déjà une plateforme JADE active, c.à.d, que la création d'un *MainContainer* a échoué, il s'enregistre auprès de cette plateforme toute en créant un nouveau conteneur d'agent (AgentContainer).

Le Bundle Agent Factory quand à lui, utilise le service exporté par le Bundle Agent Loader (*deployAgent*) et exporte les services "*AgentFactory*" et *"GenericAgentFactory"* qui permettent la création des agents à l'intérieur d'une plateforme OSGi par l'invocation de méthodes de création d'agent "createAgent".
Ainsi pour chaque agent qu'on veut créer :

- On crée un bundle qui contient le code source de l'agent ;
- On enregistre un service "AgentFactory" avec la propriété suivante :"agent-type={the agent class name}" ;
- Quand la méthode "createAgent ()" est appelée, le service retourne une nouvelle instance d'un agent après les trois étapes suivantes :
  - Chercher le service "*AgentFactory*" avec la propriété "*agent-type= the agent class name*".
  - appeler la méthode "*createAgent()*" sur ce factory "*AgentFactory*"
  - Charger l'agent dans le conteneur.

L'avantage est que si nous voulons qu'un agent crée un autre agent, il suffit d'utiliser le service *GenericAgentFactory* à l'intérieur d'OSGi, car on ne peut pas demander à AMS de créer un agent parce qu'il ne trouvera pas la classe de l'agent.
Ces deux bundles, avec les services qu'offrent, présentent les avantages suivants:

- Les agents peuvent à présent utiliser les services OSGi, et donc présenter une implémentation et une validation de la couche inférieure de notre proposition.
- On peut démarrer des agents à tout instant à partir d'autres agents.
- Au cours de développement des agents on peut :
  - modifier le code de l'agent,

- recompiler, et mettre à jour le bundle correspondant dans la plateforme OSGi,
- recharger l'agent dans JADE, sans redémarrer aucune plateforme.

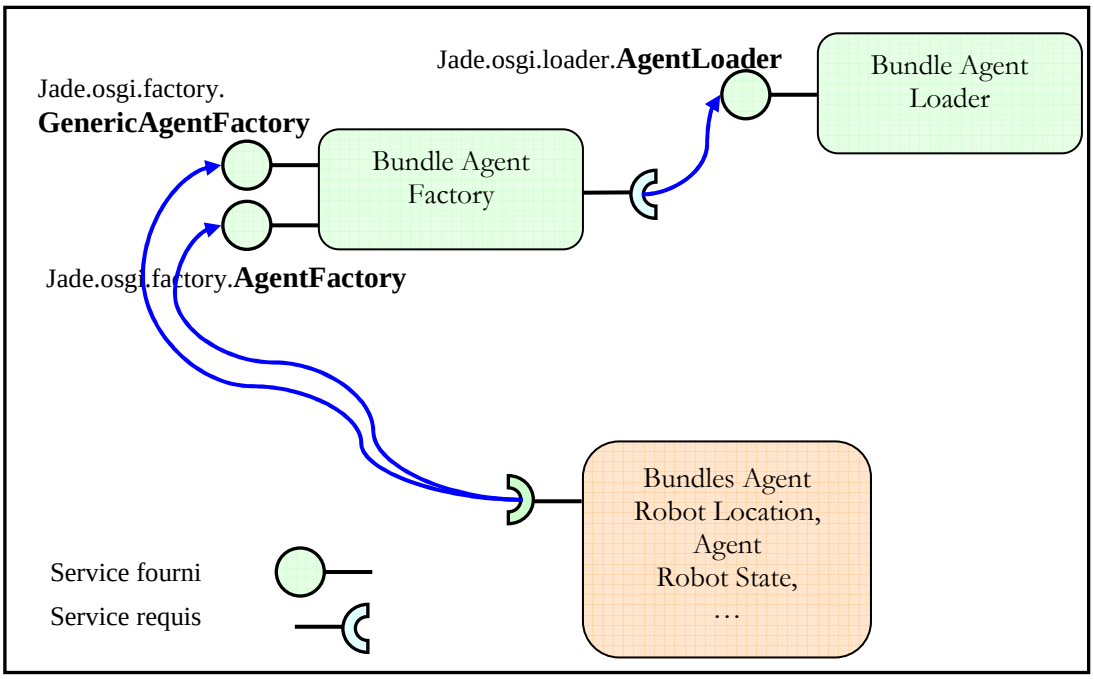

Figure 5.6: Bundle Agent Loader et Bundle Agent Factory

5.5. Modèle de validation de l'intergiciel

Pour mettre en œuvre notre proposition intergicielle sémantique basé sur le standard OSGi, des services robotiques de base et optionnels ont été précédemment détaillés.

Autres que ces services OSGi, quelques agents JADE ont été implémentés comme des bundles OSGi, et d'autres simplement en tant qu'agent JADE. Parmi ces bundles agents on distingue, le bundle agent d'état du robot, le bundle agent de localisation du robot, le bundle agent de localisation des victimes à travers les tags RFID, le bundle agent de localisation des victimes à travers la voix, le bundle agent de localisation des victimes à travers la détection des mouvements de ces derniers, et le bundle agent du mouvement du robot.
L'agent utilisateur et l'agent de détection des victimes sont des agents JADE (non implémentés dans OSGi) (Figure 5.7).

Les trois agents : "*Agent Sound Victim Location*", "*Agent Human Motion*", et "*Agent Rfid Victim Location*" sont des agents de recherche de victimes dans l'environnement en détectant respectivement la voix émise par les victimes, les mouvements des victimes et en lisant les tags RFID embarqués sur les victimes.
Les autres agents servent à donner l'état du robot à l'agent utilisateur.

Ces services OSGi et ces agents JADE ont été utilisés dans un scénario de recherche des victimes dans un environnement de désastre en utilisant le simulateur USARSim.

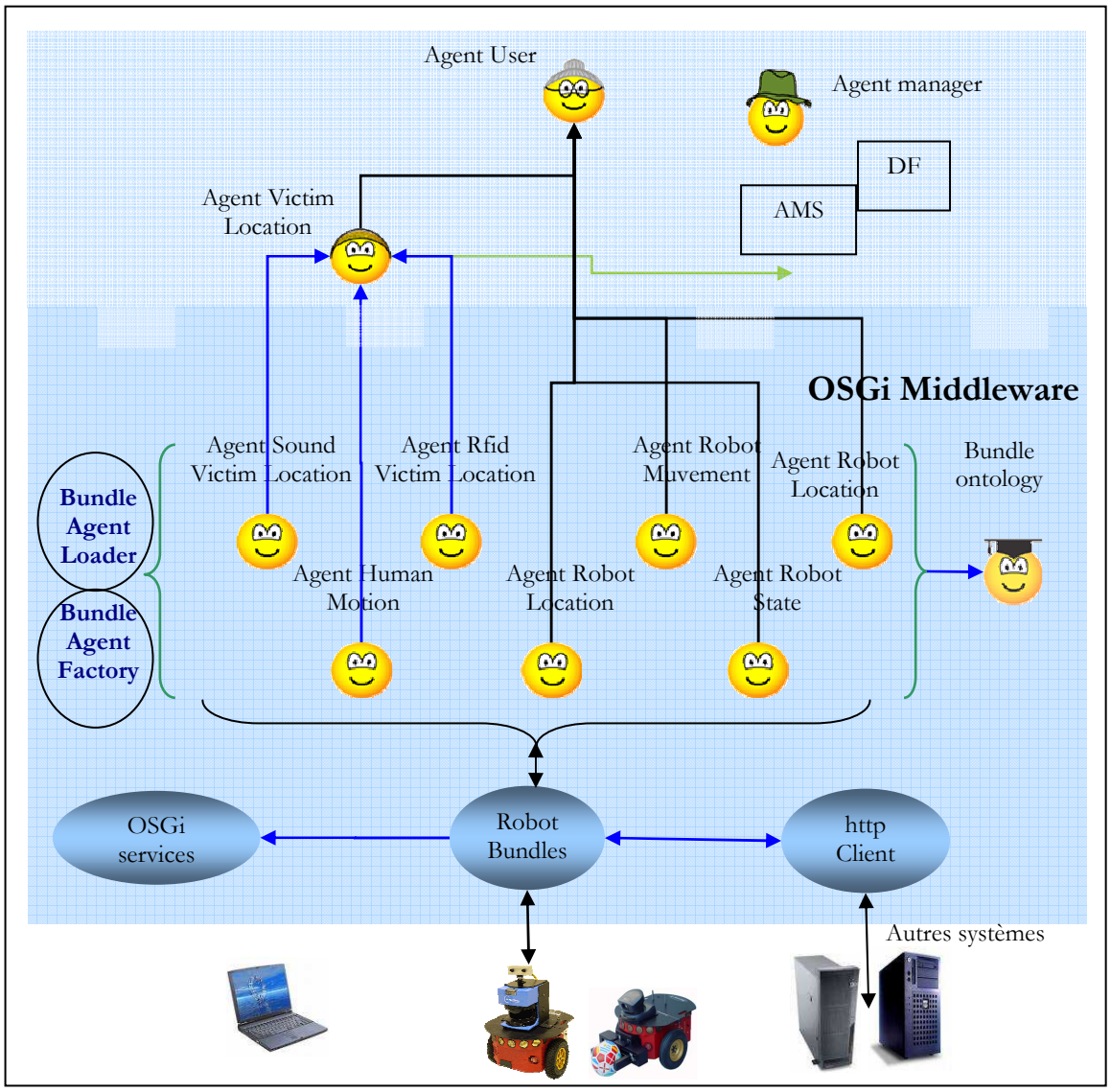

Figure 5.7: Mise en oeuvre de l'intergiciel sémantique basé sur le standard OSGi

Chaque agent installé comme un bundle dans la plateforme OSGi utilise les services qu'offre le bundle Robot. Dans ce qui suit, nous étudions en détail le comportement de l'agent de localisation du robot ainsi que l'agent utilisateur. Le comportement des autres agents bundles est similaire à celui de l'agent de localisation du robot.

Le comportement de l'agent de localisation du robot est illustré par le diagramme d'état de la Figure 5.8. On distingue deux comportements indépendants qui correspondent à deux (*Agent Behaviours* en termes de programmation). Le premier concerne la tâche d'invocation et de réception des données à partir du service OSGi (*Robot Location*) du bundle Robot. L'exécution de cette tâche est déclenchée juste après l'instanciation de l'agent. Ce dernier vérifie ensuite la disponibilité des données contextuelles en assurant la disponibilité du service OSGi *(RobotLocation)*. Si le service est indisponible, l'agent se met en attente jusqu'à ce que ce dernier devient disponible. Ensuite il envoie une annonce à l'agent annuaire pour notifier sa présence dans l'environnement et enregistre sa description. Une fois cette étape terminée, l'agent procède à

l'invocation du service OSGi puis à la représentation de ces données sous forme de connaissances de haut niveau, conformément au modèle sémantique du contexte (Figure 5.9).

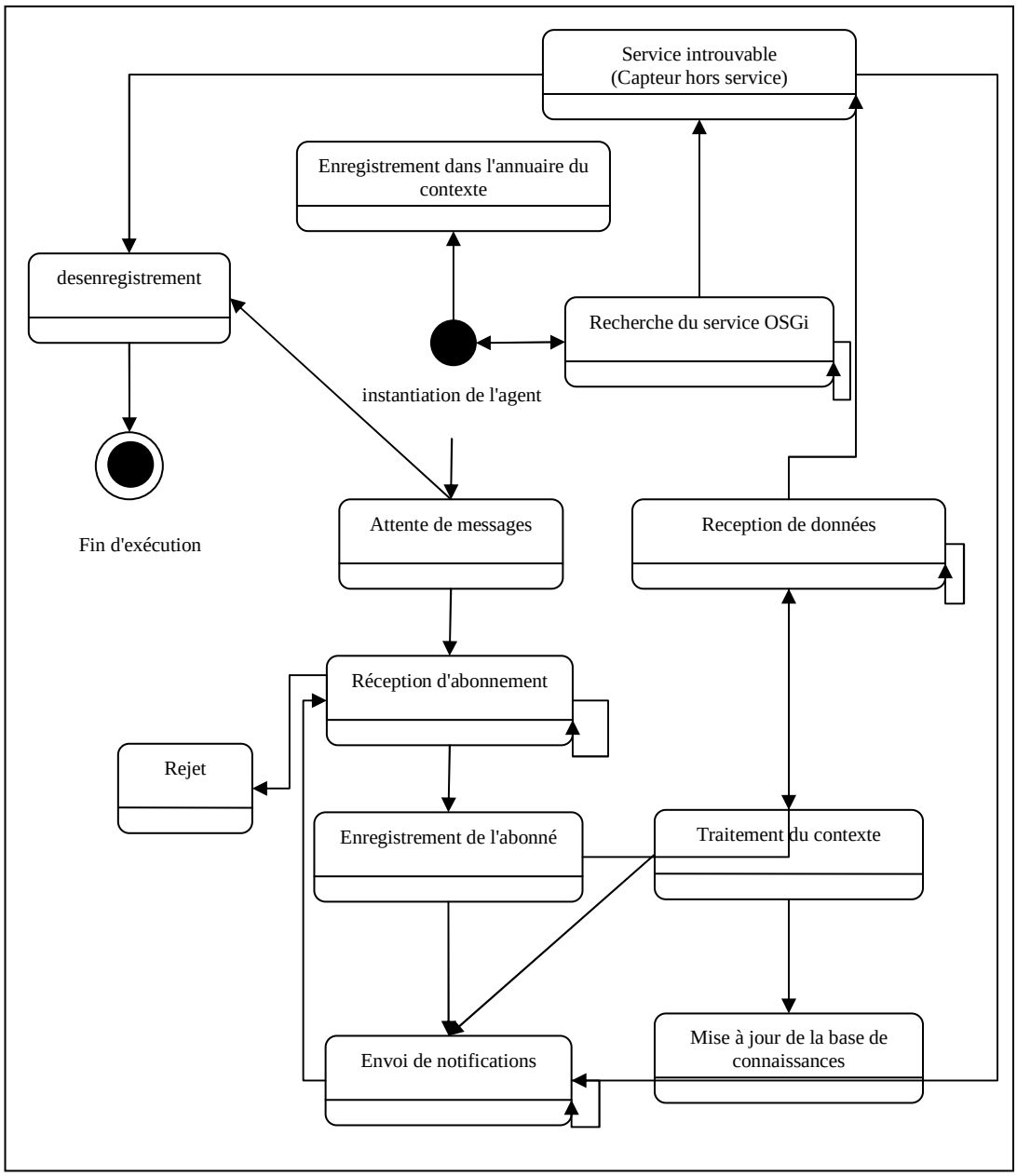

Figure 5.8: diagramme d'état de l'agent de localisation du robot

Parallèlement à l'invocation du service RobotLocation d'OSGi, l'agent de localisation du robot gère les requêtes d'interrogation émanant des autres agents. L'agent de localisation commence à traiter les requêtes des autres agents juste après l'enregistrement de sa description dans l'annuaire.

```
(INFORM
:sender  ( agent-identifier :name agent_Robot_Location@IBM:1099/JADE
    :addresses (sequence http://IBM:7778/acc http://IBM:3698/acc ))
:receiver  (set ( agent-identifier :name agent_User@IBM:1099/JADE
    :addresses (sequence http://IBM:7778/acc http://IBM:3698/acc )
    :X-JADE-agent-classname jade.my_agent.agent_User ) )
:content  "((Robot_Located_In_Predicate (ROBOT :name P2AT)
(LOCATION :x \"5.0\" :y \"2.0\" :z \"0.0\")))"
:language  fipa-sl
:ontology  RobotOntology )
```

Figure 5.9: message ACL de notification sur la localisation du robot P2AT

Le traitement des requêtes consiste en une boucle d'attente de messages qui peuvent contenir des requêtes d'abonnement à des notifications périodiques ou des interrogations à réponses immédiates.

L'agent de localisation vérifie continuellement si les conditions d'envoi des notifications aux abonnés sont satisfaites avant de procéder à l'envoi des connaissances contextuelles demandées. Dans le mode interrogation immédiate, l'agent de contexte vérifie la disponibilité des connaissances contextuelles demandées dans sa base de connaissances interne puis envoie à l'agent demandeur une réponse. La figure 5.10 montre un message d'interrogation immédiate envoyé par l'agent utilisateur et sa réponse.

```
 (QUERY-REF
:sender  ( agent-identifier :name agent_User@IBM:1099/JADE
    :addresses (sequence http://IBM:7778/acc http://IBM:3698/acc )
    :X-JADE-agent-classname jade.my_agent.agent_User )
:receiver (set(agent-identifier:name agent_Robot_Location@IBM:1099/JADE
     :addresses (sequence http://IBM:7778/acc http://IBM:3698/acc )) )
:content "((iota ?x(Robot_Located_In_Predicate(ROBOT :name P2AT) ?x)))"
:language  fipa-sl
:ontology  RobotOntology )

(INFORM
:sender  ( agent-identifier :name agent_Robot_Location@IBM:1099/JADE
    :addresses (sequence http://IBM:7778/acc http://IBM:3698/acc ))
:receiver  (set ( agent-identifier :name agent_User@IBM:1099/JADE
    :addresses (sequence http://IBM:7778/acc http://IBM:3698/acc )
    :X-JADE-agent-classname jade.my_agent.agent_User ) )
:content  "((Robot_Located_In_Predicate (ROBOT :name P2AT)
(LOCATION :x \"5.0\" :y \"2.0\" :z \"0.0\")))"
:language  fipa-sl
:ontology  RobotOntology )
```

Figure 5.10: messages ACL d'interrogation et de réponse

La requête d'interrogation peut ne pas être satisfaite pour diverses raisons : service indisponible. Dans ce cas, l'agent de localisation se met en état d'attente fini jusqu'à ce que le service OSGi devienne à nouveau disponible. À l'expiration du délai d'attente, l'agent de localisation peut considérer qu'il s'agit d'une panne majeure et décide de mettre fin à son service en envoyant un message de désenregistrement à l'agent annuaire qui le retire ainsi de la liste des services disponibles.

L'agent de localisation des victimes (*Agent Victim Location*) se distingue des autres agents bundle de la couche OSGi inférieure par son indépendance vis-à-vis des sources de contexte. Ainsi quand un agent fournisseur de contexte comme l'agent de localisation de victime à travers la voix émise devient indisponible, il est possible de le substituer par l'agent de localisation de victimes à travers les tags RFID sans suspendre le comportement de l'agent.

Contrairement aux autres agents, l'agent de mouvement de robot offre aux agents applicatifs tel que l'agent utilisateur des services permettant d'agir sur le contexte de l'environnement par l'exécution de l'action de déplacement du robot. Cet agent possède un comportement simple, qui consiste à enregistrer son service auprès d'un agent annuaire puis à répondre aux invocations des autres agents. La réponse à une invocation consiste à traduire le contenu du message d'invocation en une action sur la localisation du robot. La figure 5.11, montre deux messages ACL, l'un envoyé par l'agent applicatif (*User Agent*) pour déplacer le robot, et la réponse envoyée par l'agent de déplacement du robot après l'exécution de la tâche.

```
(REQUEST
:sender( agent-identifier :name agent_User@IBM:1099/JADE
   :addresses (sequence http://IBM:7778/acc http://IBM:2834/acc )
   :X-JADE-agent-classname jade.my_agent.agent_User )
:receiver(set(agent-identifier:name  agent_Robot_Mouvement@IBM:1099/JADE
   :addresses (sequence http://IBM:7778/acc http://IBM:2834/acc )) )
:content  "((action (agent-identifier
    :name agent_User@IBM:1099/JADE
    :addresses (sequence http://IBM:7778/acc http://IBM:2834/acc))
    (Robot_Move_Action :ROBOT (ROBOT :name P2AT)
                    :LOCATION (LOCATION :x \"-0.83084\" :y \"0.33236\" :z \"0\")
                              :vitesse_mouving   \"10.0\" :vitesse_turning \"10.0\")))"

(INFORM
:sender  ( agent-identifier
  :name agent_Robot_Mouvement@IBM:1099/JADE
  :addresses (sequence http://IBM:7778/acc http://IBM:2834/acc )
  :X-JADE-agent-classname jade.my_agent.agent_Robot_Mouvement )
:receiver  (set ( agent-identifier :name agent_User@IBM:1099/JADE
  :addresses (sequence http://IBM:7778/acc http://IBM:2834/acc )
  :X-JADE-agent-classname jade.my_agent.agent_User ) )
:content  "((done (action (agent-identifier
  :name agent_User@IBM:1099/JADE
  :addresses (sequence http://IBM:7778/acc http://IBM:2834/acc))
   (Robot_Move_Action :ROBOT (ROBOT :name P2AT)
                   :LOCATION (LOCATION :x \"-0.83084\" :y \"0.33236\" :z \"0\")
                             :vitesse_mouving \"10.0\" :vitesse_turning \"10.0\"))))"
```

Figure 5.11: messages ACL d'action et sa réponse

À la réception des messages de notification et des réponses à ces requêtes. L'agent utilisateur utilise sa base de connaissances pour inférer les nouvelles données contextuelles et décide ensuite soit d'envoyer d'avantage de requêtes d'interrogation ou des requêtes d'action. La figure 5.13 montre quelques règles qui utilisent les données de contexte reçues pour mener des actions dans l'environnement. La fonction *move_to_victim* est une fonction java qui utilise le moteur d'inférence JESS, et qui sert à envoyer à son tour des messages ACL à l'agent de mouvement du robot pour déplacer ce dernier vers la victime.

La fonction *infer* permet quant à elle d'insérer des nouvelles connaissances dans la base de règles de l'agent applicatif.

```
(import jade.my_ontology.*)
(deftemplate Location   (declare (from-class Location)))
(deftemplate State   (declare (from-class State)))
(deftemplate Victim_Location   (declare (from-class Victim8location)))
(deftemplate Mouving   (declare (from-class Mouving)))
(deftemplate Rfid_Location   (declare (from-class Rfid_Location)))
;---------------------------------------------------
(defrule informe
 ?m <- (ACLMessage (communicative-act INFORM) (sender ?s) (receiver ?r)(content ?c)
(language ?l)(ontology ?o))
=>
(printout t "message arrivé to agent_User"crlf)
 (infer ?m)
)
;---------------------------------------------------
(defrule victim_location_rule
?fact <- (Victim_Location (name ?n) (prob ?p)(x ?var_x)(y ?var_y)(z ?var_z))
(Mouving (state no))
(State (state ok))
=>
(move_to_victim ?fact)
(retract ?fact)
)
;---------------------------------------------------
(defrule rfid_location_rule
?fact <- (Rfid_Location (id ?i) (memory ?mem)(x ?var_x)(y ?var_y)(z ?var_z))
(Mouving (state no))
(State (state ok))
 =>
 (move_to_rfid_victim ?fact)
(retract ?fact)
)
```

Figure 5.12: quelques règles d'inférence de l'agent applicatif

Comme le montre la figure 5.7, l'ensemble des agents de la plateforme partagent la même description sémantique de l'environnement à travers une ontologie *Robot Ontologie*. Ce partage est assuré par l'exportation du package *jade.my_ontology* du bundle Ontologie.

Le schéma de la figure 5.13 représente une partie de l'ontologie robot créée est générée par l'éditeur d'ontologie Protégé.

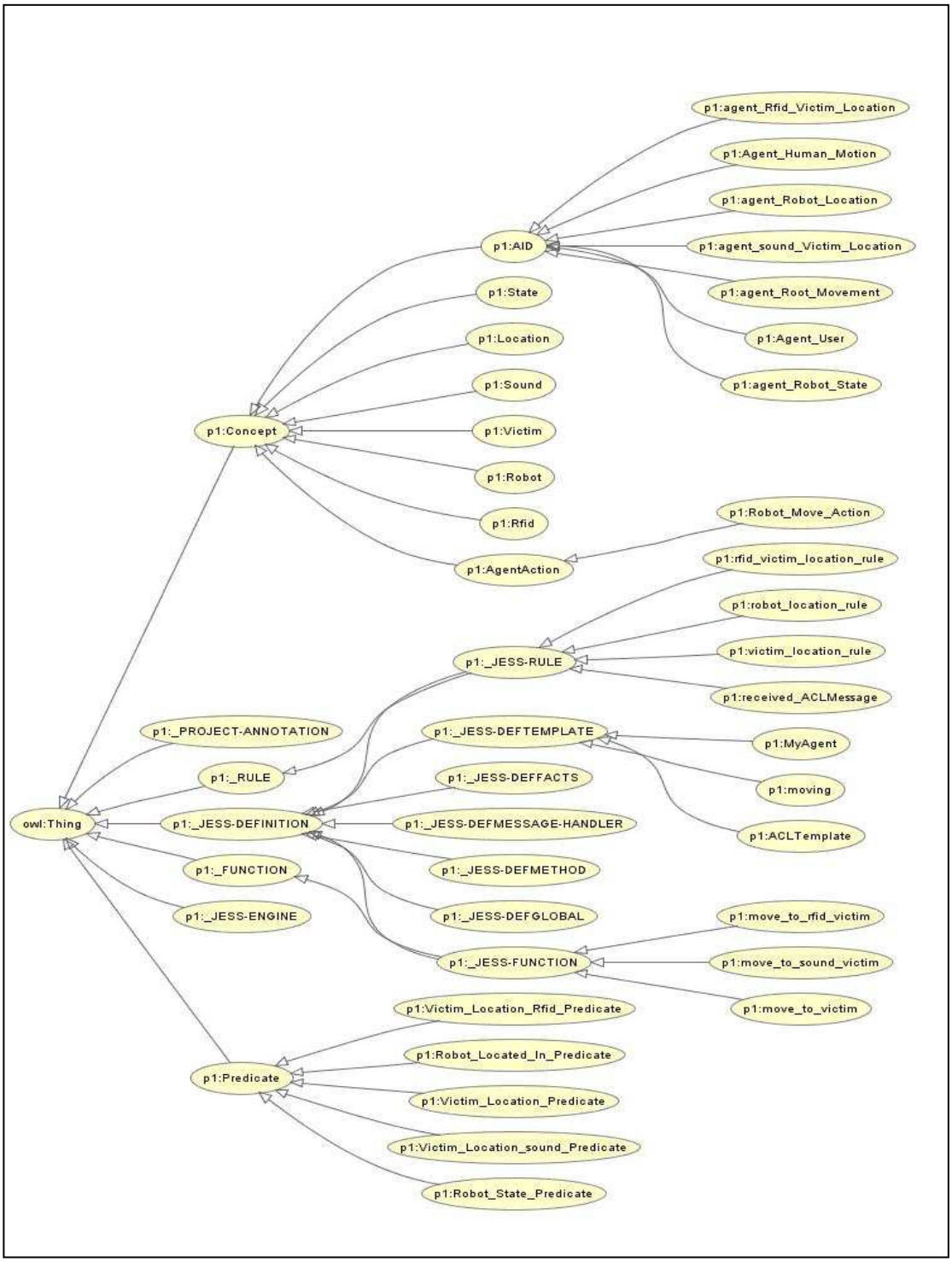

Figure 5.13: schéma de l'ontolgie Robot

5.6. Déroulement du scénario

Dans notre scénario, les agents communiquent par l'échange des messages. Nous retenons le diagramme de séquences de la figure 5.15 pour représenter les échanges de messages. Ce diagramme résume une partie du scénario dans laquelle les agents de localisation des victimes (l'agent de localisation par des mouvements, et l'agent de localisation des tags Rfid des victimes)

informent l'agent utilisateur de la présence des victimes dans l'environnement. L'agent utilisateur envoie à son tour des requêtes d'interrogation à l'agent de localisation du robot et à l'agent d'état du robot. Si le robot est en service et dans une position autre que la position de la victime (donc il doit se déplacer) l'agent utilisateur envoie à l'agent actionnaire (agent de mouvement du robot) une requête d'action pour déplacer le robot vers la victime.

Ce diagramme de séquences résume une petite partie de communication dans notre scénario. La figure 5.14 quant à elle présente un schéma de communication et d'interaction entre les agents sniffés par le snifer de la plateforme JADE.

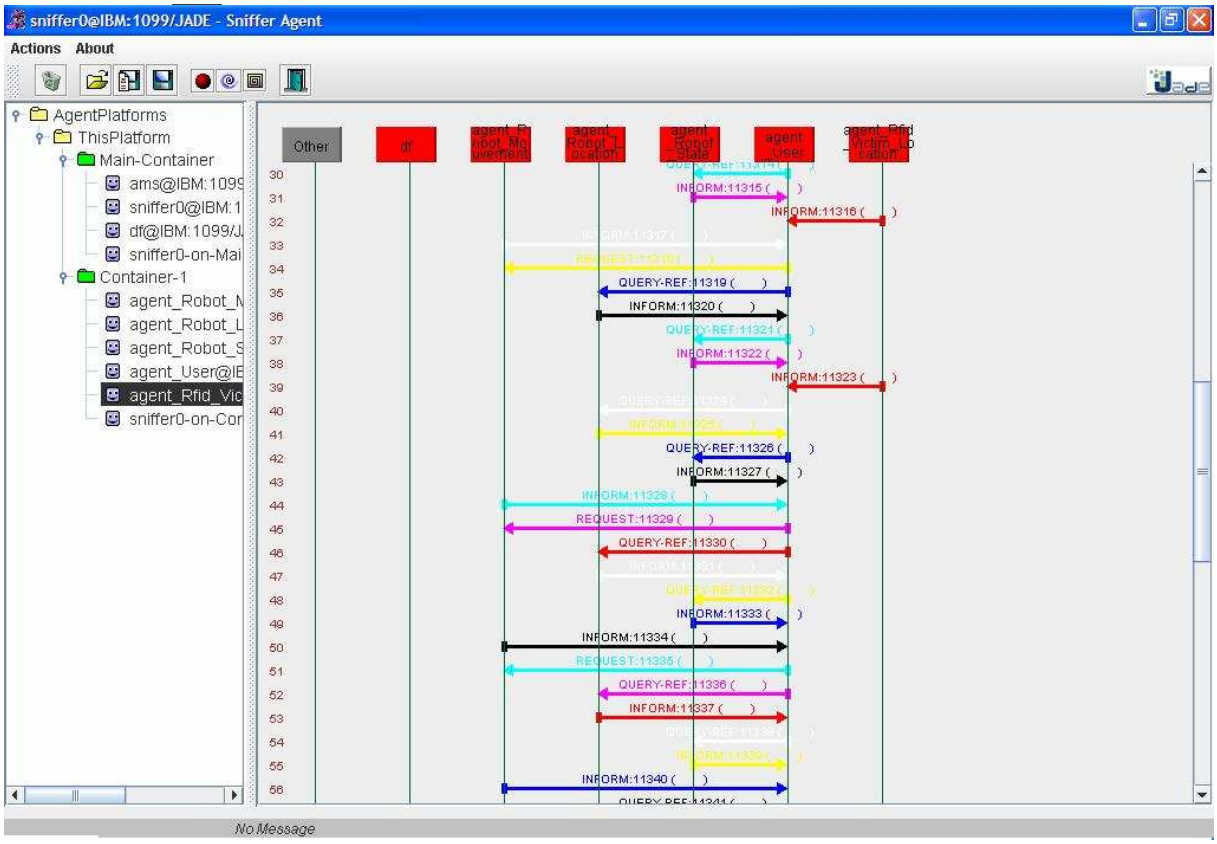

Figure 5.14: Schéma Communication et d'interaction entre agents

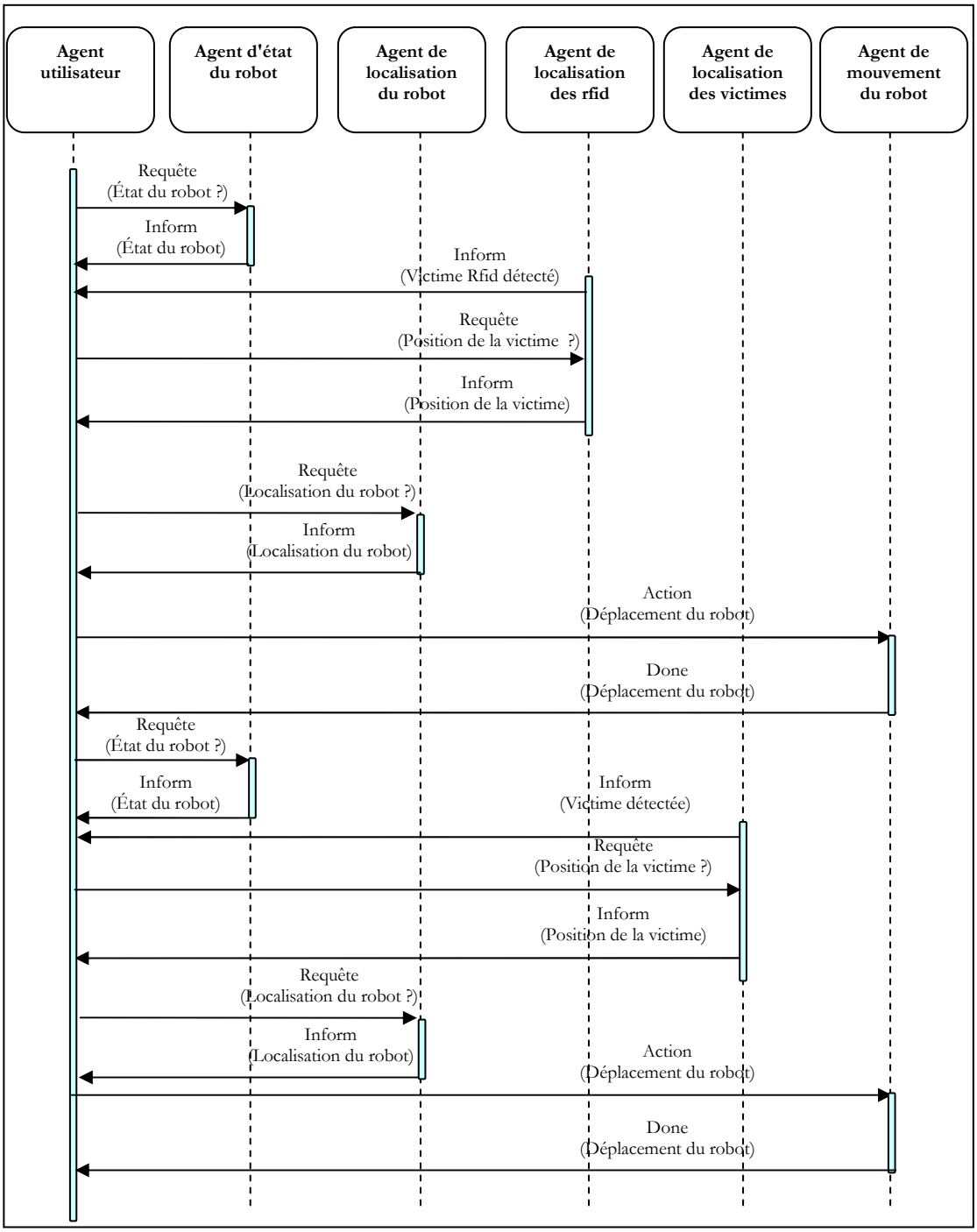

Figure 5.15: Diagramme de séquences d'une partie de la simulation

Ce diagramme de séquences résume une partie de notre scénario, par l'interaction et la communication entre agents.

Comme le montre la figure 5.16 (à partir du simulateur USARSim), l'agent de localisation des victimes par leurs tags RFID détecte la présence de trois victimes, et envoie des messages ACL à l'agent utilisateur, qui envoie à son tour une série de messages ACL d'action à l'agent de mouvement du robot après que les données contextuelles reçues de l'agent de localisation des victimes soient inférées dans sa base de connaissances contextuelles.

Figure 5.16: Environnement de la simulation

Le robot doit donc se déplacer vers les victimes (*A*, *C*, *D*). En déplaçant vers *A*, l'agent de localisation des victimes détecte une autre victime, et de la même manière, le robot se déplace vers cette victime (*B*).

Le robot continue à visiter les victimes détectées précédemment (*C* et *D*). À son arrivée à la victime *D*, l'agent de détection des victimes par les mouvements détecte les mouvements de la victime *E* et informe l'agent utilisateur, qui actionne le robot à travers l'agent de mouvement pour le déplacer vers *E*. En même temps, l'agent de localisation des victimes à travers la voix détecte la voix émise par la victime *F*, et informe l'agent utilisateur. Notre scénario se termine par la visite de cette dernière victime (Figure 5.17).

Dans notre scénario, beaucoup d'obstacles ont été supprimés de la carte pour avoir un simple scénario, puisque la détection et l'évitement d'obstacles demande encore plus de travail, ce qui n'est pas l'objet de ce présent projet, en plus, la carte utilisée dans ce scénario est la carte de l'environnement jaune (*yellow*) du simulateur USARSim, dont nous avons ajouté des mouvement aux victimes, de la voix et des tags RFID.

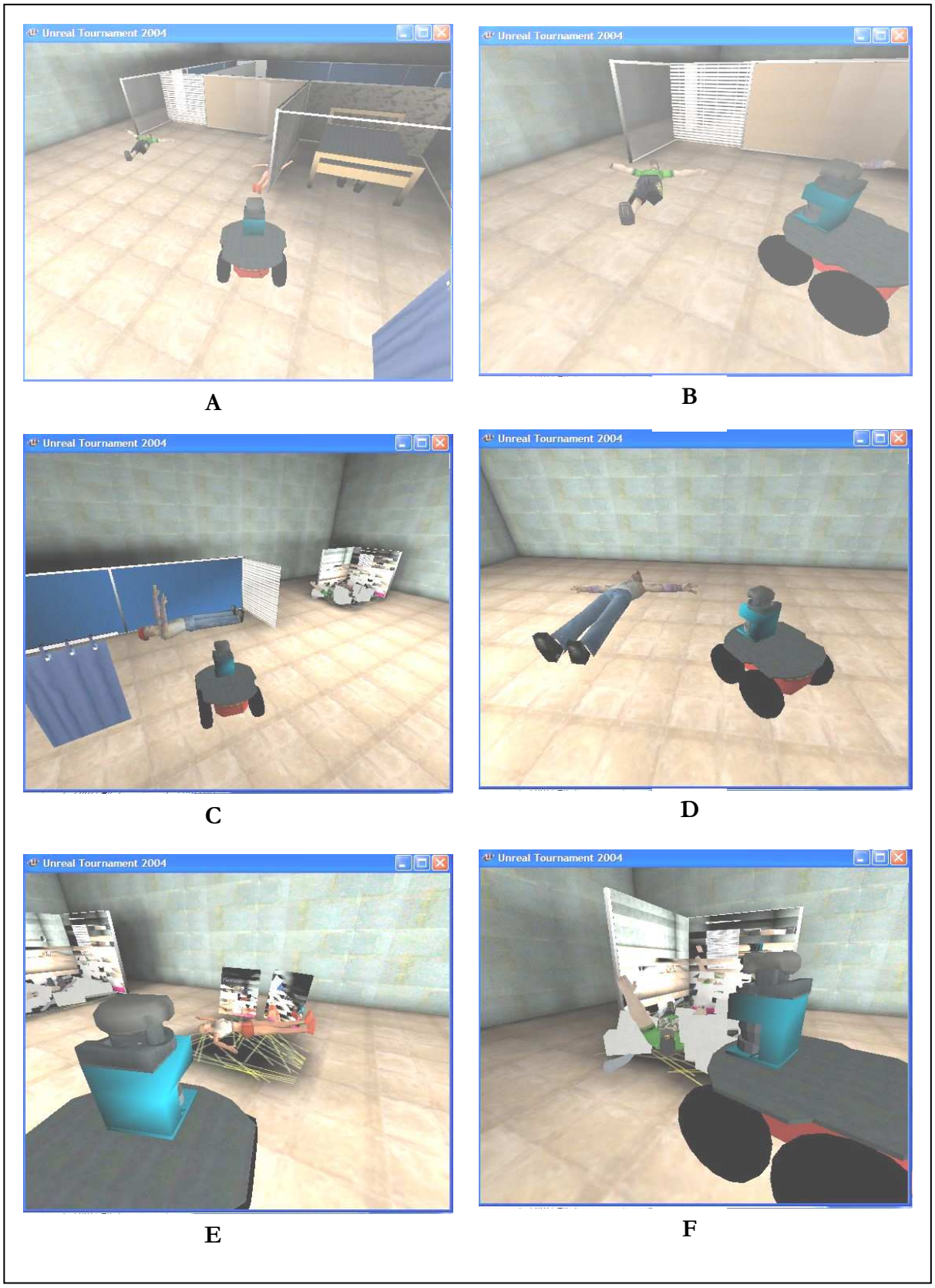

Figure 5.17: déroulement de la simulation (Victimes visitées par le robot)

Conclusion Générale

Le travail de recherche présenté dans ce mémoire fait l'objet d'une contribution qui concerne la proposition d'une architecture intergicielle orientée agent de services contextuels basé sur le standard ouvert OSGi, garantissant une décentralisation complète de la gestion du contexte. Cette architecture permet ainsi de faciliter le développement et l'intégration des agents de contexte comme des bundles dans la plateforme OSGi. La gestion de contexte repose sur les agents de contexte de la couche OSGi, ainsi, les services OSGi de cette couche offrent une abstraction vis-à-vis de l'hétérogénéité des données contextuelles brutes, et une transparence vis-à-vis des détails d'implémentation de la logique d'accès à ces données. Les agents d'agrégation du contexte ont pour rôle, quant à eux, de fournir des connaissances contextuelles inférées à partir de celles fournies par d'autres agents de contexte. Un agent d'agrégation se distingue d'un agent capteur par son indépendance vis-à-vis des sources de contexte. Conjointement aux agents de contexte, les agents actionneurs offrent une couche d'abstraction qui permet de masquer l'hétérogénéité et les détails d'implémentation des actionneurs disponibles dans l'environnement.

Enfin, au niveau applicatif, le comportement d'un agent personnel est organisé autour d'une boucle perception du contexte-décision-action. La perception correspond à la découverte et l'invocation des services de contexte, la décision à l'interprétation des connaissances contextuelles acquises et l'action au résultat de cette interprétation.

Pour bénéficier de la standardisation du framework d'OSGi, on construit le framework basé sur le système multi agents comme une couche de développement au-dessus de celle d'OSGi. Ainsi, les agents qui construisent cette couche sémantique seront installés comme des bundles dans la couche OSGi. Ces agents peuvent donc être ajoutées, supprimées et mettre à jour dynamiquement.

Ajouter les agents de la couche sémantique comme des bundles dans la couche OSGi permet de faciliter la gestion du contexte et de partager les connaissances contextuelles entre les services OSGi et les agents de la couche sémantique (basé sur le framework JADE).

Pour la mise en oeuvre et la validation de notre architecture intergicielle, nous avons adopté une approche de réutilisation des technologies logicielles existantes. La mise en oeuvre de scénario de validation nous a permis de prouver la validité de l'architecture l'intergicielle proposée. Elle a également montré la faisabilité du couplage et de l'interopérabilité *JADE-OSGI* et les avantages qu'elle présente vis-à-vis des développeurs en termes de facilité d'intégration des agents dans la agents dans la plateforme OSGi.

Perspectives

Si le présent travail nous permet aujourd'hui d'avoir une infrastructure intergicielle sensible au contexte basé sur le standard OSGi pour les services robotiques avec un faible couplage entre les deux plateformes JADE et OSGi, il reste néanmoins d'autres travaux relativement important. Sur la base du travail réalisé, nous pouvons dresser plusieurs perspectives de recherche et de développement qui sont :

- Enrichir les services OSGi par d'autres services tels que le service de détection et d'évitement d'obstacles, et le service de navigation autonome. Ces services doivent être développé et installé dans l'intergiciel.
- Prise en compte de la gestion de la sécurité et de la confidentialité du contexte ;
- Validation formelle de l'architecture intergicielle du contexte ;
- Extension du modèle proposé pour permettre l'interopérabilité des services de contexte avec les standards de services web sémantique notamment *WSDL*, *OWL-S*, *WSMO*, etc.
- Extension de la description des services pour prendre en compte la description des services des agents actionneurs.
- Développement d'autre service OSGi sur la base du protocole UPnP pour l'assistance aux personnes handicapées dans leur vie quotidienne.
- Extension du modèle de découverte des services de contexte : Il s'agit ici de proposer un nouveau modèle de découverte de services du contexte qui intègre la prise en compte de l'incertitude sur les connaissances du contexte et des fonctionnalités d'apprentissage. Ceci a pour objectif de permettre aux agents annuaires de construire une expertise qui lui permet d'améliorer les résultats de découverte.

Bibliographie

[Alami 98] R. Alami, R. Chatila, S. Fleury, M.Ghallab, F. Ingrand. An Architecture for Autonomy. International Journal of Robotics Research, Vol. 17, No. 4. pp.315-337, 1998.

[Alberto] Alberto Sanfeliuan and Juan Andrade-Cetto, Ubiquitous Networking Robotics in Urban Settings.

[BEC 05] Sean Bechhofer and Raphael Volz. WonderWeb OWL Ontology Validator. http://phoebus.cs.man.ac.uk:9999/OWL/Validator, (en ligne au 16 juin 2005).

[BEL 00] F. Bellifemine, C. Giovani, T. Tiziana, G. Rimassa. Jade Programmer's Guide, Jade version 2.6 (http://sharon.cselt.it/projects/jade/), 2000.

[Berger 02] Michael Berger, Steffen Rusitschka, Dmitri Toropov, Michael Watzke, Marc Schlichte. Porting Distributed Agent-Middleware to Small Mobile Devices Workshop on Ubiquitous Agents on embedded**.**

[Berners-Lee 01] Tim Berners-Lee, and al. The Semantic Web A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities, Scientific American, May 2001.

[Bieber 02] G. Bieber, J. Carpenter. Introduction to Service-Oriented Programming. http://www.openwings.org, 2002.

[Blank D] D. Blank, D. Kumar, L. Meeden, & H. Yanco. Pyro: A python- based versatile programming environment for teaching robotics. ACM Journal on Educational Resources in Computing (JERIC).

[Bourr] T. Bourron. Application des systèmes multi agents dans les télécommunications: États de l'art, enjeux et perspectives.

[Broo 89] R. A. Brooks. A Robot that Walks: Emergent Behaviors From a Carefully Evolved Network. A.I. Massachusetts Institute of Technology, 1989.

[Brown 97] P.J. Brown, J.D. Bovey, and X. Chen. Context-aware Applications: from the Laboratory to the Marketplace. IEEE Personal Communications, 4(5):58–64, October 1997.

[Brown 98] Brown, P. J. Triggering information by context. In personnal Technologies. v 2, p 19, 1998.

[Bruyninckx 01] H. Bruyninckx. Open robot control software: the OROCOS project. In Proceedings of IEEE international conference on robotics and automation (ICRA) (Vol. 3, pp. 2523–2528), 2001.

[Capra 00] L. Capra, W. Emmerich, and C. Mascolo. CARISMA : Context-Aware Reflective middleware System for Mobile Applications. IEEE Transactions on Software Engineering, 29(10):929– 945, oct 2000.

[Capra 01] L. Capra, W. Emmerich, and C. Mascolo. Reflective Middleware Solutions for Context-Aware Applications. In Proc. of REFLECTION. The Third International Conference

on Metalevel Architectures and Separation of Crosscutting Concerns, volume 2192 of lncs, pages 126–133, Kyoto, Japan, September 2001.

[Capra 02] L. Capra, G. S.Blair, C. Mascolo, W. Emmeric, and P.Grace. Exploiting Reflection in Mobile Computing Middleware. ACM SIGMOBILE Mobile Computing and Communications Review, 6(4) :34–44, OCT 2002.

[Capra 03] L. Capra. Reflexive Mobile Middleware for Context-Aware Applications. PhD thesis, Université de Londre, 2003**.**

[Carzaniga 98] A. Carzaniga, A. Fuggetta, R.S. Hall, A. van der Hoek, D. Heimbigner, and A.L. Wolf. A characterization framework for software deployment technologies. Technical Report CU-CS-857-98, Dept. of Computer Science, University of Colorado,1998.

[CCM 02] OMG. CORBA Components Version 3.0: An adopted Specification of the Object Management Group, June 2002.

[CCMW 01] E. Christensen, F. Curbera, G.Meredith, and S. Weerawarana. Web Services Description Langage 1.1, W3C Note. Technical report, March 2001.

[Chen 03 a] Harry Chen et al. An Intelligent Broker for Context-Aware Systems, Adjunct Proceedings of Ubicomp 2003.

[Chen 04a] 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.

[Chen 04b] 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.

[Chibani 06] A. Chibani, Intergiciel multi agents orienté web sémantique pour le développement d'applications ubiquitaires sensibles au contexte, Thèse de doctorat de l'Université Paris XII, décembre 2006.

[Chen 04c] Harry Chen et al., SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications, InProceedings, International Conference on Mobile and Ubiquitous Systems: Networking and Services, August 2004.

[Christophe 05] Christophe . Agents Mobiles Coopérants pour les environnements dynamiques , Décembre 2005.

[CL 99] B .Christensson and O.larsson. Universal Plug and Play connects Smart Devices. WinHEC 99, 199**9.**

[Cote 04] C. Cote, D. Letourneau, F.M.J. Valin, Y. Brosseau, C. Raievsky, M. Lemay, and V. Tran. Code reusability tools for programming mobile robots. In IROS. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 2004.

[CZH 99] E. Steven, Czerwinski, Ben Y. Zhao, D. Todd, Hodes, D. Joseph, and H. Katz Randy . An architecture for a secure service discovery. In mobile computing and Networking, pages 24-35, 1999.

[Demazeau 01] Yves Demazeau. Principes et architecture des systèmes multi agents, 2001.

[Dey 00] A.K. Dey, D?G Abowd. Towards a better understanding of context and contextawarness. In Computer Human Interactions (CHI2000) Workshop on the What, Who, Where and How of ContextAwarness, 2000.

[Dey 01] A.K. Dey. Understanding and Using Context, Personal and Ubiquitous Computing Journal, Vol. 5 (1), pp. 4-7,2001.

[Diethers 04] K. Diethers, N. Kohn, M. Kolbus, T. Reisinger, J. Steiner, & U. Thomas. PROSA: A Generic Control Architecture for Parallel Robots, Proceedings of MECHROB, Aachen, Germany, September 2004.

[DSIG] http://robotics.omg.org/ : OMG Robotics Domain Special Interesting Group (DSIG) Homepage.

[Fahy 04] P. Fahy and S. Clarke. CASS-Middleware for Mobile Context-Aware Applications. In 2nd International Conference on Mobile Systems, Applications and Services (Mobisys 2004), Boston, MA, June 2004.

[Ferb 95] J. Ferber, Les Systèmes Multi agents. Vers une intelligence collective. Interéditions, 1995.

[FIPA 01] http://www.fipa.org/specs/fipa00001/.

[Gaëtan Rey 05] Contexte en Interaction Homme-Machine: le contexteur. Thèse de doctorat à Université Joseph Fourier - Grenoble, 2005.

[Gandon 03] F. Gandon, and N. Sadeh. A Semantic eWallet to Reconcile Privacy and Context Awareness, Second International Semantic Web Conference, Florida, October 2003.

[Gandon 04] F. Gandon, and N. Sadeh. Semantic Web Technologies to Reconcile Privacy and Context Awareness, Web Semantics Journal. Vol. 1, No. 3, 2004.

[GCPLA 99] Y.Goland, T.cai, P.and Y.Gu P.Leach, and S.Albright. Simple Service Discovery Protocol. IETF Draft, Draft-cai-ssdp-v1-03, 1999.

[Gerkey 03] R. Vaughan, B. Gerkey, A. Howard. On device abstractions for portable, reusable robot code. In Proceedings of IROS (pp. 2121–2427). Las Vegas, Nevada, 2003.

[Gong 01] L. Gong. A Software Architecture for Open Service Gateways. IEEE Internet Computing, 2001.

[GOMEZ PEREZ 02] Gomez Perez. A Survey of Ontology Tools, Deliverable 1.3, Ontoweb, May 2002.

[Guttman 99] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol, Version 2. IETF RFC 2608, Network Working Group, June 1999.

[Haiping 03] Haiping XU. A model base approach foe development of multi-agent software systems, Chicago, 2003.

[Hang Wang 04] Xiao Hang Wang, Tao Gu, Da Qing Zhang, Hung Keng Pung. Ontology Based Context Modeling and Reasoning using OWL, 2004.

[Henning] M. Henning and M. Spruiell. Distributed Programming with Ice.

[Henricksen 02] K. Henricksen, J. Indulska, and A. Rakotonirainy. Modeling Context Information in Pervasive Computing Systems. In the First International Conference on Pervasive Computing, pages 167–180, 2002.

[Henricksen 04] K. Henricksen and J. Indulska. Modeling and Using Imperfect Context Information. In PerCom Workshops, pages 33–37, 2004.

[Henricksen 05] J. I. Karen Henricksen and T. McFadden. Modeling context information with ORM. Volume 3762 of Lecture Notes in Computer Science, pages pages 626–635, 2005.

[Henricksen 06] K. Henricksen and J. Indulska. Developing context-aware pervasive computing applications: Models and approach. Journal of Pervasive and Mobile Computing, volume 2(1) :pages 37–64, Elsevier, 2006.

[Horrocks 02] Ian Horrocks. DAML+OIL: a description logic for the semantic web. Bull. of the IEEE Computer Society Technical Committee on Data Engineering, March 2002.

[HYVÖNEN Eero 02] HYVÖNEN Eero. Semantic Web Kick-Off in Finland, Vision, Technologies, Research and Applications, HIT Publications, 2002.

[IETF] IETF : Internet Engineering Task Force web site. http://www.ietf.org/.

[Ingrand 02] F. Ingrand, and F. Py. An Execution Control System for Autonomous Robots. In IEEE International Conference on Robotics and Automation, 2002.

[Ingrand 96] F. Ingrand, R. Chatila, R. Alami, and F. Robert. PRS : A High Level Supervision and Control Language for Autonomous Mobile Robots. In IEEE International Conference on Robotics and Automation, 1996.

[JENA 05] Jena – A Semantic Web Framework for java http://jena.sourceforge.net/.

[Jeremy] Jeremy Fierstone les services web, 2002

[JDN] www.journaldunet.com/encyclopedie/definition/214/51/20/middleware.shtml

[JINI] Jini Community, Jini Architecture Specification. http://www.jini.org/standards.

[Kaupp 05] T. Kaupp. Orca robotics. http://orca-robotics.sourceforge.net/, 2005.

[Koda 96] T. Koda . Agents with Faces: A Study on the Effects of Personification of Software Agents. M.S. Thesis, Massachusetts Institute of Technology, 1996.

[Konolige 97] K. Konolige, K. Myers, E. Ruspini, & A. Saffiotti. The Saphira architecture: A design for autonomy. Journal of experimental & theoretical artificial intelligence: JETAI, 215–235, 1997.

[LabLej 93] W. Lejouad. De l'intelligence Artificielle Distribuée aux Systèmes Multi agents, Rapport de Recherche, Programme n°2, Calculs Symbolique programmation et Génie Logiciel, INRIA N°2004, 39 p, 1993.

[LaFary 05] M. LaFary, & C. Newton. Aria html documentation, Octobere 2005.

[Lhouari 06] Lhouari ROUANE, L'apport des systèmes multi agents à la recherche d'information dans les bases de données reparties, thèse de magistère, 2006.

[Makoto] Masahiko Narita & Makiko Shimamura Makoto Oya. Reliable Protocol for Robot Communication on Web Services, Hokkaido University.

[Makoto 02] Makoto Mizukawa, Hideo Matsuka' Toshihiko Koyama, Toshihiro Inukai, Akio Noda, Hirohisa Tezuka, Yasuhiko Noguchi, Nobuyuki Otera, "ORiN: Open Robot Interface for the Network – The Standard and Unified Network Interface for Industrial Robot Applications – ",Osaka 2002.

[Marc99] Marc Coté. NetSA, une architecture multi agents et son application aux services financiers », mémoire de maitre séances, Département d'informatique, faculté des séances et de génie, University Laval, Avril 1999.

[Mca2] Modular controller architecture. http://mca2.sourceforge.net/, 2005.

[Miro] Miro Manual.

[Montemerlo 03a] M. Montemerlo, N. Roy, & S. Thrun. CARMEN, Carnegie Mellon Robot Navigation Toolkit. http://carmen.sourceforge.net/, 2003.

[Montemerlo 03b] M. Montemerlo, N. Roy, & S. Thrun. Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (CARMEN) toolkit. In IROS (Vol. 3, p. 2436-2441), 2003.

[Nabiha 06] B.B. Nabiha. Ajout de mécanismes de réactivité au contexte dans les intergiciels pour composants dans le cadre d'utilisateurs nomades, thèse de doctorat au sein de l'Institut National des Télécommunications, 2006.

[Nam-Ho] Nam-Ho Kim, Yi-Seok Jeong, Sang-Hwan Ryu, Dong-Ryeol Shin. Mobile Healthcare System based on Collaboration between JADE and OSGi for Scalability, School of Information and Communication Engineering.

[Nesnas 03] I.A. Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. Estlin, Won Soo Kim. CLARAty: An Architecture for Reusable Robotic Software. SPIE Aerosense Conference, pp.121-132, 2003.

[Noriaki] Noriaki Ando, Takashi Suehiro, Kosei Kitagaki, Tetsuo Kotoku and Woo-Keun Yoon. RT-Middleware: Distributed Component Middleware for RT (Robot Technology).

[Nwa 96] H. S. Nwana. Software Agents: An Overview. In Knowledge Engineering Review, Vol. 11(3), pp. . 205-244, 1996.

[OSGi 07] Intergiciel et Construction d'Applications Réparties : http://creativecommons.org/licenses/by-nc-nd/2.0/fr/deed.fr, janvier 2007.

[OSGi wp] About the OSGi Service Platform Technical Whitepaper.

[OWL 04] OWL Web Ontology Language Overview http://www.w3.org/TR/owl-features/.

[Pascoe 98] J. Pascoe, N. S. Ryan, and D. R. Morse. Enhanced Reality Fieldwork: the Context-aware Archaeological Assistant. In V. Gaffney, M. van Leusen, and S. Exxon, editors, Computer Applications in Archaeology 1997, British Archaeological Reports, Oxford, October 1998.

[Quinot 01] Thoms Quinot, Conception et réalisation d'un intergiciel schizophrène pour la mise en oeuvre de systèmes répartis interopérables. Thèse de doctorat ENST Paris, 2001.

[RAG 05] David Rager, Troy Self. Information sur le projet vOWLidator.http://projects.semwebcentral.org/projects/vowlidator/ (en ligne au 16 juin 2005).

[RDF] Resource Description Framework (RDF) http://www.w3.org/RDF/.

[RDF 04a] RDF Primer, W3C Recommendation www.w3c.org.

[RDFS 04b] RDF Vocabulary Description Language 1.0: www.w3c.org.

[Rekesh 99] Rekesh, John. "UPnP, Jini and Salutation - A look at some popular coordination framework for future network devices." Technical Report, California Software Lab, http://www.cswl.com/whiteppr/tech/upnp.html, 1999.

[RGK97] D. Rus, R. Gray, D. Kotz. Transportable Information Agent. Journal of Intelligent Information systems. 9, 3, pp. 215-238, 1997.

[Rho00] J. Rhodes. Just-In-Time Information Retrieval. Ph.D. Thesis, MIT Media Lab, May 2000.

[Román 02] 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.

[Russell 95] S. Russell and P. Norvig. Artificial Intelligence: A Modern Approach, The Intelligent Agent Book. Prentice Hall Series in Artificial Intelligence, 1995.

[Salber 98] D. Salber, K.A. Dey,D. G. Abowd. Ubiquitous Computing: Defining an HCI research agenda for an emerging interaction paradigm. In Georgia Tech GVU technical report, Janvier 1998.

[Salber 99] D. Salber, A.K. Dey, A.K., G.D Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. Proceedings of the 1999 Conference on Human Factors in Computing Systems (CHI '99), May 15-20, pp. 434-441, 1999.

[Salutation 99] Salutation Architecture Specification, Salutation Consortium, 1999.

[Sang 06] Sang Chul Ahn, Ki-Woong Lim, Jung-Woo Lee, Heedong Ko, Yong-Moo Kwon and Hyoung-Gon Kim. UPnP Robot Middleware for Ubiquitous Robot Control, URAI 2006.

[Schmidt 02] Albrecht Schmidt. Ubiquitous Computing in Context. November, 2002.

[Schmidt, v Laerhoven 01] A. Schmidt, and K. van Laerhoven. How to build smart applications?, IEEE Personal Communications, Vol. 8, No. 4, pp.66–71, 2001.

[Schraft 00] R.D. Schraft, G. Schmierer. Service Robot , USA, 2000.

[Schilit 94] Schilit, B., Adams, N., and Want, R., Context-Aware Computing Applications, In Proc. of the Workshop on Mobile Computing Systems and Applications (Santa Cruz, CA, Dec. 1994), pp. 85-90.

[SDP 03] Specification of the Bluetooth System, Bluetooth SIG, Feb. 2003.

[SDP99] Bluetooth Specifecation Part E, Service Discovery Protocol (SDP). http://www.bluetooth.com, November 1999.

[SFW 02] S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. K. S. Gupta. Reconfigurable Context-Sensitive Middleware for Pervasive Computing. IEEE Pervasive Computing, 1(3) :33–40, 2002.

[Sheng 05] Q. Z. Sheng and B. Benatallah. ContextUML : A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services. In The 4th International Conference on Mobile Business (ICMB'05), IEEE Computer Society. Sydney, Australia., July 11-13 2005.

[Shengyuan 04] Shengyuan Luo, Ping Jiang, Jin Zhu. Software Evolution of Robot Control Based on OSGi, 2004.

[Simmons 04] R. Simmons. Inter process communication (IPC). http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/TCA/www/ipc/, 2004.

[STAN 05] Université de Stanford. The protégé ontology editor and knowledge acquisition system. http://protege.stanford.edu/.

[Stefan] Stefan Enderle, Hans Utz, Steffen Simon, Gerhard Kraetzschmar. MIRO: Middeleware For Autonomous Mobile Robots.

[STW 93] B. Schilit, M. Theimer, and B. Welch. Customizing Mobile Application. In USENIX Symposium on Mobile and Location-independent Computing, pages 129–138, Cambridge, MA, US, 1993.

[Szyperski 02] C. Szyperski, D. Gruntz, and S. Murer. Component software : beyond object-oriented programming, 2002.

[Tao Gu 04] Tao Gu et al. Toward an OSGi-Based Infrastructure for Context-Aware Applications in pervasive computing, (Vol. 3, No. 4) pp. 66-74, October-December 2004.

[Tao Gu 05] Tao. Gu, H. K. Pung, and D. Q. Zhang. A service-oriented middleware for building contextaware services. Journal of Network and Computer Applications, 28(1) :1–18, 2005.

[Théodoloz] Théodoloz Nicolas Sun ONE vs Microsoft .NET.

[Thrun 00] S. Thrun, D. Fox, W. Burgard, & F. Dellaert. Robust monte carlo localization for mobile robots. Artificial Intelligence, p99–141, 2000.

[TILAB] http://www.telecomitalialab.com/.

[TrEsp 99] E. Tranvouez, B. Espinasse. Protocoles de coopération pour le réordonnancement d'atelier. Ingénierie des Systèmes Multi agents (Actes des Journées Francophones de l'Intelligence Artificielle Distribuée et des Systèmes Multi agents), Novembre 1999.

[USARSim] Jijun Wang, USARSim Manual V3.1.1 A Game-based Simulation of mobile robots.

[Uschold 96] M. Uschold and M. Grüninger. Ontologies : principles, methods, and applications. Knowledge Engineering Review, 11(2) :93–155, 1996.

[Volpe 00] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, & H. Das. CLARAty : Coupled layer architecture for robotic autonomy. Technical Report, Jet Propulsion Laboratory, 2000.

[Ward 97] A. Ward, A. Jones, A. Hopper. A New Location Technique for the Active Office. IEEE Personal Communications 4(5) 42-47, 1997.

[Wel 99] Matt Welsh. NinjaRMI : A free java RMI. http://www.cs.berkeley.edu/mdw/proj/ninjarmi.html, 1999.

[Wikipedia] http://fr.wikipedia.org/wiki/Radio-identification.

[Wool 02] M. Wooldridge. An Introduction to Multiagent Systems, 340 p, John Wiley & Sons Publishers, Chichester, England, 2002.

[Yarp] Giorgio Metta, Paul Fitzpatrick & Lorenzo Natale, YARP :Yet Another Robot Platform, ARS Advanced Robotic Systems.

[Zarri 02] Gian Piero Zarri. Semantic Web and Knowledge Representation. DEXA Workshops, 2002.

Annexe : Les services robotiques

Les services robotiques développés

1. Les services robotiques de base

Les tableaux de cette annexe résument les services robotiques de base et optionnels développés pour le contrôle du robot.

A. Le service de déplacement du robot *(RobotMove)*

Le service de déplacement du robot se résume dans l'interface *RobotMove*, avec quatre fonctionnalités de base qui sont :

- Déplacer le robot vers l'avant (*Move_To*).
- Déplacer le robot vers l'arrière (*Move_Back*).
- Déplacer le robot vers une localité de coordonnées (x, y) (*Move_To_Coordinate*).
- Tourner le robot à droite (*Turn_Right*).
- Tourner le robot à gauche (*Turn_Left*).

Le tableau 0-1 résume le service de déplacement, toute en détaillant les entrées et les sorties ainsi qu'une explication pour chaque fonctionnalité.

Interfaces du service	Préconditions (Entrées)	Effets (Sorties)	Explication
Move_To	*Distance* et *Vitesse*	Robot dans une nouvelle localisation	Le robot se déplace d'une distance (*Distance*) avec une vitesse (*Vitesse*) vers l'avant
Move_Back	*Distance* et *Vitesse*	Robot dans une nouvelle localisation	Le robot se déplace d'une distance (*Distance*) avec une vitesse (*Vitesse*) vers l'arrière
Move_To_Coordinate	*CoordonnéeX*, *CoordonnéeY*, *Vitesse*_Mouvement et *Vitesse_Tournage*	Robot dans une nouvelle localisation de coordonnées (CoordonnéeX, CoordonnéeY).	Le robot se déplace vers la nouvelle localité de coordonnées (*CoordonnéeX*, *CoordonnéeY*).

Turn_Right	*Angle* et *Vitesse*	Robot dans une nouvelle orientation	Le robot tourne à droite d'un angle (*Angle*) avec une vitesse angulaire (*Vitesse*)
Turn_Left	*Angle* et *Vitesse*	Robot dans une nouvelle orientation	Le robot tourne à gauche d'un angle (*Angle*) avec une vitesse angulaire (*Vitesse*)

Tableau 0-1 : Le service de déplacement du robot

B. Le service du mouvement de la caméra du robot (*RobotCamera*)

- Le mouvement de la caméra du robot se résume dans les quatre actions suivantes (Tableau 0-2).
- Tourner la caméra vers le haut (*Turn_Up*).
- Tourner la caméra vers le bas (*Turn_Down*).
- Tourner la caméra vers la gauche (*Turn_Left*).
- Tourner la caméra vers la droite (*Turn_Right*).

Interfaces du service	Préconditions (Entrées)	Effets (Sorties)	Explication
Turn_Up	*Angle*	La caméra dans une nouvelle orientation	La caméra du robot tourne vers le haut d'un angle (*Angle*)
Turn_Down	*Angle*	La caméra dans une nouvelle orientation	La caméra du robot tourne vers le bas d'un angle (*Angle*)
Turn_Left	*Angle*	La caméra dans une nouvelle orientation	La caméra du robot tourne vers la gauche d'un angle (*Angle*)
Turn_Right	*Angle*	La caméra dans une nouvelle orientation	La caméra du robot tourne vers la gauche d'un angle (*Angle*)

Tableau 0-2: Le service du movement de la caméra du robot

C. Le service de localisation du robot (*RobotLocation*)

Le service de localisation du robot est la brique de base pour tout scénario de services robotiques. Ce service contient une seule fonctionnalité qui retourne les coordonnées et l'orientation du robot (Tableau 0-3) :

- Récupérer la localisation et l'orientation du robot (*Get_Robot_Location*).

Interfaces du service	Préconditions (Entrées)	Effets (Sorties)	Explication
Get_Robot_Location	*Vide*	CoordonnéeX, coordonnée et l'orientation du robot.	Retourne la localisation et l'orientation du robot

Tableau 0-3 : Le service de localisation du robot

D. Le service de l'état du robot (*RobotState*)

Le service d'état du robot vérifier en permanence l'état du robot en se basant sur l'état de sa batterie et de quelques autres équipements tels que la lampe du robot (Tableau 0-4).

Dans cet objectif, le service d'état du robot a été développé autour de quatre fonctionnalités qui sont :

- La vérification de l'état du robot (*Get_Robot_State*).
- La vérification du niveau de la batterie (*Get_Robot_Batterylevel*).
- La vérification de l'état de la lampe du robot (*Get_Robot_lightToggle*).
- La vérification de l'intensité de la lumière de la lampe (*Get_Robot_lightIntesity*).

Interfaces du service	Préconditions (Entrées)	Effets (Sorties)	Explication
Get_Robot_State	*Vide*	Retourne l'état du robot 0 = Le robot en *fonction*. 1 = Batterie *faible* 2 = Robot *en panne*	Retourne l'état du robot selon le niveau de la batterie sous forme d'un entier qui explique si le robot est *en service* ou *en panne*
Get_Robot_Batterylevel	*Vide*	rne le niveau de la e (le niveau est *rée* de vie en condes)	Retourne en millisecondes la *durée* qui reste avant que la batterie soit épuisée
Get_Robot_lightToggle	*Vide*	Retourne l'état de la lampe (*allumée* ou *éteinte*)	Retourne l'état de la *lampe* du robot
Get_Robot_lightIntesity	*Vide*	Retourne le niveau de l'intensité de la lumière émise par la lampe (entre *0* et *100*)	Retourne le niveau de l'intensité de la lumière de la lampe du robot

Tableau 0-4: Le service de l'état du robot

E. Le service du contexte brut (RobotData)

Le service du contexte brut capte toutes les données brutes envoyées par le simulateur USARSim. Ces données sont traitées puis rendues accessibles aux autres services à travers une interface de fonctionnalité. Les tableaux suivants résument l'ensemble de ces fonctionnalités par type de service.

Pour le service *RobotMove* :

L'ensemble des primitives qu'offre le service du contexte brut pour le service RobotMove sont détaillées dans le tableau suivant (tableau 0-5).

Interfaces du service	Préconditions (Entrées)	Effets (Sorties)	Explication
get_Current_posx_From_Ins	*Vide*	Position *x* du robot	Retourne la coordonnée x captée par le capteur *INS*
get_Current_posy_From_Ins	*Vide*	Position *y* du robot	Retourne la coordonnée y captée par le capteur *INS*
get_Current_posz_From_Ins	*Vide*	Position *z* du robot	Retourne la coordonnée z captée par le capteur *INS*
get_Current_angle_From_Ins	*Vide*	Angle (*alpha*)	Retourne l'orientation du robot captée par le capteur *INS*

Tableau 0-5: interfaces RobotData pour le service RobotMove

Dans chaque interface du service, on remarque l'ajout du nom du capteur. Car pour ce service de déplacement du robot deux capteurs peuvent être utilisés. Le *INS* et l'*Odometry* et chacun de ces capteurs donnent des valeurs légèrement différentes. Pour cela, l'ensemble des quatre interfaces du tableau précédent existe aussi pour le capteur Odometry, et l'utilisateur ou le développeur peut par exemple prendre la moyenne de ces données dans ces calculs.

Pour le service RobotCamera :

Le service du contexte brut (*RobotData*) offre les primitives du tableau 0-6 pour le service RobotCamera. Ces primitives sont utilisées pour récupérer l'angle d'orientation de la camera, pour être utilisée dans les primitives qu'offre RobotCamera.

Interfaces du service	Préconditions (Entrées)	Effets (Sorties)	Explication
get_Current_Camera_angle_x_From_cameraPanTilt	*Vide*	Angle (*alpha*)	Retourne l'angle de la camera du robot selon l'axe des *X* calculée par le capteur

			cameraPanTilt
get_Current_Camera_angle_y_From_cameraPanTilt	*Vide*	Angle (*alpha*)	Retourne l'angle de la camera du robot selon l'axe des *Y* calculée par le capteur *cameraPanTilt*

Tableau 0-6: interfaces RobotData pour le service RobotCamera

Pour le service RobotLocation :

Les mêmes primitives utilisées pour le service *RobotMove* sont aussi utilisées pour le service de localisation du robot (*RobotLocation*). Ces primitives concernent la coordonnée x, la coordonnée y et la coordonnée z, ainsi que l'angle d'orientation du robot.

Pour le service RobotState :

Au profil du service *RobotState*, le service *RobotData* fournit la primitive du tableau 0-7 qui permet de connaître l'état de la batterie du robot.

Interfaces du service	Préconditions (Entrées)	Effets (Sorties)	Explication
get_Current_batterylevel_From_groundvehicule	*Vide*	entier	Retourne le niveau de la batterie du robot calculé par le capteur *groundvehicule*

Tableau 0-7: interfaces RobotData pour le service RobotState

Ce service de contexte brut offre également d'autres primitives pour être utilisées par les services robotiques optionnels tels que le service de détection des victimes par le mouvement et par la voix, et le service de détection à travers les tags RFID.

2. Les services robotiques optionnels

Les tableaux de cette section résument les services optionnels OSGi développés pour un robot qui explore un environnement de désastre. Ces services comportent : un service de détection des mouvements des victimes, un service de détection de la voix des victimes, et un service de détection des victimes à travers les tags RFID.

A. Le service de détection des mouvements des victimes (*HumanMotion*)

L'interface du service de détection des victimes à travers des mouvements contient deux primitives : une pour la détection du mouvement et l'autre pour la localisation de l'objet en mouvement (Tableau 0-8) :

- Détecter le mouvement d'un objet et donner la probabilité que c'est une victime (*Get_HumanMotion_Prob*).

- Localiser l'objet (la victime) en mouvement (*Get_HumanMotion_Location*).

Interfaces du service	Préconditions (Entrées)	Effets (Sorties)	Explication
Get_HumanMotion_Prob	*Vide*	Une probabilité *p*	Retourne la probabilité (*p*) que l'objet en mouvement représente une victime.
Get_HumanMotion_Location	*Vide*	Coordonnées *(x, y, z)*	Retourne les coordonnées de l'objet en mouvement *(x, y, z)*

Tableau 0-8: Le service de détection des victimes

B. Le service de détection de la voix des victimes (*VictimSound*)

Ce service fournit deux primitives pour détecter l'amplitude et la durée de la voix de la victime, et une troisième pour la localiser (Tableau 0-9) :

- Calculer l'amplitude du son (*Get_Sound_Victim_Loudness*).
- Calculer la durée du son (*Get_Sound_Victim_Duration*).
- Déterminer la position de la victime via sa voix (*Get_Sound_Victim_Location*).

Interfaces du service	Préconditions (Entrées)	Effets (Sorties)	Explication
Get_Sound_Victim_Loudness	*Vide*	Un réel	Retourne un réel qui représente l'amplitude du son de la victime
Get_Sound_Victim_Duration	*Vide*	Un réel	Retourne un réel qui représente la durée du son de la victime
Get_Sound_Victim_Location	*Vide*	(x, y, z)	Retourne les coordonnées *(x, y, z)* de la victime

Tableau 0-9: Le service de détection de la voix des victimes

C. Le service de détection des victimes à travers les RFID (*RobotRfid*)

Ce service de détection des victimes à travers la lecture de la mémoire des tags RFID présente quatre primitives qui sont (Tableau 0-10) :

- Donner le nombre des RFID tags détecté dans l'environnement (*Get_RfidTags_Number*).
- Donner les identificateurs de tous les RFIDs présents dans l'environnement (*Get_All_RfidTags_Id*).
- Lire la mémoire d'un tag Rfid (*read_RfidTags_memory*).
- Ecrire dans la mémoire d'un tag RFID (*write_RfidTags_Memory*).
- Donner la position d'un tag RFID (*get_RfidTags_Position*).

Pour effacer la mémoire d'un RFID il suffit d'écrire la valeur "*0*" dans sa mémoire, toute fois il est possible d'utiliser la primitive *erase_RfidTags_Memory*.

Interfaces du service	Préconditions (Entrées)	Effets (Sorties)	Explication
Get_RfidTags_Number	*Vide*	Un entier	Retourne le nombre des RFID tags présent dans le champ du lecteur *RFID* du robot
Get_All_RfidTags_Id	*Vide*	Une chaîne de caractère	Retourne une chaîne de caractère de tous les tags *RFID* détectés sous la forme {id_1, id_2, … id_n}.
read_RfidTags_memory	*Id*	Une chaîne de caractère	Retourne le contenu de la mémoire du tag *RFID* dont l'identificateur est (*id*)
erase_RfidTags_memory	*Id*	{*"0"*}	Retourne la chaîne de caractère "0" qui signifier que la mémoire de *RFID* dont l'identificateur *id* est vide
get_RfidTags_Position	*Vide*	*(x, y, z)*	Retourne les coordonnées *(x, y, z)* de la victime

Tableau 0-10: Le service de détection des victimes à travers les RFID