Gestion de la migration des agents mobiles dans un environnement informatique diffus

par

Yablai Arsène Bougouyou

mémoire présenté au Département d'informatique

en vue de l'obtention du grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES

UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, Décembre 2008

Your file Votre référence
ISBN: 978-0-494-53380-2
Our file Notre référence
ISBN: 978-0-494-53380-2

Le 25 mai 2009

le jury a accepté le mémoire de M. Yablai Arsène Bougouyou dans sa version finale.

Membres du jury

M. Gabriel Girard
Directeur
Département d'informatique

M. Sylvain Giroux
Codirecteur
Département d'informatique

M. Marc Frappier
Membre
Département d'informatique

M. Bessam Abdulrazak
Président-rapporteur
Département d'informatique

Sommaire

Le laboratoire DOMUS, qui reproduit un habitat intelligent, a pour objectif de rendre les personnes en déficience cognitive plus autonomes. Dans cette optique, nous construisons un composant du système informatique de cet habitat dont le but est d'assister ces personnes dans leurs activités de la vie quotidienne. Habituellement, les patients sont assistés dans l'accomplissement de leurs activités de la vie quotidienne par un personnel médical spécialisé. Pour rendre les patients plus autonomes, nous devons construire un système autonome qui remplace autant que possible les aidants. L'analyse des besoins engendrés par cet objectif nous conduit à la construction d'un système multi-agents. Étant donné qu'un système multi-agents et un système orienté objet sont fondamentalement différents, nous adoptons aussi une méthodologie différente de celle orientée objet pour développer notre solution. Ainsi, nous avons construit un système multi-agents basé sur l'infrastructure JADE permettant de localiser de façon autonome le patient à l'intérieur de l'habitat et de faire migrer un agent automatiquement vers lui en vue de lui porter assistance, si cela s'avère nécessaire. Notre solution implante trois versions de notre système : une pour les machines de bureau et deux autres pour les systèmes embarqués dont une n'intègre pas la mobilité des agents. Notre système intégré au système de l'habitat permettra de localiser le patient en tout temps, aussi bien à l'intérieur qu'à l'extérieur de l'habitat, et de faire migrer des agents vers lui en cas de besoin.

Remerciements

Ce travail a été rendu possible grâce à certaines personnes que j'aimerais remercier du fond du cœur.

J'aimerais d'abord remercier Gabriel Girard, professeur au département d'informatique à la Faculté des sciences de l'Université de Sherbrooke, qui a supervisé ce travail en me guidant jusqu'à son terme.

J'aimerais également remercier Sylvain Giroux, professeur au département d'informatique, à la Faculté des sciences de l'Université de Sherbrooke qui a co-supervisé ce travail.

Je tiens à remercier du fond du cœur mon épouse Myrlène Perdriel, qui m'a soutenu tout au long de ce travail.

Je profite de l'occasion pour rendre hommage à ma mère qui a quitté ce monde depuis 1990 et à mon père qui a aussi quitté ce monde récemment.

Je tiens aussi à rendre hommage à mon oncle paternel, Zirignon Michel Lazilé pour tout son soutien et son encouragement à la poursuite de mes études supérieures.

Je tiens également à remercier les personnes qui ont participé à ma formation et qui m'ont soutenu en plus de m'encourager à poursuivre des études de maîtrise, notamment :

- Sylvie Villa, professeure à la HES-SO et responsable du domaine des sciences de l'ingénieur à la HES-SO en Suisse ;
- Marcos Rubinstein, professeur à la HES-SO et consultant en télécommunication en Suisse ;
- Khaled Gafaiti, professeur à la HES-SO en Suisse ;
- Jürgen Ehrensberger, professeur à la HES-SO, Suisse ;
- Jacques Hufschimid, professeur à la HES-SO, Suisse.

Je tiens aussi à remercier du fond du cœur tous ceux qui, de près ou de loin, ont contribué à ce travail.

Table des matières

Liste des abréviations

AAA :	Authentication, Authorization and Accounting
ACC :	Agent Communication Channel
ACL :	Agent Communication Language
AGUI :	Advanced Graphics and User Interface
AID :	Agent IDentifier
AMS :	Agent Management System
API :	Application Programming Interface
APDU:	Application Data Unit
AUP:	Agile Unified Process
BIND	Berkeley Internet Name Domain
CDC :	Connected Device Configuration
CLDC :	Connected Limited Device Configuration
CODEC:	COdeur DÉCodeur
CORBA:	Common Object Request Broker Architecture
CTL :	Computation Tree Logic
DF :	Directory Facilitator
DI :	Département d'informatique
DNS :	Domain Name System
EAP :	Extended Authentication Protocol
FIPA:	The Foundation for Intelligent Physical Agents
FTP :	File Transfer Protocol

HES-SO:	Haute École Spécialisée de la Suisse Occidentale
J2SE :	Java 2 Standard Edition
J2ME:	Java 2 Micro Edition
JCRMI :	Java Card RMI
JDK :	Java Development tools Kit
JSA :	JADE Semantics Add-on
LEAP :	Lightweight Extensible Agent Platform
LGPL:	Lesser General Public License
LTL :	Linear Temporal Logic
PKI:	Public Key Infrastructure
RAM :	Read Access Memory
RADIUS:	Remote Authentication Dial In User Service
RAS:	Rien À Signaler
RFID	Radio Frequency Identifier
RMI :	Remote Method Invocation
ROM :	Read-Only Memory
RPC:	Remote Procedure Call
RUP:	Rational Unified Process
SIM :	Subscriber Identifier Module
SOAP:	Simple Object Access Protocol
SOCKS:	Socket Server
TCP :	Transport Control Protocol
SL:	Semantic Language

SSL : Secure Socket Layer

UDP : User Datagram Protocol

UML: Unified Modeling Language

WPA : Wireless Protected Access

Liste des tableaux

Liste des figures

Introduction

L'explosion des nouvelles technologies de l'information et de la communication a créé de nouveaux types d'usagers, à savoir les usagers mobiles et itinérants. La baisse des coûts du matériel électronique, notamment les dispositifs de stockage tels que les disques durs et la mémoire « flash », et l'augmentation significative des largeurs de bande passante pour l'accès à Internet favorisent de plus en plus l'utilisation des terminaux mobiles tels que les ordinateurs portables, les PDA, les téléphones intelligents et les téléphones cellulaires avec des fonctionnalités très enrichies. L'accès Internet filaire et sans fil ne cesse de croître dans les ménages, ainsi que dans les entreprises et les institutions. Compte tenu de cette envergure d'interconnexion que représente Internet aujourd'hui et de l'augmentation des capacités des ressources des systèmes embarqués, le développement d'applications logicielles mobiles serait là une opportunité de répondre à certains des nombreux besoins de notre société, notamment ceux de l'assistance aux personnes en déficience cognitive dont il est question dans ce mémoire. Notre travail a pour but de gérer la migration des agents mobiles dans un environnement informatique diffus. Afin de mieux situer notre travail, nous allons introduire le projet en quatre points. D'abord, le contexte situe le cadre et l'environnement dans lequel se déroule notre projet. Puis, les objectifs présentent les buts visés par notre travail. Ensuite, nous présentons les résultats obtenus et enfin, la structure du mémoire en décrivant brièvement les sujets traités dans chaque chapitre.

Contexte

Notre travail fait partie du projet du laboratoire DOMUS de la Faculté des sciences, au Département d'informatique. Ce laboratoire reproduit un habitat intelligent pour les personnes souffrant de déficience cognitive. Il a pour but de rendre les personnes plus autonomes, c'est-à-dire permettre à celles-ci de pouvoir vaquer à leurs activités de la vie quotidienne sans l'intervention du personnel spécialisé qui les assiste habituellement dans leurs tâches.

Le laboratoire DOMUS, notre environnement de travail, est équipé d'appareils électroniques et électroménagers servant à différentes expérimentations. Soit ces appareils sont équipés de modules informatiques, soit ils sont connectés à des serveurs. Les modules informatiques de ces appareils ou les serveurs auxquels ils sont connectés hébergent des objets logiciels. Certains de ces objets logiciels sont intégrés dans notre système multi-agents.

Dans l'habitat intelligent, des capteurs sont disséminés dans des endroits précis. Ces endroits peuvent être par exemple, le tapis placé devant la cuisinière ou celui placé devant le réfrigérateur ou encore celui sous le matelas du patient. Les capteurs servent à signaler la présence du patient. Quand le patient est devant la cuisinière ou sur son lit par exemple, il exerce une pression sur les capteurs qui émettent des signaux électriques. Ces signaux électriques renseignent un objet logiciel, le *Contexte*.

Dans notre environnement informatique, nous interagissons essentiellement avec le *Contexte*. Le *Contexte* est un objet logiciel qui représente l'état de certains objets physiques et la position du patient dans son habitat. Par exemple, lorsqu'une armoire de la cuisine est ouverte, le *Contexte* est renseigné de cet état via les capteurs.

L'environnement susmentionné constitue les éléments fondamentaux sur lequel se base notre travail. Notre système est un système multi-agents qui prend appui sur l'infrastructure de cet environnement.

Objectifs

Comme mentionné plus haut dans la description du contexte de notre projet, l'objectif général est d'assister le patient dans ses activités de la vie quotidienne en réduisant autant que possible sa dépendance au support du personnel spécialisé. Cette assistance comporte plusieurs aspects, à savoir, suivre le patient partout dans l'habitat et lui apporter de l'aide si cela s'avère nécessaire. De cet objectif découlent les tâches suivantes à réaliser :

- La localisation du patient à l'intérieur et à l'extérieur de l'habitat. Pour la localisation du patient hors de l'habitat, nous nous contentons de dire que le patient est sorti.
- La logique décisionnelle qui nous permet de savoir si le patient a besoin d'aide et de déterminer l'aide spécifique qu'il faut lui apporter. Dans cette tâche, selon les actions du patient, nous devons être en mesure de déduire par inférence logique que le patient a besoin ou non d'aide et de spécifier l'aide dont il a besoin.
- L'apport de l'aide au patient. Il faut préciser que dans cette étape, nous concevons et mettons en œuvre l'infrastructure logicielle pour exécuter l'aide à apporter au patient.

Il est important, à ce stade, de préciser que la mise en place de la logique décisionnelle sort du cadre de notre travail. En lieu et place de cette logique décisionnelle, nous allons utiliser une logique de base qui est décrite dans la section présentant les résultats de notre travail.

Méthodologie

Dans le processus de conception et de développement de notre système, nous utilisons la méthodologie Gaia. Cette méthodologie a été conçue pour la construction des systèmes multi-agents.

Gaia considère les systèmes multi-agents comme des sociétés humaines organisées. Étant donné que les composants de base utilisés dans ce type de système ne sont pas des humains, nous parlons d'organisation artificielle, car les agents logiciels représentent les personnes.

À l'instar d'une organisation, notre système a un chef, des secrétaires, des délégués, etc. Les différents rôles affectés aux agents logiciels sont accompagnés de responsabilités et de permissions (droit d'accès aux ressources). Chaque agent logiciel offre des services spécifiques, lesquels sont en fait les méthodes des agents. Toute demande de service suit une procédure appelée protocole d'interaction. Le protocole d'interaction est mis en œuvre par échanges de messages entre les agents. Cette interaction entre les agents implique un ensemble d'activités. Les activités sont les tâches qu'accomplissent les agents pour offrir ou demander des services sans interagir avec d'autres agents. Ainsi, dans le processus de construction de notre système basé sur la méthodologie Gaia, les phases *d'analyse et de conception* se fondent sur les notions susmentionnées, à savoir les rôles qui sont composés de responsabilités, de permissions, de protocoles d'interactions et d'activités.

Les rôles définis dans notre organisation vont déterminer les types d'agents qui seront implantés, ces différents types d'agents représentent le modèle des agents. Les agents doivent communiquer dans le but d'accomplir les tâches devant être réalisées par notre système. L'ensemble constitué par les chemins que doivent emprunter tous les messages et les agents impliqués représente un graphe orienté, c'est l'implantation du

modèle d'accointance. Dans ce graphe, les nœuds représentent les agents et les voies empruntées, des arcs orientés. La façon dont les agents doivent interagir pour obtenir ou pour offrir un service est appelée modèle d'interaction. L'ensemble des services offerts par les agents est répertorié sous forme de tableau qui représente le modèle des services du système.

Résultats

Nous avons construit un système multi-agents dont les fonctionnalités sont présentées plus loin dans ce paragraphe. Nous avons implanté le système en trois versions :

- Une version pour ordinateur de bureau qui intègre la mobilité des agents, c'est-à-dire que les agents peuvent migrer d'un ordinateur à un autre via le réseau de l'habitat ;
- Une version pour PDA qui intègre la mobilité des agents ;
- Une autre version pour PDA qui n'intègre pas la mobilité, cependant l'aide est chargée dynamiquement à la demande.

Étant donné que la construction de notre logiciel s'appuie fortement sur l'objet logiciel *Contexte,* que nous avons présenté dans la mise en contexte, il est important de souligner que nous travaillons avec un *Contexte* fourni. Nous avons conçu notre Contexte sous forme d'une interface graphique. Le signalement du patient dans un endroit précis est simulé par l'événement d'un clic de bouton intégré à notre interface graphique. Dans cette interface graphique, il y a un champ qui permet de saisir une durée en milliseconde pour simuler le temps que passe le patient devant un appareil. Quand le temps entré est supérieur à quarante secondes, nous estimons que le patient a besoin d'aide.

Pour rendre notre système plus efficace dans la localisation du patient, nous avons conçu un sous-système basé sur des barrières infrarouges permettant de détecter l'entrée ou la sortie du patient d'une pièce donnée de l'habitat.

Notre système comprend essentiellement cinq types d'agents à savoir :

- L'agent de gestion des agents appelé « Agent Management System » (AMS) :

  L'AMS représente l'autorité dans notre système multi-agents. Entre autres, il permet aux agents de joindre la plateforme et représente aussi une sorte de pages blanches (annuaire téléphonique des abonnés d'une ville) qui répertorient tous les agents de la plateforme. Il contient aussi la description de tous les agents.

- L'agent page jaune appelé «Directory Facilitator» (DF) :

  L'agent DF représente les pages jaunes qui contiennent la description des services offerts par chaque agent qui s'inscrit dans son registre des services offerts.

- Les agents résidents :

  Les agents résidents possèdent chacun une copie de l'objet logiciel *Contexte*, à savoir l'interface graphique qui le simule. Ils sont responsables d'envoyer un message à l'agent de localisation pour l'informer d'un événement en relation avec la détection de la présence du patient.

- L'agent aidant :

  L'agent aidant est un agent mobile qui se déplace vers le patient et exécute l'aide en adéquation avec la situation éprouvée par le patient sous l'ordre de l'agent de localisation.

- L'agent de localisation :

L'agent de localisation affiche les références du patient dans un tableau et a la responsabilité d'ordonner à un agent aidant l'envoi de l'aide au patient si cela s'avère nécessaire.

L'interaction entre les agents et leur environnement fonctionnent de la façon suivante:

- Détection du patient devant un appareil :
  - Quand le patient est devant un appareil, sa présence actionne les capteurs qui à leur tour renseignent l'objet logiciel *Contexte*. L'agent résident, qui observe en permanence le *Contexte*, se rend compte que le patient est devant la cuisinière, par exemple. Il envoie un message contenant la description de l'événement à l'agent de localisation.
  - Si le temps passé devant l'appareil est supérieur à quarante secondes, l'agent résident formule un message de demande d'aide qui est envoyé à l'agent de localisation.
- Détection du patient dans une pièce de l'habitat :
  - Quand le patient entre ou sort d'une pièce, l'agent résident de la pièce se rend compte immédiatement et envoie un message d'information à l'agent de localisation.
- Localisation du patient :
  - Lorsque l'agent de localisation reçoit un message d'information provenant d'un agent résident, il affiche les informations dans un tableau lisible par l'humain.
  - Si un message provenant d'un agent résident arrive à l'agent de localisation et que celui-ci correspond à un message de demande d'aide, il extrait les informations nécessaires à l'affichage des références du patient. Ensuite, il envoie un message qui ordonne à un agent aidant d'aller vers le patient pour lui apporter le type d'aide indiqué dans le message.

- Apport d'aide au patient :
  - L'aide dans notre système est apportée par les agents aidants.
  - Pour la version de notre système sans mobilité, l'agent de localisation indique par un message à l'agent résident l'aide à charger et à exécuter.

Structure du mémoire

Le présent mémoire se subdivise en deux grandes parties à savoir, l'état de l'art et l'analyse ainsi que la construction de notre système. La première partie présente quelques travaux réalisés dans le domaine des agents mobiles et deux projets actuellement en cours. La seconde partie traite de l'analyse en vue de la conception et de l'implantation de notre système.

Dans le chapitre 1, nous présentons quelques travaux réalisés sur la mobilité logicielle dans le cadre des systèmes multi-agents. Dans le chapitre 2, nous abordons deux projets sur les systèmes multi-agents encore actifs, « Voyager Edge » et JADE (« Java Agent Development Framework »). Le chapitre 3 présente l'analyse en vue de la conception de notre solution dans laquelle nous choisissons l'approche à adopter pour le concevoir. Le chapitre 4 présente la méthodologie adoptée, à savoir, la méthodologie Gaia. Le chapitre 5 présente la modélisation de notre système. Le système étant conçu et développé, les chapitres 6 et 7 présentent notre solution, c'est l'implantation de notre développement. Le chapitre 8 se consacre aux jeux d'essais servant à tester notre système.

Partie I : État de l'art

Chapitre 1

Mobilité faible avec Java

La mobilité sous-entend deux concepts fondamentaux : d'une part, la mobilité matérielle, à savoir la mobilité des terminaux tels que les ordinateurs portables, les téléphones cellulaires, les PDA, et d'autre part, la mobilité logicielle ou mobilité des applications, qui implique le déplacement d'une application d'un site à un autre [8]. La mobilité logicielle concerne la totalité ou une partie de l'application, on parle ici de la granularité de la mobilité. Dans ce travail, il est essentiellement question de la mobilité logicielle, plus précisément de la mobilité dans les environnements diffus tels que les habitats intelligents. Après une mise en contexte, nous faisons un historique sur la mobilité logicielle. Ensuite, nous étudions les possibilités d'implantation des agents mobiles dans les éditions de Java dédiés aux systèmes embarqués, à savoir J2ME et JavaCard.

1.1 Mise en contexte

Les agents mobiles sont avant tout des agents qui ont la possibilité de migrer d'un nœud de réseau à un autre. En fait, le concept des agents mobiles n'est pas nouveau, il s'agit d'une extension du modèle client-serveur [7]. Comme les serveurs, les agents offrent des services via leurs méthodes. La demande d'un service offert par un agent se fait par message en se servant d'un protocole de communication, car les méthodes d'un agent ne sont pas directement accessibles de l'extérieur. Contrairement à un simple serveur, qui attend toujours les requêtes des clients pour réagir, l'agent peut prendre des initiatives en se déplaçant par exemple vers son client afin de réduire les flux engendrés par ses requêtes sur le réseau. Cet aspect de mobilité peut s'avérer très important dans les heures de pointe où la courbe du trafic atteint un sommet.

Dans notre projet, le langage d'implantation des agents est Java. Le «cœur» d'un agent programmé en Java est le fil d'exécution (« Thread »). L'intégration de la mobilité dans les agents écrits en Java implique aussi la possibilité de faire migrer les fils qui les intègrent. Bien que Java intègre le mécanisme de sérialisation des objets [18], il n'offre pas la possibilité de faire migrer les fils d'exécution, leur contexte d'exécution n'étant pas accessible au niveau du code. C'est pourquoi de nombreux projets d'agents mobiles utilisant le langage Java contournent le problème en utilisant soit un préprocesseur [30], soit un chargeur de classe modifié [42], soit carrément une machine virtuelle Java modifiée [45]. De façon générale, les agents mobiles, y compris la migration des fils d'exécution, ont intéressé et continuent d'intéresser le monde universitaire [3,26,34,38] et le monde de l'industrie [1,39,40,42].

1.2 Historique

Comme il a été précédemment mentionné dans la mise en contexte, les agents mobiles ne sont pas un concept nouveau. Ils ont été introduits par Xerox dans le cadre du projet *Worm* dans les années 1980 [36]. Cette idée est née du travail sur la migration des processus dans les systèmes d'exploitation au cours des années 1980. À cette époque, Xerox a conçu un programme qu'il baptisa «vers» (« worm ») qui était un programme distribué s'installant sur au moins trois machines d'un réseau pour être opérationnel. Il a été expérimenté sur un réseau de machines dont les plateformes étaient quasiment homogènes. Les parties du programme installées sur les machines du réseau sont appelées «segment». Un vers est en général constitué de trois ou quatre segments au minimum, qui représente le minimum de segments que doit avoir le vers pour « rester en vie ». Il commence son exécution sur un nœud et se propage sur d'autres machines disponibles du réseau au moyen du protocole « File Transfert Protocol » (FTP). Par cette même voie, le vers transfère aussi son état d'exécution. En arrivant sur un

nouveau nœud, il ne fait que se dupliquer et reprendre son état précédent pour continuer son exécution. Tous les segments du vers restent connectés par échanges de messages via le réseau. Chaque segment du vers est informé de son modèle et du nombre total de segments qui constitue le programme. Pour que le vers puisse s'étendre sur une machine, celle-ci doit être disponible (en veille). Pour détecter la disponibilité d'une station, le vers envoie à cette machine un paquet ayant un format spécial.

Le vers est une application qui intègre la mobilité parce qu'il est capable de faire migrer ses composants. Comme dans un système multi-agents constitué d'agents mobiles, le système lui-même ne migre pas, mais ce sont plutôt les agents, ses composants, qui migrent. À l'instar d'un système multi-agents mobiles, le vers pourrait constituer un support logiciel pour l'implantation de programmes usagers [36]. Ce vers est comparable à un intergiciel (« middleware ») utilisé pour la mobilité, à la différence que l'intergiciel ne se déplace pas. Il fournit plutôt les mécanismes sous-jacents à la migration des agents.

Bien que la mobilité du vers ne soit pas granulaire, on peut classer son mécanisme de migration dans le groupe des mobilités fortes puisque le contexte d'exécution et les données sont transférés. L'exécution continue son cours à partir de l'état transféré sur le nœud destinataire.

Le vers est l'un des premiers programmes à mettre en œuvre la migration des processus. Cette idée a été inspirée d'un travail fait au « Massachussetts Institute of Technology » (MIT) dans le cadre de l'évaluation à distance [37]. Il faut noter que plusieurs institutions avaient démarré leurs travaux sur les agents mobiles dans les années 1990. C'est Tacoma, un projet sur les systèmes multi-agents mobiles qui a motivé cet engouement [38]. En effet, Tacoma offre un support pour la migration des processus au niveau du système d'exploitation. Il permet aux processus de migrer d'un processeur à un autre. Le système Tacoma est très versatile, il fournit un support pour

les agents écrits en Tcl, Perl, C, C++, ML, Python, Scheme et Visual Basic. Avec Tacoma, chaque agent possède son contexte d'exécution dans un dossier, sous forme de chaîne de caractères ASCII, en bit non interprété avec lequel il se déplace vers un autre site. Ce dossier est appelé valise. À destination, l'agent peut reprendre son contexte d'exécution antérieur dans sa valise si l'action qu'il doit entreprendre est liée aux actions précédentes. Les agents peuvent aussi collaborer. Quand un agent veut rencontrer un autre agent, il envoie ses références contenues dans sa valise à son contact.

Le système Tacoma a aussi été implanté sur certains systèmes embarqués utilisé sur les PDA, plus exactement le *Palm Pilot*, et les appareils compatibles avec le système d'exploitation *Microsoft Windows CE*. Cette version de Tacoma, se nomme « Tacoma Lite » [17], a été conçue pour les environnements ayant des ressources très limitées pour s'adapter à ce genre de système. Des primitives pour la gestion efficace de la mémoire ont été rajoutées, les piles utilisées par les processus et les fils d'exécution ont été réduites. L'API a été aussi réduite de moitié. L'API de «Tacoma Lite» a été transformée en bibliothèque dynamique dans l'environnement *Windows CE* et en bibliothèque partagée dans l'environnement *Palm OS*. Aussi, tous les agents Tacoma ont été écrits dans le langage C afin de réduire la taille du code dans les PDA. Pour répondre à ce genre d'environnement très contraignant, il a aussi fallu rajouter une couche supplémentaire à l'architecture logicielle de Tacoma.

Il faut mentionner un personnage important dans le cheminement des agents mobiles, Dag Johansen, qui a contribué au développement des agents par la recherche et la mise en œuvre d'intergiciel dans le cadre du projet Tacoma [23].

Une année et demie après Tacoma, la firme General Magic utilise le terme **agent**. Le terme «agent mobile» a été introduit par Telescript qui supportait déjà la mobilité au niveau du langage de programmation [26].

Telescript est la première implantation du concept d'agent mobile dans le commerce électronique quand bien même le concept de commerce électronique n'existait pas encore [41]. Les abstractions implantées par Telescript sont : places, agents, voyage (« travel »), rencontres (« meetings »), connexions, autorités et permissions. La notion de place désigne des endroits virtuels d'échanges commerciaux où un agent peut acheter, par exemple, des billets pour le cinéma ou pour une partie de hockey. En somme, ce sont des supermarchés virtuels. Ces endroits peuvent être des serveurs ou tout autre environnement logiciel. La notion de voyage correspond à la mobilité, c'est-à-dire au déplacement d'un agent d'un nœud à un autre. L'agent peut quitter sa place d'origine pour aller obtenir des services sur un autre site et retourner chez lui. Par exemple, il peut aller acheter un billet d'opéra et revenir à la maison. Avec Telescript, l'agent est sous forme de bibliothèque qui contient ses procédures et son état. Quand un agent doit se déplacer, il appelle l'instruction « *go* ». Cette instruction requiert un billet; ce sont les données qui spécifient la destination de l'agent et d'autres éléments nécessaires pour sa migration. La migration d'un agent est faite dans le but d'aller à une place où un service est offert. Ce service peut être offert par un autre agent. Dans ce cas, les agents peuvent se déplacer à un endroit où ils désirent se rencontrer. À cet effet, Telescript implante la notion de rencontre (« meetings ») qui permet aux agents d'interagir sur une même place quand l'instruction « meet » est exécutée. Des agents peuvent établir des connexions à distance pour échanger des informations pratiques. Ainsi l'instruction «connect» est appelée avec en paramètre un objet représentant la cible ainsi que des paramètres de qualité de service.

En 1995, les développeurs de l'Agent TCL à Darmourt College (USA), Hanover, sortent l'un des premiers modèles d'agent mobile qui se déplace d'un site à un autre avec son contexte d'exécution et qui continue son exécution sur le site de destination [31]. En effet, le composant principal dans l'architecture logiciel de l'Agent TCL est un serveur

qui s'exécute sur chaque machine. Quand un Agent TCL veut migrer vers une nouvelle machine, il appelle la fonction *agent_jump* qui capture automatiquement son état complet et envoie les informations de cet état au serveur de destination. Le serveur de destination déploie à son tour un environnement d'exécution approprié et charge les informations de l'état d'exécution reçues, puis redémarre l'agent depuis le point exact de suspension de son exécution [26].

Il faut mentionner que David Kotz, qui a dirigé le projet Agent TCL aujourd'hui appelé D'Agent, a énormément contribué au travail sur les agents mobiles [16].

On ne saurait parler d'agents mobiles sans mentionner les «Aglet» d'IBM. Les «Aglets» doivent leur nom à Danny B. Lange qui combina les mots *agent* et *Applet*. Il faut noter que ce dernier a dirigé le projet Aglet d'IBM qu'il a quitté en 1997 pour se joindre à General Magic où il a travaillé par la suite sur le déploiement des agents mobiles.

Les Aglets d'IBM implantent la mobilité faible et sont développés en Java. Dans l'environnement des Aglets, on parle plutôt de contexte. L'agent est créé et initialisé dans un nouveau contexte où il s'exécute. Chez les Aglets, on ne parle pas de migration mais plutôt de «dispatching», c'est-à-dire du déplacement d'un agent d'un contexte source vers un contexte de destination [27].

En 1996, les chercheurs du « Stevens Institute of Technology » ont introduit les agents dans les télécommunications comme système d'achat intelligent [31].

En 1997, *Mitsubishi Electric ITA Horizon Systems Laboratory* met en place une infrastructure de mobilité [42] appelée « Concordia » développée en Java. Le système Concordia implante la mobilité faible. Pour la migration des agents, Concordia utilise la même technique que les navigateurs de la toile pour télécharger les applets Java, le chargeur de classe. En effet, Concordia utilise un chargeur de classe modifié pour transférer le code d'un site à un autre. Le chargeur de classe modifié permet d'inclure

dans le code de la classe à sérialiser, les variables de membre. Il offre aussi une infrastructure logicielle pour les systèmes embarqués.

1.3 Conditions de mise en œuvre de la mobilité

La mise en œuvre de la mobilité requiert des propriétés et des fonctionnalités que le langage et l'intergiciel doivent fournir. Ces aspects à implanter sont les suivants:

- Homogénéité de la plateforme : Les agents mobiles se déplacent d'un nœud de réseau à un autre. Les nœuds de réseau sont des machines équipées d'un système d'exploitation. Le matériel et le système d'exploitation peuvent être différents d'une machine à une autre ce qui signifie que l'environnement d'exécution peut changer. Pour permettre à un agent mobile de migrer sans problème, il est indispensable de lui offrir le même environnement d'exécution quelle que soit la plateforme de destination. Une couche logicielle supplémentaire est donc nécessaire pour offrir une homogénéité des plateformes, c'est l'un des rôles fondamentaux de l'intergiciel. Puisque nous utilisons Java dans le cadre de notre projet, l'intergiciel est la machine virtuelle Java. Cette machine virtuelle a la même spécification quelle que soit la plateforme. Cette spécification est une sorte de norme qui garantit le même comportement des instances de la machine virtuelle quelle que soit la plateforme où elles s'exécutent. Aussi, elles offrent les mêmes environnements d'exécution partout où elles sont implantées à l'exception des aspects fortement liés au matériel comme le graphisme.
- Autonomie : Une fois démarré, un agent mobile doit pouvoir continuer à s'exécuter sans intervention extérieure et être capable de prendre des décisions puis agir pour satisfaire ses buts internes. Pour cela, l'intergiciel censé offrir l'autonomie aux agents doit pouvoir offrir des sous-systèmes légers comparables

au fil d'exécution de Java afin de permettre une exécution de l'agent sans intervention extérieure.

- Migration : Cet aspect constitue l'élément fondamental d'un agent mobile. Selon le degré de mobilité, l'agent doit pouvoir migrer la combinaison de plusieurs éléments à savoir, son code, ses données et son état. Dans le cas de la mobilité faible, l'agent migre son code et ses données alors que dans la mobilité forte, il migre le code, les données et l'état d'exécution [48].
- Persistance : La persistance et la sérialisation sont similaires [49]. La persistance est un mécanisme de sauvegarde de l'état des composants dans un espace non volatile. Les composants individuels ont besoin d'emmagasiner les propriétés et les autres informations d'état afin que celles-ci soient ultérieurement relues. De cette définition, nous pouvons déduire que la persistance doit permettre de conserver le contexte d'exécution de l'agent, son code et ses données en vue de la migration ainsi que les messages qui lui sont adressés puisqu'il ne peut pas les lire durant le processus de migration.
- Communication : Dans un système multi-agent, les interactions entre les agents se font essentiellement par messages. Ces interactions respectent des protocoles bien définis et les messages sont échangés via une infrastructure de communication.
- Transparence : Quand les agents migrent, cela doit se faire sans incidence sur le fonctionnement des autres applications qui ne migrent pas et qui sont en cours d'exécution.

1.4 J2ME

Étant donné que le travail se rapporte aussi aux systèmes embarqués, il est indispensable de trouver un environnement Java destiné à ces appareils caractérisés par leurs limites en ressources telles que la mémoire, la capacité du processeur et l'autonomie en énergie. Fort heureusement, l'une des éditions de Java, « Java 2 Micro Edition » (J2ME), est dédiée à ce type de matériel. L'édition J2ME se subdivise en deux sous éditions, l'édition «Connected Limited Device Configuration» (CLDC) pour les appareils aux ressources limitées et l'édition «Connected Device Configuration» (CDC) pour les appareils embarqués ayant moins de contraintes de ressources tels que les PDA haut de gamme. Il faut noter que le CLDC est un sous-ensemble du CDC.

J2ME offre des interfaces graphiques flexibles aux utilisateurs, des protocoles de réseaux intégrés et des fils d'exécution. Les applications basées sur J2ME sont portables sur de nombreux terminaux mobiles embarqués [19].

Puisque cette édition (J2ME) est dédiée aux applications embarquées, on pense aussi à la mobilité physique c'est-à-dire la mobilité matérielle. En général, les usagers se déplacent avec leurs terminaux mobiles, leur connexion sans fil étant toujours activée. Il n'est pas toujours aisé de gérer les connexions réseaux IP liées à ce type d'appareil, car, à certains endroits, le signal pour les réseaux sans fil est très faible ce qui rend difficile voire impossible la communication. Il y a aussi des points où il existe des trous (« the dead spots ») c'est-à-dire des points où le signal est inexistant. Il est important d'avoir des protocoles qui gèrent la persistance des communications en vue de la reprise après des pertes du signal qui causent la perte de la connexion réseau. Les conditions dans les environnements embarqués peuvent être très sévères dû à leurs ressources très limitées. De ce fait, les machines virtuelles qui les équipent ne possèdent pas tous les aspects de la machine virtuelle standard. Dans un premier

temps, il faut s'assurer qu'il est possible d'implanter des fils d'exécution avec l'édition J2ME et d'avoir un environnement réseau avec des communications persistantes ou avec la possibilité de les implanter.

Puisqu'un système à agents est essentiellement basé sur la communication et que le développement d'un nouveau protocole de communication peut s'avérer long et laborieux, il est intéressant de connaître d'autres infrastructures Java qui implantent la communication. Dans la même optique, il est aussi bon de savoir qu'un système à agents est aussi un système pair-à-pair, car les agents peuvent être à la fois client et serveur. Dans les conditions susmentionnées, on pensera certainement à l'infrastructure JXTA, protocole pair-à-pair (« Peer-To-Peer ») qui offre une extension pour les systèmes embarqués compatibles avec l'environnement J2ME, JX-ME [50]. JXTA étant bâti au niveau applicatif, il peut aisément se servir des communications sans fil qui équipent très souvent les systèmes embarqués. JXTA n'est pas orienté agent et n'offre pas la mobilité, c'est-à-dire la migration des fils d'exécution, mais il peut toutefois servir de base de développement pour un système multi-agents.

1.5 JavaCard

Les cartes à puce font partie intégrantes de notre vie, elles équipent de nombreux objets de la vie quotidienne tels que les cartes de transport en commun, les cartes bancaires, les cartes de crédit, les cartes d'accès de certains bâtiments, les cartes d'étudiants, les cartes « Subscriber Identifier Module » (SIM) des téléphones cellulaires. La « javacard » est, une édition de Java conçue pour équiper les cartes à puce. Elle est aussi équipée d'une machine virtuelle Java (JavaCard JVM) spécialement conçue pour les cartes à puces. Dans notre environnement de travail qui, rappelons-le, est un environnement

informatique diffus, il est fort probable qu'on y retrouve des cartes à puces. Aussi est-il intéressant d'étudier la possibilité de son utilisation dans notre travail.

La carte Java est dédiée au développement d'applications destinées aux puces équipées de mémoire de stockage non volatiles avec ou sans microprocesseur. Les processeurs qui leur sont destinés sont sur 8 ou 16 bits, c'est aussi la taille des mots en bit que peut traiter une carte Java. Les contraintes de ces types de puces sont très sévères du fait de leur capacité de stockage très limitée (*EPROM* ou flash, 1ko de RAM et 16ko de ROM en moyenne) et de faible puissance de calcul (3,7MHz) des processeurs qui peuvent les équiper. Les cartes Java respectent le standard des cartes à puce telles que l'ISO 7816 ou Europay, MasterCard, Visa (EMV) et répondent à plusieurs standards de l'industrie. Elles sont accessibles au moyen de lecteurs de carte de 8 ou de 16 bits, avec des adaptateurs « Universal Serial Bus » (USB) ou des dispositifs d'émission et de réception sans fil intégrés répondant à la norme des «Radio Frequency Identifier» (RFID) [51].

Les seules applications Java pouvant être développées sur les cartes Java sont des applets des cartes Java [52]. La communication avec les applets des cartes Java se fait de deux façons, soit avec des messages via un protocole spécial, soit par un sous-ensemble de Java *RMI*, JCRMI (« Java Card RMI ») [52].

Il faut noter que les cartes Java sont des composants passifs du fait qu'elles ne sont pas autoalimentées, elles ne sont actives que lorsqu'elles sont accédées par un lecteur de carte ou un connecteur USB par exemple. Les micro-applications résidentes sur les cartes Java ne peuvent pas s'exécuter de façon autonome [53], d'où l'impossibilité d'y implanter des applications mobiles qui sont des applications autonomes. La figure 1 nous montre l'architecture de la JavaCard.

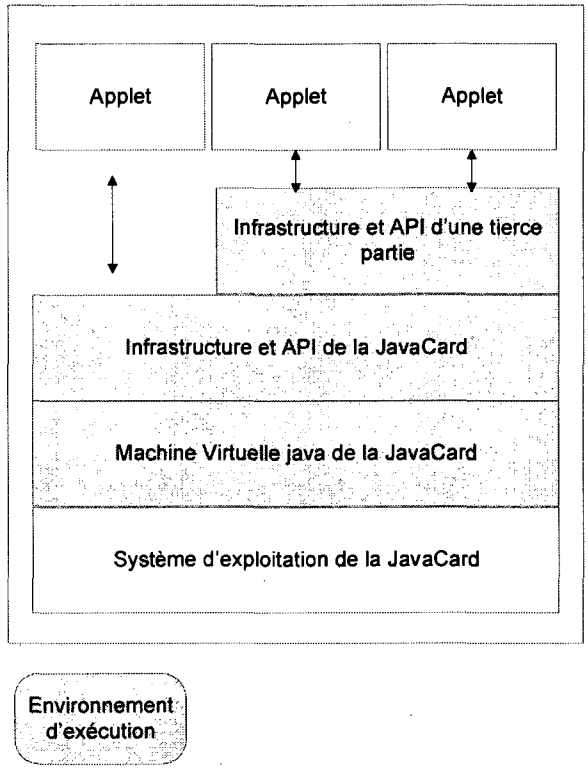


Figure 1: Architecture de la « Java Card »

1.6 Conclusion

Malgré le fait que Java n'offre pas la possibilité de faire migrer les fils, il reste toujours un très bon candidat pour le développement d'agents mobiles. La machine virtuelle Java, à elle seule, est un très bon intergiciel qui offre une très grande homogénéité des plateformes, ce qui est très avantageux pour un système à agent mobiles.

Chapitre 2

Voyager Edge et JADE

L'engouement pour les agents mobiles qui avait commencé dans les années 1990 s'est quasiment estompé aux alentours de 2002 par l'abandon de nombreux projets autant académiques qu'industriels [1, 24, 40]. Cependant, cet engouement a refait surface depuis ces dernières années par la reprise de certains projets comme *Voyager* et le lancement de nouveaux projets tel que JADE (« Java Agent Development Framework »).

Dans ce chapitre, nous présentons *Voyager Edge* et JADE, deux infrastructures pour les agents mobiles qui intègrent des aspects importants pour notre projet, à savoir, les systèmes embarqués. En plus d'intégrer les systèmes embarqués, ces infrastructures sont écrites en Java. Au moment où nous écrivons notre mémoire, d'autres travaux intéressants relatifs à la mobilité ont été publiés, nous en parlons également.

Voyager Edge

Voyager a été développé par l'entreprise *Object Space*. C'est une plateforme à agents basée sur le modèle pair-à-pair et la communication. *Voyager* est entièrement écrit en Java et implante la mobilité faible. Il a été développé avec une emphase particulière sur son architecture logicielle. L'architecture logicielle de *Voyager* a été conservée [5]. Cette plateforme utilise une machine virtuelle standard pour les éditions J2SE et J2EE de Java.

Actuellement, *Voyager* a été rebaptisé *Voyager Edge* et a été étendu aux systèmes embarqués après avoir été racheté par l'entreprise *Recursion* [35]. Il faut noter que *Voyager Edge* n'est pas gratuit.

2.1 Environnement

Dans le but de toucher un public plus large, *Voyager Edge* a été conçu pour supporter plusieurs langages de programmation. Il intègre dans son environnement des composants distants pouvant s'exécuter aussi bien dans une machine virtuelle Java que dans un environnement « .net » basé sur les langages C#, C++ ou Visual Basic (VB). Il est aussi possible de développer un système à agents basé sur *Voyager Edge* dans les langages C#, C++ ou VB. Il intègre comme support de communication, les protocoles RMI, « Simple Object Access Protocol » (SOAP), IIOP et XML-RPC. *Voyager Edge* possède aussi des fonctionnalités de découverte, c'est-à-dire, lorsqu'un dispositif intégrant un composant de *Voyager Edge* est en marche et que la connexion réseau est active, il est capable de découvrir les autres composants actifs du système se trouvant sur d'autres nœuds du réseau. C'est aussi un système multi-agents qui a pour but de fonctionner dans un environnement informatique diffus. Il fonctionne sur de nombreux systèmes embarqués tels que les PDA, les téléphones intelligents, les dispositifs RFID, etc. Il est capable de s'exécuter sur quinze types de système d'exploitation embarqués différents.

2.2 Mobilité

Voyager Edge implante la mobilité via le mécanisme de sérialisation des objets disponible au niveau du langage Java. Les agents de *Voyager Edge* ne se déplacent pas, ils sont représentés par des objets logiciels java appelés objets virtuels qui migrent en lieu et place des agents. Ces objets virtuels acquièrent les propriétés des agents après compilation. Le compilateur utilisé pour transférer les propriétés d'un agent à un objet virtuel est appelé compilateur virtuel (VCC). Il s'agit d'un compilateur spécial. Pour

migrer, un objet virtuel doit initier la migration en appelant la méthode «moveTo» avec en paramètre l'adresse du nœud de destination.

JADE

JADE est un projet initié par « Telecom Italia » et Motorola pour le développement de systèmes pair-à-pair basés sur les agents et la communication. C'est un projet qui s'inscrit dans le domaine du logiciel libre (License LGPL). L'infrastructure JADE a été développée entièrement en Java. Elle est aussi un environnement d'exécution et un environnement de développement implantant la communication avec ou sans fil. JADE a été conçue pour trois éditions de Java, à savoir, J2SE, J2EE et J2ME [4]. Avec J2ME, la mobilité est activée selon que le déploiement soit fait en mode unique (« single mode ») (avec la mobilité intégrée) ou selon un mode séparé « splitter mode ». Le « single mode » consiste à héberger un conteneur entier dans les systèmes embarqués qui se connectent au conteneur principal par contre le « splitter mode » consiste à partager le conteneur d'un serveur avec des systèmes embarqués, ce qui a pour but de réduire leurs charges vu la contrainte des ressources de ses derniers. Il faut noter que le serveur qui partage son conteneur avec les systèmes embarqués constitue une passerelle entre les environnements embarqués et ceux du réseau filaire. Par souci d'interopérabilité et de flexibilité, l'API JADE a été développé en conformité avec les spécifications de la « Foundation for Intelligent Physical Agents» (FIPA) [28]. En effet, la FIPA est une organisation qui a pour but de créer des standards architecturaux de conception pour faciliter l'interopérabilité et la flexibilité des agents issus de différentes plateformes. La communication étant un élément fondamental pour les agents, la FIPA a élaboré des structures de messages qui constituent le fondement de base pour les échanges entre les agents. Le langage ACL a été créé à cet effet. Il faut noter que la

FIPA a été acceptée depuis le 8 juin 2005 par «IEEE Computer Society» [54] comme le 11^{e} Comité pour la standardisation des spécifications [9].

2.3 Architecture abstraite de la FIPA

L'architecture abstraite de la plateforme à agents mobiles de la FIPA est présentée à la figure 2. Cette architecture définit les composants majeurs suivants :

- L'« Agent Management System » (AMS);

  L'AMS représente l'autorité sur une plateforme et il est indispensable. Il a la responsabilité de démarrer ou de détruire un agent de sa plateforme. L'AMS joue aussi le rôle des pages blanches (services d'annuaire). Il contient la description de tous les agents, notamment, leur identificateur et l'identificateur du conteneur dans lequel ils s'exécutent. Quand un agent démarre, il s'enregistre auprès de l'AMS via des messages spécifiques. Il faut noter que chaque agent possède un identificateur unique (AID). Si un agent démarre avec un AID existant, il n'est pas autorisé à joindre la plateforme, son amorçage est interrompu et l'agent en question est détruit.

- « Directory Facilitator » (DF)

  L'agent DF est un agent optionnel qui joue le rôle des pages jaunes. Il contient un registre de tous les services offerts par les agents de la plateforme. Pour qu'un service figure dans le registre du DF, il doit être explicitement enregistré par l'agent qui offre le service. Quand un agent veut demander un service, il s'adresse à cet agent (au DF) pour connaître les références de ce service. Il faut noter que le DF est unique dans une plateforme JADE.

- L'«Agent Communication Chanel» (ACC)

L'ACC, comme son nom l'indique, est une interface de communication de haut niveau que les agents utilisent pour envoyer et recevoir des messages au travers du «Message Transport Protocol» (MTP) [2], il est comparable à un port de communication, c'est un canal logique.

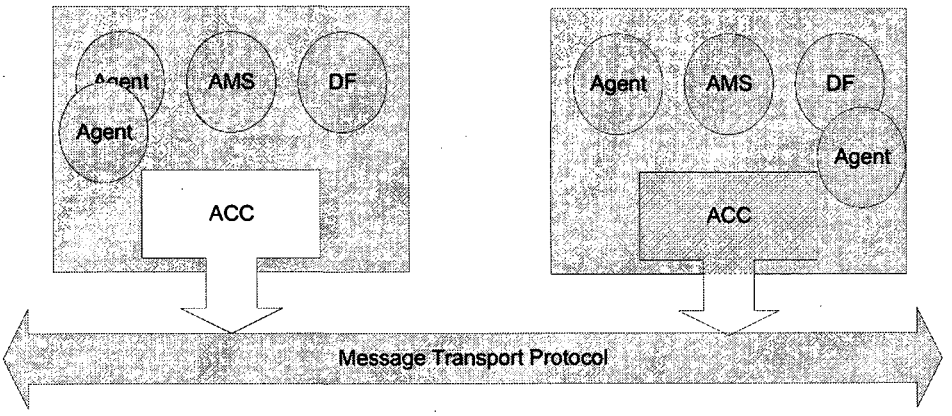


Figure 2: Architecture logicielle de référence de la plateforme à agent de la FIPA

2.4 Architecture de JADE

JADE est une infrastructure basée sur la séparation en modules de la plateforme qui facilite sa compréhension, son extension et son utilisation. La maîtrise de l'utilisation de cette plateforme nécessite une bonne compréhension de son architecture.

2.4.1 Architecture de la plateforme JADE

Le schéma de la figure 3 illustre l'architecture de la plateforme JADE. Le déploiement d'une plateforme requiert des postes avec une machine virtuelle Java installée et la pile de protocoles TCP/IP (« Transport Control Protocol/ Internet Protocol ») pour permettre aux différentes machines virtuelles de s'interconnecter. L'interconnexion entre les machines virtuelles est possible grâce au protocole « Remote Method Invocation » (RMI) ou « Jade Inter-container Protocol » (JICP) pour les environnements embarqués.

Le port de connexion par défaut est le port 1099. C'est un port qui est fermé par défaut par les pare-feu. La plateforme JADE comprend un conteneur principal et des conteneurs auxiliaires. Le conteneur principal constitue le point central de la plateforme auquel les autres conteneurs viennent se connecter.

2.4.2 Architecture logicielle

L'architecture logicielle représente la structure logicielle de JADE telle qu'illustrée à la figure 4. Les bibliothèques IIOP et «Hyper Text Transfert Protocol» (HTTP) sont des bibliothèques qui fournissent des interfaces de communication HTTP, RMI et «Common Object Request Broker Architecture » (CORBA). En fait, IIOP est une bibliothèque qui regroupe les meilleurs aspects de RMI et de CORBA entièrement écrits en Java. Pour permettre aux agents de pouvoir communiquer, une interface de haut niveau, l'ACC, est implantée. Cette interface permet d'accéder à l'infrastructure de communication pour l'envoi des messages, le MTP. Comme illustré à la figure 4, le proxy de la plateforme permet de connecter les conteneurs distants au conteneur principal de la plateforme. L'« Internal Message Transport Protocol » (IMTP) permet d'acheminer les messages entre les agents d'un même conteneur.

2.4.3 Les conteneurs

Comme nous pouvons le remarquer aux figures 3 et 4, un des concepts fondamentaux de JADE est le concept de conteneur [28]. Chaque conteneur est une instance d'exécution de la machine virtuelle Java qui permet à plusieurs agents de s'exécuter de façon concurrente en fournissant un environnement d'exécution multi-fils. Chaque agent contient un seul fil [28] et chaque machine peut héberger un seul conteneur.

Chaque conteneur dans JADE s'exécute dans une machine virtuelle différente ; cependant, elles sont toutes interconnectées par RMI, comme illustré à la figure 4.

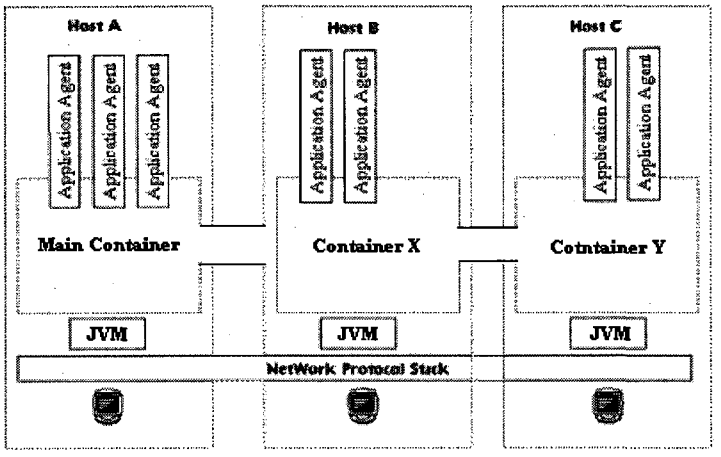


Figure 3: Architecture de la plateforme JADE

Source: http://jade.tilab.com/ *Scalability and Performance of JADE Message Transport System*

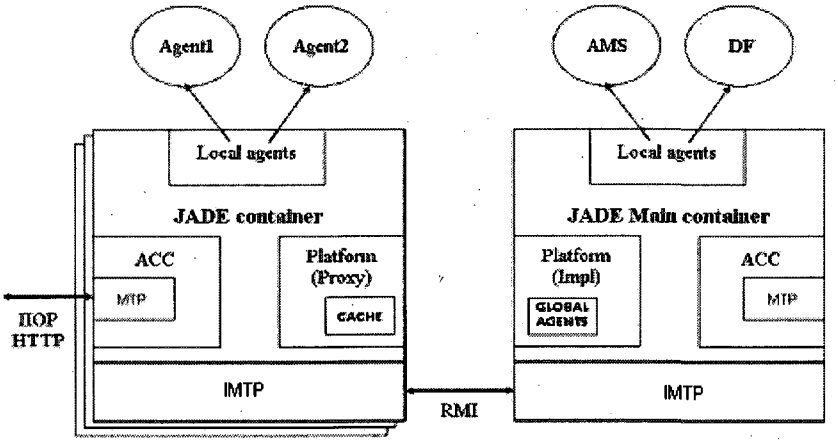


Figure 4: Architecture logicielle de la plateforme JADE

Source: http://jade.tilab.com/ *Scalability and Performance of JADE Message Transport System*

2.5 Mobilité

Dans les infrastructures JADE, les conteneurs hébergent les agents et peuvent être répartis sur plusieurs nœuds d'un réseau. La mobilité pour les agents JADE consiste à se déplacer d'un conteneur à un autre. JADE met en œuvre deux types de mobilité : la mobilité intra-plateforme et la mobilité inter-plateforme. Il faut noter qu'une plateforme JADE est constituée d'un conteneur principal qui peut être connecté à plusieurs conteneurs auxiliaires. La mobilité intra-plateforme implique le déplacement d'un agent d'un conteneur à un autre, le conteneur principal de la plateforme restant le même. Cependant, pour la mobilité inter-plateforme, l'agent se déplace d'un environnement à un autre où le conteneur principal n'est pas le même.

JADE implante la mobilité faible, ses initiateurs la qualifient de mobilité pas si faible (« not-so-weak mobility ») [4] parce que l'état de l'agent est emmagasiné dans une machine à états finie qui est récupérée après la migration.

2.6 D'autres travaux relatifs à la mobilité logique

Aujourd'hui, plusieurs travaux sur la mobilité liés aux environnements diffus ont été réalisés. L'un des travaux les plus populaires, à savoir, OSGi [56], que nous présentons plus loin, permet d'implanter des concepts de mobilité logicielle en développant des extensions. Il faut mentionner aussi que OSGi est un intergiciel basé sur la machine virtuelle java compatible avec l'édition J2ME. Un autre travail qui a aussi suscité notre attention est («Reconfigurable Ubiquitous Networked Embedded Systems») RUNES [57], un projet européen qui déploie les environnements diffus en intégrant la mobilité logique et matérielle pour les systèmes embarqués.

2.6.1 Quelques travaux sur la mobilité basés sur OSGi

L'alliance OSGi a défini la spécification OSGi pour des plateformes fournissant et gérant des services basés sur des environnements réseaux tels que ceux des habitats. À l'origine, OSGi avait pour objectif de se concentrer sur des services de passerelle interconnectant le réseau domestique et les fournisseurs de services externes, avec la possibilité de gérer ces services à distance via la passerelle. Ces services peuvent être, par exemple, l'assistance aux personnes en déficience cognitive.

La plateforme de service OSGi comprend deux éléments, à savoir, l'infrastructure OSGi et un ensemble de définitions de services standards. L'infrastructure OSGi, est un conteneur java léger pour le déploiement d'applications orientées services. Cette infrastructure définit un modèle de composant, un registre de services et fournit un environnement d'exécution pour la gestion des interactions entre ces composants.

Le modèle d'application de OSGi combine deux approches, à savoir, l'approche orientée composants et l'approche orientée service, les deux ayant pour résultante connue sous l'appellation de «modèle de service orienté composants». Dans cette dernière approche les applications logicielles sont vues comme un ensemble de composants qui collaborent entre eux, en se fournissant, et s'offrant des services.

Il faut mentionner que les composants dans une plateforme OSGi sont appelés « bundle » et ces « bundles» sont une unité de livrable et de déploiement. Ils se présentent sous forme d'archives java («.jar»). L'infrastructure OSGi assure la séparation des « bundles » sous forme de module et permet à ces «bundles» le partage de la même machine virtuelle java. La principale innovation de OSGi est d'offrir les possibilités aux « bundles » de se partager les bibliothèques java («package »). OSGi permet aussi l'utilisation du mécanisme de chargeur de classe pour résoudre le problème de dépendance des « bundles » afin de rendre des bibliothèques distantes

disponibles. Avec OSGi, un service peut être installé, mis à jour ou arrêté dynamiquement. Quand un service est arrêté, il est dynamiquement rendu indisponible de même que les services qui en dépendent. En somme, tout changement est reporté dans l'ensemble de la plateforme pour donner un fonctionnement cohérent.

Il est bon de savoir qu'il existe plusieurs versions de OSGi, Apache Felix [59], Eclipse Equinox [60], Knopflerfish [61] et Concierge [62].

Si OSGi permet d'offrir des environnements diffus, il n'offre pas la mobilité logicielle. Par contre, il existe des travaux qui, basés sur une extension de OSGi permettent d'offrir différents concepts de mobilité logicielle notamment, la mobilité des agents comme implantée dans [58]. Dans ce travail, une infrastructure qui constitue une extension de OSGi a été développée et implantée. Cette infrastructure logicielle implante la mobilité faible et le concept de services distribués. Vu au niveau de la plateforme OSGi, cette extension de OSGi est implantée sous forme de «bundle» qui déploie une infrastructure de communication en vue de mettre en œuvre la mobilité et la distribution des services. Pour mettre en œuvre cette mobilité et cette distribution de services, une adaptation du concept de «structures de données virtuelles globales» a été implanté (GVDS) [65] qui décrit le concept de la coordination d'un environnement mobile comme suite :

- Distribution : Chaque nœud possède et emmagasine une partie des structures de données;
- Construction : Pour un nœud quelconque, une vue locale de la GVDS est formée par les structures de données résidantes sur l'ensemble des nœuds atteignables ;
- Portée : Toute opération sur une structure de données distantes apparaît comme locale pour le client qui l'appelle.

Concernant la mobilité dans [58], on parle de mobilité des services, ce qui correspond aux paradigmes suivants :

- Code sur demande («Code on Demand, COD») : Avec cette approche, le «bundle» ou service peut demander le téléchargement d'une entité requise qui peut être soit un «bundle», soit un service situé sur un nœud distant. L'entité téléchargée peut en ce moment créer un lien avec le code sur le nœud de destination et s'exécuter;
- Évaluation à distance (Remote EValuation, REV ») : Dans ce cas, le « bundle » ou le service est physiquement migré ou cloné vers un nœud distant où l'exécution est en cours. Le résultat peut être délivré ou retourné avec l'entité mobile au nœud source;
- Agents mobiles : Avec le support des agents mobiles, il est possible pour un «bundle» ou un service de démarrer son exécution dans une instance d'une infrastructure pour finir dans une instance différente.

Ces mobilités, tout en coexistant, sont déployées selon que le contexte s'y prête. Par exemple, si l'économie de la bande passante s'avère nécessaire, alors ce sont les agents mobiles qui seront impliqués dans l'offre de services; si une bibliothèque est manquante pour résoudre un problème lié aux dépendances de code, la mise en œuvre du COD s'avère une alternative. De même, pour effectuer des calculs dans un environnement où la puissance de calcul fait défaut, comme c'est souvent le cas pour les téléphones portables, l'évaluation à distance peut être utilisée pour effectuer le calcul sur un nœud plus indiqué et le résultat pourra être retourné sur le téléphone portable en question.

Il faut remarquer que OSGi est très utilisé dans le déploiement d'environnements informatiques diffus dans le cadre des réseaux domestiques [63]. En plus d'être une base de déploiement d'environnements diffus, il est utilisé pour implanter la mobilité logicielle dans les habitats intelligents comme effectués dans ces travaux [64].

2.6.2 Mobilité basée sur RUNES

RUNES est construit sur des infrastructures qui constituent des composants logiciels de base. Ces composants sont des architectures réutilisables qui peuvent être déployés dynamiquement. Hautement modulaires, les composants et les services personnalisés, ainsi que les mécanismes spécifiques aux applications peuvent être construits par composition de composants ou d'infrastructures.

RUNES, est un intergiciel bâti sur une architecture à deux couches. La couche fondamentale, à savoir la couche la plus base, est un minimum conçu indépendamment du langage de programmation. Constitué de composants logiciels, cette couche est légère et s'exécute sur n'importe quel terminal constituant un nœud de réseau de systèmes embarqués.

La couche juste au-dessus de la couche fondamentale est un ensemble de composants logiciels qui offre les fonctionnalités nécessaires d'un intergiciel. Cette couche est constituée de plusieurs composants indépendants qui peuvent être sélectivement déployés selon les contraintes des ressources et le besoin des applications. Cet ensemble de composants logiciels peut être mis à jour de façon dynamique, à chaud, et constitue une base pour des systèmes hautement dynamiques et reconfigurables.

RUNES offre les fonctions de découverte des services, il possède un registre des services offerts et permet de reconnaître le contexte dans lequel le système opère en vue de fournir des informations relatives à d'autres applications pour qu'elles s'adaptent au nouveau contexte.

Concernant la mobilité logicielle, il injecte de nouvelles fonctionnalités aux terminaux comme exigences réalisées. Autrement dit, la mobilité logicielle avec RUNES est implantée sous forme de transfert de fonctionnalités d'un nœud à un autre du réseau.

Il faut noter que RUNES est implanté de quatre façons :

- une implantation basée sur la machine virtuelle java
- une implémentation basée sur C/Unix
- une implémentation basée sur les systèmes embarqués miniatures compatible avec Contiki OS
- une autre implémentation basée sur la machine virtuelle Erlang

D'un point de vue conceptuel, RUNES et OSGi sont très proches.

2.7 Conclusion

Le succès du développement d'une plateforme à agents mobiles est fortement lié à l'indépendance de la plateforme et du langage. Le langage lui-même doit être basé sur un intergiciel offrant une certaine indépendance vis-à-vis du matériel et du système d'exploitation. *Sun Microsystems* a réussi en grande partie cette tâche avec Java, mais n'offre pas encore la possibilité de faire migrer les fils d'exécution au niveau du langage.

Les projets *Voyager* et JADE offrent la mobilité dite faible. Certains agents ne peuvent pas migrer et le contexte d'exécution lié aux fils n'est pas transférable. Cependant, JADE contourne le problème en implantant le contexte d'exécution des agents dans une machine à états, raison pour laquelle on parle de mobilité pas si faible (« not so weak mobility ») [48].

Dans la suite de notre travail, nous utilisons l'infrastructure JADE pour l'implantation de notre système, car nous n'avons pas trouvé une autre infrastructure, écrite en Java, pour les systèmes multi-agents qui nous offre toutes les propriétés de la machine

virtuelle Java, à savoir l'homogénéité des plateformes et la possibilité d'être déployée dans un environnement informatique diffus. De même, JADE est un projet encore actif, il implante la mobilité faible en plus d'être gratuit et libre d'utilisation (License LGPL « Library Gnu Public License »).

Partie II : Construction de notre système multi-agents

Chapitre 3

Analyse en vue de la conception de notre système

Les moyens technologiques à notre disposition aujourd'hui nous permettent d'envisager des solutions scientifiques et technologiques dans différents domaines de la vie quotidienne, en particulier celui de l'assistance aux personnes ayant des déficiences cognitives. Les recherches du laboratoire DOMUS se consacrent, entre autres, à ce domaine. Ce laboratoire comporte un environnement informatique diffus permettant d'expérimenter l'assistance aux personnes atteintes de déficiences cognitives.

Dans le cadre de ces recherches, notre travail propose un système multi-agents dans lequel divers agents intégrés aux appareils électroniques et électroménagers sont utilisés comme moyen logiciel à des fins d'assistance. Puisque les personnes se déplacent dans la maison, certains agents devront être en mesure de suivre leurs déplacements, d'où l'intérêt d'intégrer les aspects de mobilité à certains composants. En d'autres termes, le but de ce travail est de proposer un système qui déploie un algorithme de gestion de la migration des agents.

Dans un premier temps, notre travail consiste à trouver un algorithme de migration (des protocoles d'interaction au niveau des agents) et, dans un deuxième temps, à concevoir une technique de gestion de ces agents via une interface graphique.

Vu la complexité d'implémentation d'un système multi-agents, il est indispensable de s'assurer que notre solution est en adéquation avec nos besoins. Pour cette raison, nous allons d'abord analyser nos besoins et trouver l'approche à adopter pour développer le système. Puis, nous présentons la méthodologie adoptée et, enfin, nous utilisons cette approche pour le développement de notre système.

3.1 Adoption d'une approche

Un système multi-agents est avant tout un logiciel qui se doit d'être une solution à un problème (les besoins). Cette solution se développe avec une approche bien définie comme c'est le cas pour les systèmes orienté objet. Comparé à un système orienté objet, un système multi-agents a une approche sociétale, c'est-à-dire un système qui prend des aspects de la société humaine où le système représente l'organisation et les agents, les individus. Selon Odell [33], les différences fondamentales entre un agent et un objet sont au niveau comportemental et au niveau de l'accès aux interfaces [33]. Contrairement aux objets, les agents sont autonomes [33] et ils communiquent avec des langages bien définis. D'un point de vue conceptuel, un agent et un objet ne peuvent pas jouer le même rôle. L'adoption d'une approche de conception distincte s'avère donc nécessaire selon le contexte dans lequel nous sommes.

Avons-nous besoin de recourir aux agents ? L'analyse de nos besoins est l'une des méthodes les plus appropriées pour déterminer s'il faut recourir à un système multi-agents ou non.

3.1.1 Expression des besoins

De façon générale, notre système consiste à assister des personnes souffrant de déficience cognitive dans un environnement informatique diffus. En particulier, dans ce travail, il est question d'implémenter un algorithme de gestion de la migration afin de permettre au système de pouvoir suivre la personne et de l'assister, si cela s'avère nécessaire, dans ses tâches de la vie quotidienne.

Il faut être capable de :

- Localiser la personne

Puisque la personne se déplace, il faut être en mesure de la localiser afin de lui porter assistance. C'est pourquoi notre système doit être autonome. Cela implique que des composants de notre système doivent interagir dynamiquement sans intervention extérieure pour localiser la personne.

- Référencer toutes les entités du système

  Notre système étant formé de composants qui interagissent entre eux, nous devons toujours avoir leurs références pour que chacun d'eux soit atteignable à tout moment.

- Maintenir un service d'annuaire pour les renseignements sur les services offerts

  Toutes les fonctionnalités doivent être répertoriées pour en faciliter l'accès et l'usage d'une part et permettre l'utilisation dynamique de chaque service d'autre part.

- Être proactif

  Étant donné que nous remplaçons partiellement ou totalement les aidants selon les cas, notre système a la responsabilité de remplir le rôle que joue l'aidant qu'il remplace dans une situation bien déterminée. Il doit être capable de réagir face à une situation où il est censé intervenir. Par exemple, quand le patient se déplace dans la cuisine et que celui-ci a besoin d'aide, le système doit être capable de détecter cette situation et réagir en rejoignant le patient dans la cuisine pour lui apporter l'aide nécessaire.

- Migrer lorsque cela s'avère nécessaire

  Cette migration sous-entend une certaine autonomie et une initiative du fait que notre système doit pouvoir décider lorsqu'il doit migrer dynamiquement une de ses composantes vers le patient en vue de l'assister. Obtenir cette autonomie est le but ultime.

- Assister la personne en cas de besoin

  Reconnaître que la personne a besoin d'aide est une propriété qui relève du raisonnement basé sur des connaissances appropriées de la situation. Notre système a donc besoin d'intelligence. Un mécanisme de raisonnement basé sur des règles définies en fonction du contexte est nécessaire (inférences logiques).

3.1.2 Discussion : résultat de l'analyse des besoins

Dans notre projet, où il est question d'assistance aux personnes, il est plus simple de déléguer des tâches à des agents, car ceux-ci sont autonomes (fondamentalement) d'une part et qu'ils peuvent prendre des initiatives d'autre part. En plus, certains des agents que nous implantons sont mobiles et cela permet l'économie de la bande passante, car l'agent se déplace pour offrir le service localement. Nous adoptons par conséquent un système multi-agents mobiles comme solution. On pourrait dire qu'aujourd'hui les réseaux ont des débits considérables et que l'économie de la bande passante n'est plus un argument valable. Cette opinion ne prend pas en compte le contexte de l'habitat qui est équipé d'un réseau sans fil. Les ondes électromagnétiques des réseaux sans fil sont souvent sujettes à des interférences, des bruits, des distorsions, etc. Ces effets néfastes augmentent avec le nombre d'appareils équipés de carte de réseau sans fil en activité. Les conditions du milieu de transmission occasionnent beaucoup de retransmissions des paquets dues à des erreurs, ce qui peut affecter considérablement le débit de transfert des paquets. Étant donné que nous utilisons les agents mobiles et que les services sont offerts localement, les effets de pertes de signaux sont atténués, car une fois que l'agent arrive à destination, il n'est plus nécessaire pour lui d'utiliser les connexions réseaux.

Il est vrai que le laboratoire DOMUS est aussi équipé de réseaux filaires, cependant, les différents types de réseau (filaire et sans fil) sont interconnectés et le ralentissement d'une connexion peut affecter le débit des autres réseaux.

3.3 Choix de la méthodologie à adopter

Pour développer notre solution, nous allons utiliser une méthodologie en adéquation avec l'approche choisie, c'est-à-dire celle des agents. Celles qui ont retenu notre attention sont :

- GAIA [44] ;
- MESSAGE [6] ;
- et Nikraz [32].

MESSAGE est une méthodologie riche et complexe qui inclut aussi UML et les meilleurs aspects de GAIA, cependant les artefacts sont longs et laborieux ce qui peut nous amener à passer plus de temps sur la méthodologie elle-même au lieu de se consacrer à l'essentiel. Dans le cas de la méthodologie proposée par Nikraz, elle est simple, les artefacts ne sont pas très longs, cependant, les techniques pour extraire les agents à implanter reste ambigües raison pour laquelle nous avons préféré GAIA qui nous propose des artefacts simples, très schématiques et dont la techniques d'extraction des agents est bien définie. En plus, l'étude de la méthodologie GAIA elle-même s'avère moins longue.

3.4 Conclusion

Cette méthodologie différente s'explique par le fait qu'un objet et un agent sont conceptuellement différents [33]. Les différences fondamentales entre un agent et un

objet sont l'autonomie et l'accès aux interfaces. La méthodologie à adopter pour la construction d'un système multi-agents doit tenir compte du fait que nous sommes dans une organisation artificielle [44]. À partir de ce moment, nous changeons de niveau d'abstraction par rapport aux méthodologies orientées objets. Bien entendu, dans l'implantation de notre système, les techniques d'approche orientée objets s'appliquent puisque les agents sont implantés au travers des objets logiciels. Le choix de la méthodologie de développement étant très important, nous passons du temps pour analyser les besoins dans le but de faire un choix adéquat. Si la solution de notre problème peut être apportée par un système orienté objets, il est préférable d'opter pour la solution la moins complexe, c'est-à-dire le système orienté objets comme le mentionne Wooldridge et Jennings [43].

Nous avons donc décidé d'adopter la méthodologie Gaia parce qu'elle propose une approche plus schématique, claire et facilement applicable au niveau des concepts présentés. Nous aurions pu faire une étude comparée des méthodologies susmentionnées au point 3.3, mais cela s'avère laborieux et peut nous éloigner de notre objectif. Nous avons donc regardé la simplicité et le pragmatisme des artefacts pour opérer un choix.

Chapitre 4

Présentation de la méthodologie Gaia

La méthodologie Gaia est utilisée pour concevoir et développer des systèmes multi-agents. Elle se base sur le principe qu'un système à agents doit être considéré comme une organisation, c'est-à-dire une société artificielle hiérarchisée. Naturellement, dans une organisation, il y a un chef, des secrétaires, des délégués, etc. Chaque poste correspond à un niveau dans la hiérarchie et il est accompagné de responsabilités pour les personnes qui l'occupent afin d'accomplir les tâches correspondantes. Chaque personne a des permissions (droits) qui sont associées aux responsabilités. Fondamentalement, chaque poste correspond à un ou à plusieurs rôles spécifiques.

Dans ce chapitre, nous abordons les rôles qui occupent une place majeure dans la conception. Ils sont présentés par un schéma (schéma des rôles) et sont composés d'autres concepts, à savoir, les protocoles et activités, les permissions et les responsabilités. Il faut mentionner que les services au sein de l'organisation sont offerts par des membres qui interagissent entre eux, ces interactions sont bien définies par le modèle d'interaction que nous présentons. Nous traitons également du modèle des agents qui présentent les types d'agent existant dans notre système. Ces agents offrent des services qui sont répertoriés dans un tableau, c'est le modèle des services. Nous présentons ensuite le modèle de communication qui définit la visibilité des agents entre eux du point de vue de la communication.

4.1 Processus de conception

Comme toutes les méthodologies en génie logiciel, on suit un processus. De même, selon Wooldridge, Jennings et Kinny, le processus de développement de la méthodologie Gaia se résume en trois étapes [44] :

1. Création d'un modèle d'agent ;
2. Développement des services ;
3. Développement du modèle d'accointance.

Dans ce chapitre, nous présentons ces étapes qui constituent les bases de la méthodologie Gaia, cependant, nous présentons d'abord les concepts.

4.2 Les concepts

Dans le but de déterminer les agents qui vont être implantés dans le système, Gaia procède à une analyse en vue de trouver les différents rôles qui régissent l'organisation. Étant donné que les rôles sont joués par des agents, on détermine implicitement les agents à implanter. Les rôles sont composés de plusieurs concepts que nous présentons à la section 4.2.1.

4.2.1 Le concept des rôles

Le concept de rôle défini par Gaia est constitué de plusieurs autres concepts, à savoir, les protocoles et activités, les permissions et les responsabilités. Le concept de responsabilité se subdivise en deux parties : la vivacité et la sûreté.

Il faut noter que les rôles sont représentés sous forme de schéma, illustré à la figure 5, où les différents concepts de protocoles et d'activités, de permission et de responsabilité sont exprimés avec un formalisme mathématique dont les symboles utilisés sont

présentés par le tableau 1. Ces différents concepts sont expliqués en détails dans les sections qui suivent.

Schéma des Rôles:
Nom du rôle
Description
Brève description du rôle
Protocole et activité
Protocole d'interaction : expression formelle des interactions avec d'autres agents. Activité : expression formelle des tâches que l'agent accompli sans interaction avec d'autres agents
Permissions
Lecture : Les ressources que l'agent a le droit de lire eu égard à son rôle Écriture : Les ressources que l'agent a le droit de modifier ou de créer eu égard à son rôle
Responsabilité
Vivacité: Expression formelle de l'ensemble des protocoles et activités que l'agent doit mettre en œuvre pour accomplir son rôle **Sûreté** Expression formelle des conditions nécessaires pour que l'accomplissement du rôle se déroule normalement

Figure 5: Schéma des rôles

Opérateurs	Interprétations
x/=y	x différent de y
x.y	occurrence de x ensuite de y
x|y	occurrence de x ou de y
x*	occurrence de x de 0 à N fois
x+	occurrence de x de 1 à N fois
x^{ω}	occurrence x infiniment souvent
[x]	x est optionnel
x||y	x et y sont entrelacés

Tableau 1: Légende des symboles utilisés

Dans le schéma des rôles présenté à la figure 5, nous donnons une brève définition des notions qui constituent les rôles. Pour mieux éclairer les personnes, nous définissons davantage ces notions dans les points suivants :

- **Les responsabilités** : Les responsabilités sont les tâches accomplies par un agent. Le rôle de l'agent est d'assumer les tâches qui lui sont assignées. De même, les responsabilités se subdivisent en deux parties. La première partie représente l'action faite pour accomplir la tâche exprimée dans un formalisme bien définie, c'est la **vivacité.** La deuxième partie, la **sûreté**, exprimée dans le même formalisme que le premier, a pour but de confirmer que l'action se déroule normalement.
- **Les permissions**: C'est l'ensemble des droits qui permettent à l'agent d'accomplir ses tâches par rapport aux ressources avec lesquelles il travaille, par exemple un fichier, une base de données, des ports de communication, etc. Selon les droits qui lui sont accordés, l'agent doit être capable de lire, de créer ou de modifier des ressources.
- **Les activités** : C'est l'ensemble des actions élémentaires qui sont en adéquation avec le rôle que joue l'agent dans le système sans interaction avec d'autres agents. L'observation permanente de l'objet logiciel *Contexte* par un agent en est un exemple.
- **Les protocoles** : Les protocoles représentent l'ensemble des interactions que peut avoir un agent avec d'autres agents dans le système. En somme, il s'agit de l'ensemble des interactions définies pour interagir avec d'autres membres du système.

4.2.2 Le modèle d'interaction

Comme dans toute organisation, les membres sont appelés à interagir entre eux ou avec une tierce partie que l'on nomme aussi la clientèle selon les besoins. À l'instar d'un employé de la banque qui doit ouvrir un compte à un client, notre système doit suivre un ensemble de procédure puisque notre méthodologie se base toujours sur le fait que nous sommes dans une organisation. Bon nombre de ces procédures sont interactives. Les procédures à suivre sont bien définies par la banque et l'employé doit interagir avec le client en vue de recueillir les informations justes et nécessaires à l'ouverture du compte bancaire.

En d'autres termes, le modèle d'interaction nous montre de façon formelle les relations existantes entre les différents agents dans l'organisation. Il nous montre également une abstraction des protocoles et des activités qui peuvent avoir lieu entre les agents sans toutefois montrer tous les messages qui sont échangés.

Le schéma de la figure 6 illustre un modèle d'interaction où nous décrivons brièvement l'interaction et les entités qui interagissent. Les différents éléments du schéma sont :

- Description de l'interaction : Dans cette partie nous donnons une brève description de l'interaction qui a lieu, par exemple, la détection du patient devant la cuisinière.
- Entité A et Entité B : Ce sont des entités qui interagissent dans le but de réaliser une action.
- Description de la réaction : C'est une brève description de la réaction que l'une ou l'autre des entités aura dans le but de produire un résultat. Ce résultat peut être par exemple un message d'information envoyé à l'agent de localisation.

- Entrée : Une entrée peut être un message, un signal électrique ou tout autres stimuli. Une entrée peut être par exemple un message de demande de service à un agent.
- Sortie : C'est le résultat du traitement de l'entrée par les entités A ou B. C'est aussi le résultat des interactions entre les entités A et B.

Si cette interaction engendre une autre interaction, alors nous mettons une flèche qui part de la case de la description de la réaction vers un autre schéma qui décrit la prochaine interaction. Cette flèche représente aussi la communication, à savoir la transmission des résultats de l'interaction en question qui sera une entrée pour la prochaine interaction.

Nous utilisons dans ce travail, ce type de schéma pour illustrer les interactions entre les agents eux-mêmes et aussi entre les agents et des objets de leur environnement dans le processus de développement de notre système.

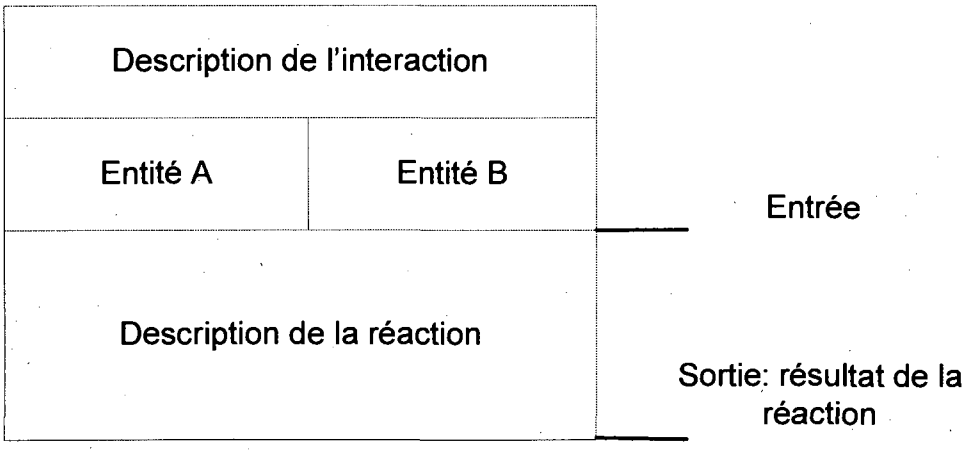


Figure 6: Interaction d'un agent résident avec le Contexte

4.2.3 Le modèle des agents

Le modèle des agents dans le processus de développement Gaia constitue la première étape. Il a pour but de documenter les différents types d'agents qui seront présents dans

le système. On peut créer un agent par rôle ou regrouper les rôles cohérents dans un seul type d'agent, tout dépendant du type d'applications et du matériel cible. Un agent par rôle nécessite un espace mémoire assez important parce qu'il donne lieu à la création de nombreux agents. Cette approche est désavantageuse pour les processeurs de faible puissance de calcul et ayant peu de mémoire tels que ceux des systèmes embarqués bas de gamme.

Pour illustrer le modèle des agents, Gaia adopte le schéma de la figure 7. Les deux termes que nous retrouvons dans ce schéma sont les suivants :

- Agent : C'est l'agent qui joue le rôle spécifié
- Rôle : Un rôle défini dans l'organisation que représente le système multi-agents

Figure 7: Documentation d'un type d'agent

4.2.4 Le modèle des services

Services	Entrées	Sorties	Pré-conditions	Post-conditions
Identifiant du service offert	Stimuli(s) d'entrée(s)	Message(s) ou action(s) exécuté(es) par l'agent	Conditions préalables à la demande de service	Conditions nécessaires pour que la demande reste toujours valide

Tableau 2: Modèle des Services

Le modèle des services définit l'ensemble des services que les agents peuvent fournir, sa création est la troisième étape de développement. En programmation orientée objets, les services sont les méthodes. Dans le cas des agents, les méthodes ne sont pas directement accessibles de l'extérieur donc la sollicitation d'un service se fait via les messages. Les messages sont transportés par un protocole déployé dans le système multi-agents.

En somme, le modèle des services est illustré par un tableau (tableau 2) à deux dimensions qui répertorie l'ensemble des services offerts par les agents de la plateforme. Le nom des services est représenté par une colonne, les pré-conditions et les post-conditions à l'obtention des services offerts constituent deux autres colonnes du tableau. Les entrées qui représentent les demandes de services et les sorties qui sont les résultats de ces demandes sont également deux autres colonnes du tableau.

À la section 5.7, nous présentons notre modèle des services qui est un exemple.

4.2.5 Le modèle d'accointance

Dans cette section, il est question de mettre en évidence la visibilité des agents entre eux à l'intérieur du système. La création d'un modèle d'accointance constitue la troisième étape de développement de la méthodologie Gaia. Ce modèle représente l'ensemble des communications potentielles qu'il y a entre les agents. C'est un graphe orienté dont les nœuds sont les agents et les arcs sont les voies empruntées par les messages échangés. Ce modèle ne donne pas de détails sur les messages échangés ni quand ni comment ils sont échangés. Cependant, il nous montre un fait important de la conception du système, à savoir, si le système est faiblement couplé ou fortement

couplé. S'il est fortement couplé, une revue de l'analyse à la conception de notre système s'avère donc nécessaire.

4.3 Conclusion

La méthodologie Gaia est simple et pragmatique parce qu'elle se fonde sur les organisations de la société humaines, un univers avec lequel nous sommes familiers. Elle permet aux développeurs de se concentrer sur l'essentiel par la simplicité des artefacts qu'elle propose. Elle facilite la conception des agents à implanter dans le système par l'analyse des rôles. Cependant le formalisme dans lequel est exprimé les composantes du rôle ne contient pas des symboles pour exprimer les notions telles que « occurrence de X jusqu'à ce que », « toujours vrai ». Toutefois cela n'empêche pas l'utilisateur d'intégrer d'autres outils pour mieux s'exprimer. C'est pourquoi nous suggérons aux utilisateurs de la méthodologie Gaia, dans le cas des applications critiques, d'utiliser un langage bien approprié à l'instar de CTL, de LTL et d'EB^3 [10], pour exprimer des notions plus complexe comme celles susmentionnées plus haut dans ce paragraphe. En plus, le modèle d'interaction ne donne pas une vue dynamique assez détaillée des interactions, les paramètres des messages et leurs contenus en sont des exemples de détails qui ne sont pas visibles.

Chapitre 5

Modélisation de notre système basée sur Gaia

Comme dans tout processus de développement en génie logiciel, nous suivons une méthodologie que nous adoptons après l'analyse des besoins. Dans ce chapitre, nous développons notre système en utilisant la méthodologie Gaia. Ce processus consiste essentiellement à créer un modèle des agents, un modèle des services et de concevoir l'architecture de communication. Le modèle des agents est important parce qu'il permet de créer les types d'agents qui sont implantés dans notre système. Ainsi, l'identification des rôles que nous abordons en premier, nous sert à créer un modèle des agents.

5.1 Identification des rôles

La création des agents étant basée sur les rôles, leur détermination s'impose avant la conception du modèle des agents. D'après l'analyse des besoins, les différents rôles qui se dégagent sont les suivants :

- la localisation du patient :
  1. Traiter et afficher des informations de localisation ;
  2. Donner l'ordre à un agent de migrer vers le patient ;
  3. Détecter la personne devant un électroménager ou un appareil électronique ;
  4. Détecter la personne à l'intérieur ou à l'extérieur d'une pièce ;
- le service d'annuaire ;
- le service des pages jaunes ;
- l'assistance aux personnes.

5.1.1 Localisation du patient

Les sous-rôles de localisation énumérés à la section 5.1 sont regroupés en deux parties pour des raisons de modularité et de cohésion. L'agent qui reçoit les informations de références du patient est le plus habilité à traiter ces informations de localisation, les afficher et donner l'ordre aux agents mobiles de migrer vers le patient quand cela s'avère nécessaire. Le rôle de cet agent résident sera le sous-rôle 1 et se nomme agent de localisation puisqu'il permet de faire connaître via son interface graphique la localisation du patient. Par contre les agents qui ont accès à l'objet logiciel *Contexte* sont les plus habilités à détecter la présence du patient dans une pièce de l'habitat et à transmettre les références du patient, nous nommons ces agents, les agents résidents qui ont la responsabilité d'assumer le sous-rôle 2.

Sous-rôle 1 : service de localisation

La figure 8 illustre les deux premiers sous-rôles de localisation du patient énumérés au point 5.1, à savoir traiter des informations de localisation et donner l'ordre au patient de migrer. Ce sous-rôle constitue le cœur du fonctionnement de notre système, c'est le sous-rôle 1. Chaque fois que la présence du patient est détectée dans un point de l'habitat, l'agent responsable de la localisation, l'agent de localisation (AL), reçoit un message qui contient les références du patient et le contexte de la situation, à savoir si ce dernier a besoin d'aide ou pas. L'agent de localisation affiche les informations qu'il reçoit dans le tableau de son interface graphique. Si l'information sur la situation du patient indique que celui-ci a besoin d'aide, alors l'agent de localisation ordonne à un agent aidant de migrer vers le patient pour lui apporter assistance. Cet ordre se fait via des messages spécifiques que nous traitons au chapitre 6.

Schéma des Rôles
Localisation
Description
L'agent doit connaître en permanence où se trouve le patient et il doit être capable de demander à l'agent aidant de migrer vers le patient en cas de besoin.
Protocole et activité
Recevoir (références(Patient)), Demander (migrerVers(patient) afficher (dans(tableau), références (patient))
Permissions
Lire : message reçus des agents résidents Écrire : message pour les agents aidants
Responsabilité
Vivacité: (Recevoir (références (patient)). afficher (références (patients))|| (Besoin(aide).demander (migrerVers(patient)))$^{\omega}$ **Sûreté:** Adresse (AL[1])/= NULL ^Adresse (AgentAidant)/= NULL^références (patient)/=NULL

Figure 8: Rôle de Localisation

1. AL : Agent de Localisation

La figure 8 illustre les aspects du rôle que joue l'agent de localisation. Pour faciliter la compréhension de ce rôle, nous expliquons les rubriques suivantes :

- Protocoles et activités :

Ce sont les tâches élémentaires que l'agent exécute dans la mise en œuvre de son rôle. Il reçoit les références du patient, *Recevoir (références(Patient))*, affiche ces références dans un tableau, *afficher (dans(tableau),références (patient))* et demande à un agent aidant de migrer vers le patient en cas de besoin, *demander (migrerVers(patient))*

- Permissions :

Concernant les permissions, l'agent de localisation à le droit d'accéder aux références du patient qui sont transmises sous forme d'un objet logiciel java

(« location »), de même, il a le droit de construire un message en y insérant les références du patient et l'ordre à exécuter par l'agent aidant.

- Responsabilités :

Dans ses activités qui consistent à accomplir les tâches mentionnées dans les protocoles et activités, si l'agent reçoit les références du patient, il les affiche dans le tableau de son interface graphique et demande à l'agent aidant de migrer vers le patient, ce qui se traduit formellement par :

(Recevoir(références (patient)).afficher

(références(patients))||(Besoin(aide).demander (migrerVers(patient))) $^{\omega}$

Pour le bon déroulement des activités de l'Agent de localisation, il faudrait qu'il soit atteignable par les agents résidents, *références(AL)/= NULL* et que les références du patient soient connues, *références(Patient)/= NULL* et bien entendu l'agent aidant doit être atteignable, *références(Agent Aidant)/= NULL.*

Sous-rôle 2 : détection de présence

Le rôle de détection du patient devant un électroménager ou à l'intérieur d'une pièce est assigné à un agent qui est hébergé de façon permanente soit dans un appareil électronique ou électroménager de l'habitat soit dans des modules informatiques connectés à ce type d'appareil en vue d'envoyer des commandes électriques. Nous appelons ces types d'agents, des agents résidents. L'agent de localisation est aussi un agent résident. Ce rôle fait aussi partie de la localisation. Nous pouvons choisir un agent résident pour chaque pièce de l'habitat, dans ce cas, cet agent se charge de détecter aussi bien les entrées du patient dans la pièce que la présence de celui-ci devant un appareil. Quand nous choisissons d'implanter un agent par appareil de l'habitat, un de ces agents est choisi pour la détection de l'entrée du patient dans la pièce où il se trouve. La figure 9 illustre ce rôle.

Schéma des Rôles
Détection de présence
Description
Signaler la présence du patient devant un appareil ou à l'intérieur d'une pièce et préciser s'il a besoin d'aide ou pas
Protocole et activité
Transmettre (références (patient), AL) Devant (appareil, patient), Entrée (patient,piece).
Permission: Lire : lecture des références du patient écrire : références du patient et de message à l'agent de localisation
Responsabilité **Vivacité:** (Devant (appareil, patient). Transmettre (références (patient), AL))$^{\omega}$ **Sûreté:** références(Patient)/= NULL^ références(AL)/=NULL

Figure 9: Détection de la présence du patient

La rubrique permission de la figure 9 illustre les droits d'écriture et de lecture qu'un agent résident a sur les ressources, en l'occurrence, il crée un objet logiciel qui représente les références du patient en lisant l'adresse de ce dernier.

Pour assumer son rôle, l'agent résident observe en permanence le *Contexte.* S'il constate la présence du patient devant un appareil, il transmet les références du patient à l'agent de localisation, ce qui se traduit formellement par, *(Devant (appareil, patient). Transmettre (références (patient), AL))*$^{\omega}$.

5.1.2 Autorité et Service d'annuaire

La figure 10 illustre le rôle de l'AMS. L'AMS représente l'autorité d'une plateforme JADE. Il autorise les agents à joindre la plateforme. Pour initialiser un agent, nous lui donnons un identifiant lors de son démarrage. Il faut éviter de donner à un agent, un identifiant

qui est déjà utilisé par tout autre agent entrain de s'exécuter sur la plateforme. Quand un agent veut accéder à une plateforme en utilisant un identifiant existant, l'AMS lui refuse l'accès et l'agent est détruit.

Comme dans la majorité des villes modernes, il existe un annuaire téléphonique que chacun peut consulter en vue de trouver le numéro de téléphone d'une tierce personne. Dans notre système, considéré comme une organisation, nous avons décidé d'utiliser ce service pour répertorier les agents du système. Chaque fois qu'un agent est crée et démarré, il s'enregistre auprès de l'agent page blanche, à savoir l'AMS. C'est un agent qui existe déjà dans l'infrastructure JADE, nous l'utilisons tout simplement. Cet agent contient l'identification (AID « Agent IDentifier ») de tous les agents. Cet AID est associé à la description de chaque agent du système.

Schéma des Rôles
Autorité et services d'annuaire
Description
Autoriser les agents à joindre la plateforme. Enregistrer tous les agents de la plateforme avec leur description dès leur initialisation dans la plateforme
Protocole et activité
Enregistrement(Agent) Détruire(Agent)
Permission
Autoriser les agents à accéder à la plateforme Arrêter ou/et détruire un agent s'il y a lieu
Responsabilité
Vivacité: (enregistrer(Agent)|| détruire(Agent))$^{\omega}$ **Sûreté:** Référence(Agent) /= Null

Figure 10: Rôle d'annuaire

5.1.3 Rôle du service des pages jaunes

L'Agent page jaune est un agent qui existe déjà dans l'infrastructure JADE. Il maintient un service d'annuaire pour tous les services offerts par les agents du système. Quand un agent veut offrir un service, il enregistre ce service auprès de l'agent page jaune. Un agent offrant déjà un service peut aussi y mettre un terme quand il le veut, c'est le désenregistrement du service.

Schéma des Rôles
Pages jaunes
Description
Fournir les adresses des services disponibles et leur description
Protocole et activité
Recevoir (demande (Info,service)), informer(agent,service) Demander (Adresse, Service), Enregistrer (agent, service), Desenregistrer(service,agent)
Permissions
Lire : Lire et transmettre la description d'un service écrire : Écrire la description d'un service offert
Responsabilité
Vivacité: ((recevoir (demande (Info, service)).donner (Info,service). || ((Enregistrer (service,agent).Desenregistrer(service, agent)) $^{\omega}$) **Sûreté:** Références (service)/=NULL

Figure 11: Rôle des pages jaunes

L'agent pages jaunes dont le rôle illustré à la figure 11, a la responsabilité de recevoir les informations des services qu'il enregistre pour ensuite informer les agents qui souhaitent utiliser les mêmes services. Cette responsabilité se traduit formellement par la formule suivante :

((recevoir (demande (Info, service)).donner (Info,service).||

((Enregistrer(service,agent).desenregistrer(service,agent)))$^{\omega}$

5.1.4 Rôle d'assistance aux personnes

Le rôle d'assistance au patient est joué par l'agent aidant. C'est un agent qui intègre la mobilité. Il reçoit ses ordres de l'agent de localisation. Quand l'agent de localisation est informé que le patient a besoin d'aide, il envoie un message à l'agent aidant. Ce message contient un ordre demandant à l'agent aidant de migrer vers le patient en vue d'exécuter l'aide demandée. La conception de l'aide, sort du cadre de notre travail, cependant, l'agent est conçu pour l'intégrer, raison pour laquelle nous prévoyons l'aide dans le schéma des rôles.

Schéma des Rôles
Assistance au patient
Description
Se déplacer à l'endroit où se trouve le patient pour l'assister, en cas de besoin
Protocole et activité
Recevoir (message(MoveAction)), MigrerVers(patient), Aider(patient)
Permissions
Lire : lecture des messages de l'agent de Localisation Écrire : indéterminée
Responsabilité
Vivacité: (recevoir (message(MoveAction)). migrerVers(patient).aider(patient))$^{\omega}$ **Sûreté**: Références (Patient)/= NULL), Références (AgentAidant)/= NULL

Figure 12: Assistance au patient

La figure 12 illustre dans sa rubrique permission que l'agent aidant a le droit de lire les messages provenant de l'agent de localisation. Aucun autre message provenant des autres agents ne peut être lu hormis ceux de l'AMS et de l'agent des pages jaunes.

Dans cette figure, à savoir la figure 12, la responsabilité de l'agent est de recevoir un ordre de l'agent de localisation et de migrer vers le patient, ce qui se traduit formellement par: *((recevoir (demande (Info, service)).donner (Info,service).*

|| ((Enregistrer (service,agent).desenregistrer(service,agent))) $^{\omega}$

5.2 Le modèle des agents

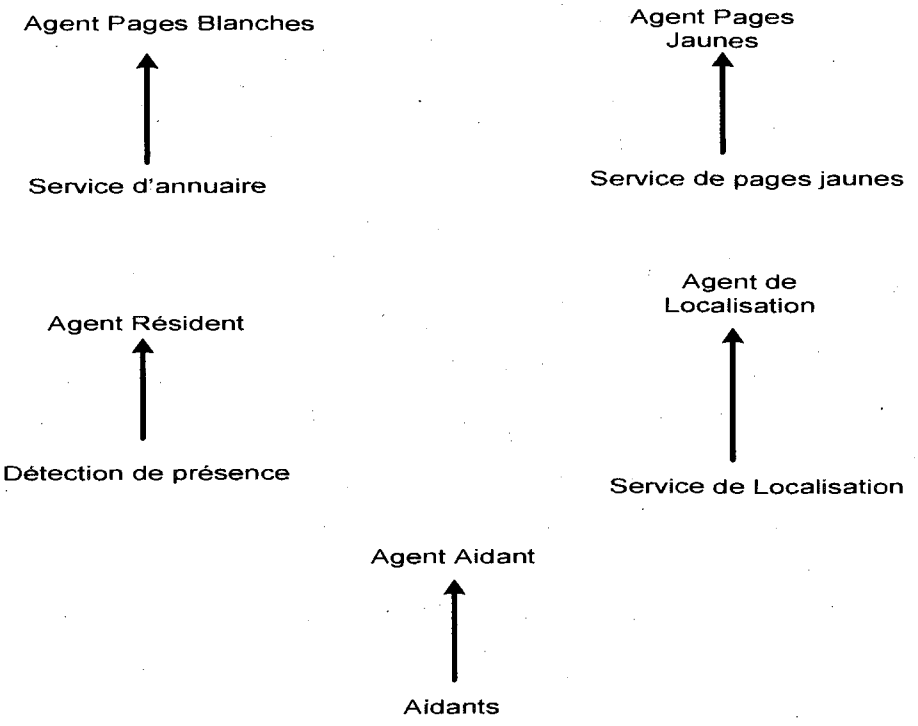


Figure 13: Modèle des agents

La figure 14 nous montre le modèle des agents. Il représente l'ensemble des agents qui existent dans le système. Il vient mettre en œuvre l'assignation des rôles par la création des types d'agents définis à la section 5.1. C'est une vue d'ensemble des agents du

système. Il faut noter que le rôle de détection de présence inclus la détection devant un appareil aussi bien que la détection du patient dans une pièce de l'habitat. Bien que la détection de présence du patient soit une partie de la localisation, nous l'avons séparé pour des raisons que nous avons mentionnées à la section 5.1.1.

5.3 Migration des agents

L'objectif principal de ce travail est de trouver un algorithme de migration qui doit se baser sur la localisation vu que le patient doit être localisé avant de pouvoir le rejoindre et l'assister, si cela s'avère nécessaire. Dans tous les cas, il faut savoir où se trouve le patient avant de faire migrer l'agent. C'est la raison pour laquelle nous nous basons sur la localisation du patient dans notre algorithme. Ainsi la localisation nécessite des interactions entre les agents résidents et l'agent de localisation.

Dans notre algorithme, nous jugeons nécessaire d'utiliser la migration, car l'aide dont le patient à besoin n'est pas nécessaire liée à la pièce où il se trouve. Par exemple à 16 h 00, le patient doit prendre son médicament qui se trouve dans le réfrigérateur. Cependant, il se trouve au salon ou dans sa chambre. Un agent aidant sera chargé de lui rappeler et de le guider s'il le faut pour qu'il puisse prendre sa médication.

5.3.1 Scénario utilisé pour prouver le concept

Notre scénario se base sur le fait que nous sommes dans un environnement diffus. Ainsi, nous prenons pleinement avantage d'avoir des objets logiciels intégrés dans les différents systèmes électroniques de l'habitat pour localiser le patient dans les différentes pièces et, plus précisément, devant les appareils électroménagers et électroniques.

Dans chaque pièce, les appareils sont munis d'agents résidents qui vont détecter la présence du patient en fonction du contexte, soit devant l'électroménager (le réfrigérateur, par exemple), soit devant l'étagère de la cuisine (pour prendre le petit déjeuner ou les médicaments), soit devant les appareils électroniques, comme la

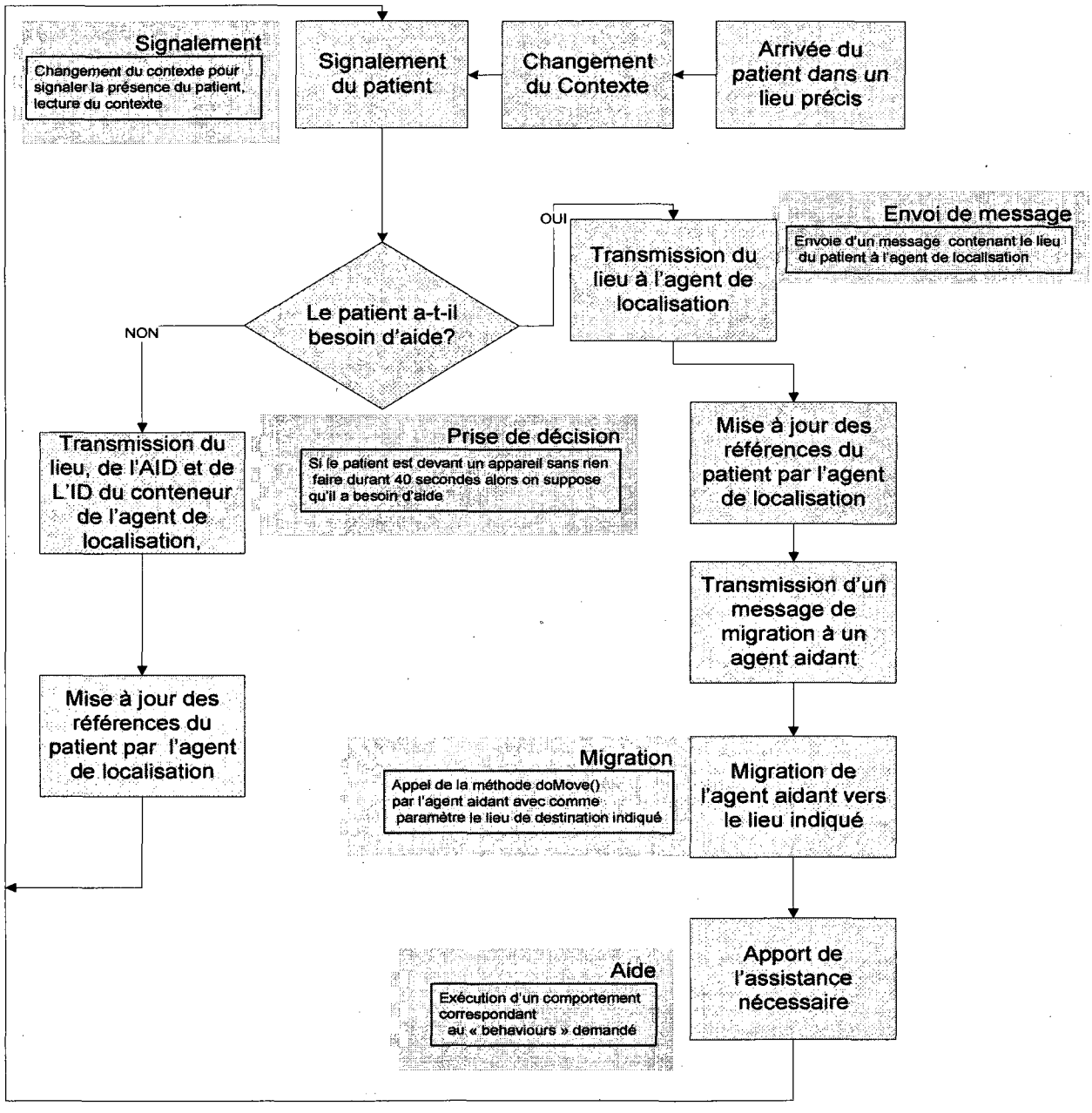


Figure 14: Algorithme de notre système

télévision. Une fois la présence du patient détectée, l'agent résident peut, en fonction du temps que passe le patient devant l'appareil, déterminer s'il a besoin d'aide. Pour nos tests, si le patient reste devant l'appareil sans agir durant 40 secondes, nous supposons qu'il a besoin d'aide. Le message qu'envoie l'agent résident à l'agent de localisation contient les références du patient (lieu, conteneur de l'agent résident, heure) et un champ pour indiquer s'il faut apporter de l'aide au patient ou non. Dans tous les cas, les références du patient sont affichées en temps réel. S'il s'avère que le patient a besoin d'aide, alors l'agent de localisation envoie un message à l'agent aidant pour qu'il migre vers le patient.

L'algorithme de la figure 14 est basé sur le scénario du paragraphe précédent. Quand le patient arrive où les capteurs sont installés, par exemple devant la cuisinière, il est détecté, ce qui entraîne une modification du *Contexte* (Objet Logiciel). L'agent résident qui observe le *Contexte* en permanence se rend compte de la présence du patient.

Si le patient est devant l'appareil pendant plus de quarante secondes alors l'agent résident envoie un message de demande d'aide à l'agent de localisation. Quand l'agent de localisation reçoit un message provenant de l'agent résident, il ouvre le contenu et affiche les références du patient dans un tableau de son interface graphique. Si le champ « needHelp » du message envoyé par l'agent résident est marqué « oui », cela signifie que le patient a besoin d'aide, ainsi l'agent de localisation envoie un message contenant l'objet «MoveAction» qui à son tour contient l'adresse à laquelle l'agent aidant doit se rendre pour apporter l'aide nécessaire au patient. Dans le cas où le temps passé par le patient devant l'appareil est inférieur à quarante secondes, l'agent résident envoie juste un message d'information à l'agent de localisation. Le contenu de ce message est le nom de la pièce, l'identité du conteneur de l'agent résident, la date de l'événement, etc.

Pour des questions d'efficacité de communications réseaux, nous évitons autant que possible la diffusion de message, les agents résidents s'enregistrent dès leur installation auprès de l'agent pages jaunes pour être joignables dynamiquement puisque leur adresse est connue à l'avance.

5.4.2 Modèle d'interaction et mise en œuvre

Notre système étant composé d'agents, ces derniers interagissent essentiellement par échanges de messages pour mettre en œuvre l'algorithme général du système. Les interactions dans le système sont bien définies et se basent sur un modèle, le modèle d'interaction.

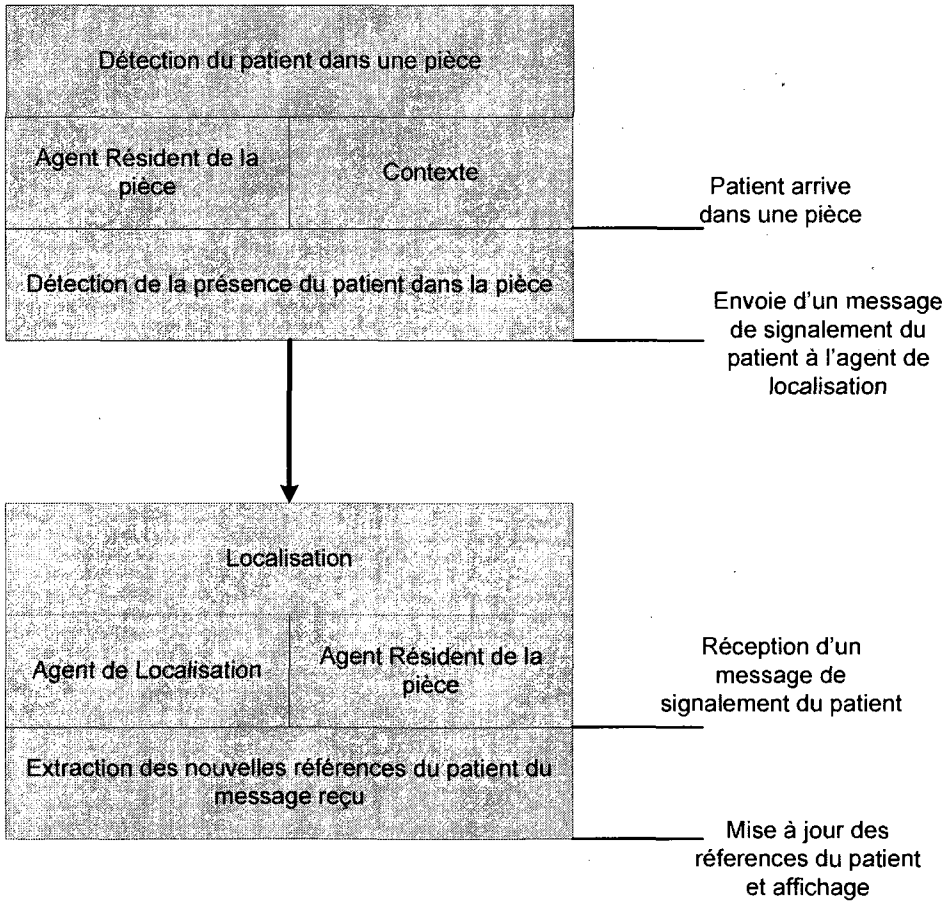


Figure 15: Protocole d'interaction pour la détection de présence du patient

Comme dans une organisation, il y a des protocoles définis pour demander et recevoir un service ; ainsi, le modèle d'interaction présenté ici définit les protocoles qui doivent être suivis pour obtenir un service tel que la localisation. Afin d'illustrer le modèle d'interaction choisi, nous en présentons quelques-unes des interactions entre les agents. Bien entendu, d'autres interactions entre les agents du système sont possibles. La figure 15 illustre l'interaction entre un agent résident et le *Contexte*, l'objet logiciel intégré aux agents résident.

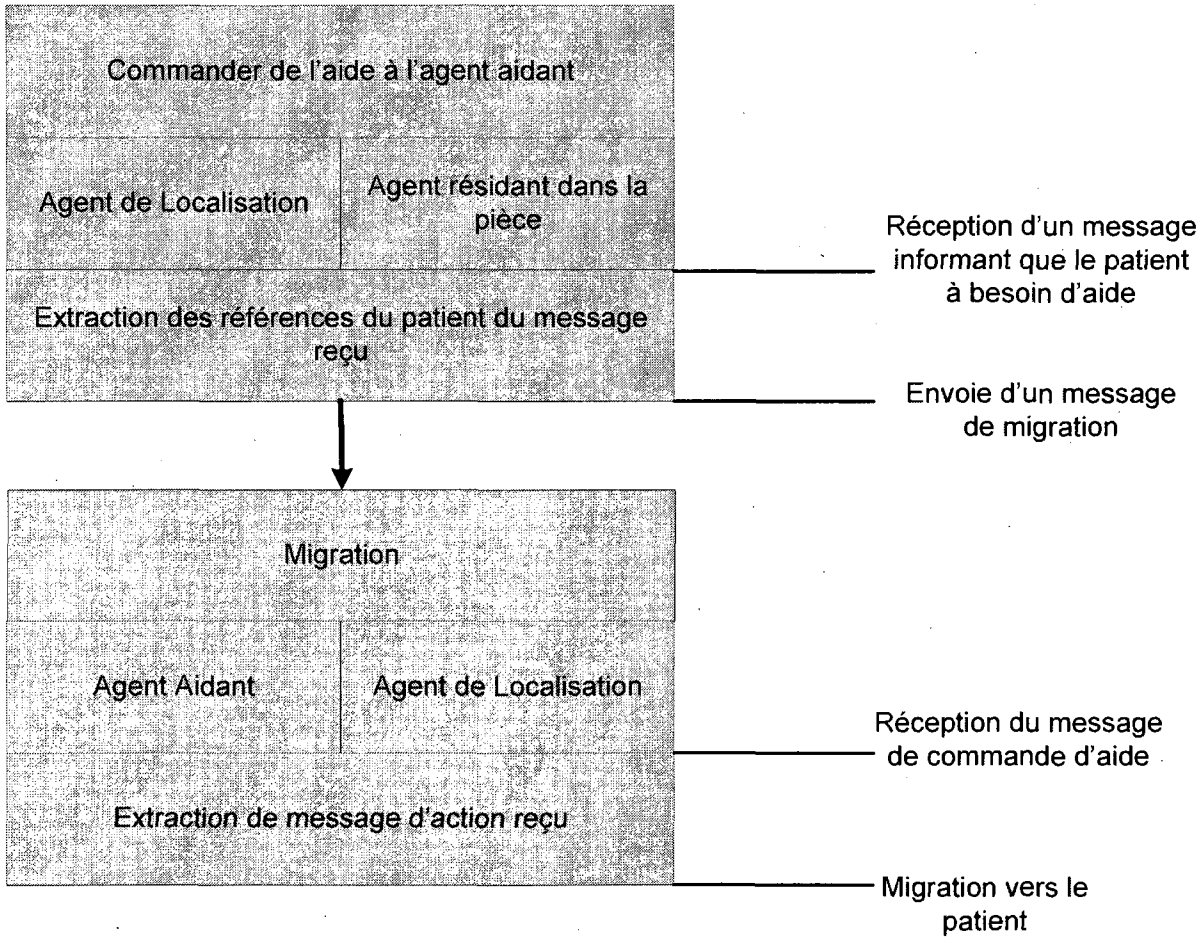


Figure 16: Protocole d'interaction pour la migration automatique vers le patient

Il faut noter qu'il y a autant d'agents résidents qu'il y a d'appareils dans l'habitat. Quand le patient arrive devant un appareil, il crée une pression sur le tapis posé devant

l'appareil. Les capteurs qui sont insérés dans le tapis génèrent un signal électrique sous la pression créée par le poids du patient, qui renseigne le Contexte. Il se produit alors un changement de l'état du Contexte. En observant le Contexte, l'agent résident détecte la présence du patient. Il envoie donc un message d'information à l'agent de localisation. La figure 16 illustre la demande de migration ordonnée par l'agent de localisation à l'agent aidant. Quand l'agent de localisation reçoit un message provenant d'un agent résident, il extrait le message pour s'enquérir des informations. Si les informations indiquent que le patient a besoin d'aide, il envoie un message à l'agent aidant pour que ce dernier migre vers le patient pour l'assister.

5.5 Proposition d'un système de localisation

La localisation du patient dans l'habitat peut être difficile quand le patient ne se trouve pas à un endroit précis où les capteurs sont placés. Hors du lit, par exemple, il est difficile de savoir si le patient a quitté sa chambre.

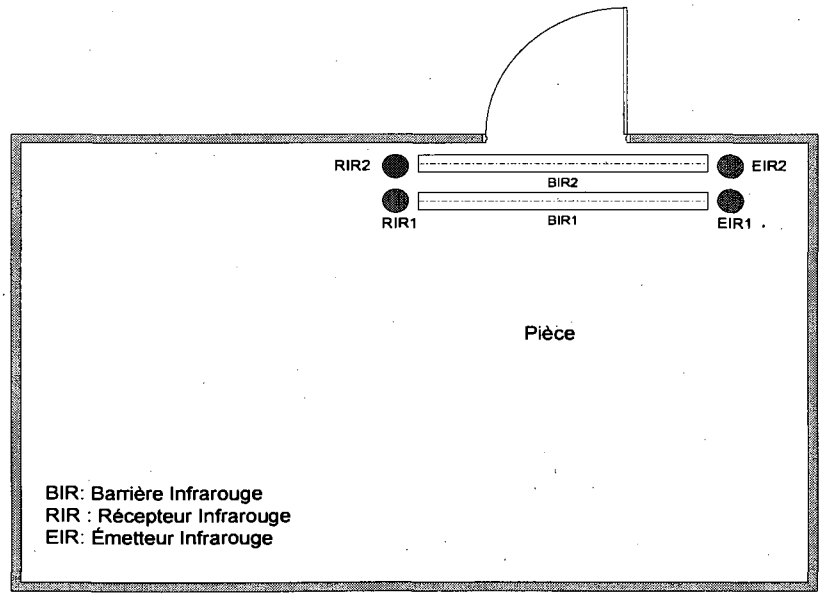


Figure 17: Positions des barrières infrarouges

Pour rendre la localisation du patient plus efficace, nous proposons donc le système de localisation suivant pour palier ces déficiences. Il est constitué de deux parties : une partie matérielle constituée par deux barrières infrarouges comme illustrée à la figure 17 et une partie logicielle implantée au travers d'une machine à états finis.

Une barrière infrarouge est formée par deux dispositifs : un dispositif émetteur infrarouge et un autre récepteur infrarouge. Quand une personne veut aller dans une pièce de l'habitat ou en sortir, elle traverse les deux barrières infrarouges. En les traversant, elle les interrompt, car la lumière infrarouge ne traverse pas le corps humain. L'interruption momentanée d'une barrière infrarouge provoque la génération d'une impulsion électrique dont la durée est proportionnelle au temps de son interruption. L'impulsion électrique renseigne le *Contexte*. Par exemple, une entrée dans la chambre occasionne une interruption de la barrière BIR2 ensuite BIR1. Cette interruption provoque la génération d'une impulsion électrique des récepteurs RIR1 et RIR2. La séquence d'apparition des deux impulsions électriques est observée via le *Contexte* et fait évoluer notre automate. Cet automate, illustrée à la figure 18, constitue la deuxième partie notre système.

Il faut noter que nous mettons en place deux barrières infrarouges pour connaître le sens de déplacement du patient. La séquence de coupure des barrières nous indique si le patient entre dans une pièce ou en sort. Avec une seule barrière infrarouge, il nous est impossible de connaître le sens du déplacement du patient.

Notre automate fonctionne de la façon suivante :

Soit **ei1**, l'impulsion électrique générée par eIR1 et **ei2** celle générée par eIR2. Si **ei1** apparaît après **ei2**, alors le patient entre dans la pièce. Dans le cas contraire, le patient sort de la pièce. Il faut noter que l'impulsion ei1 disparaît avant l'apparition de l'impulsion ei2 et inversement si ei2 apparaît en premier.

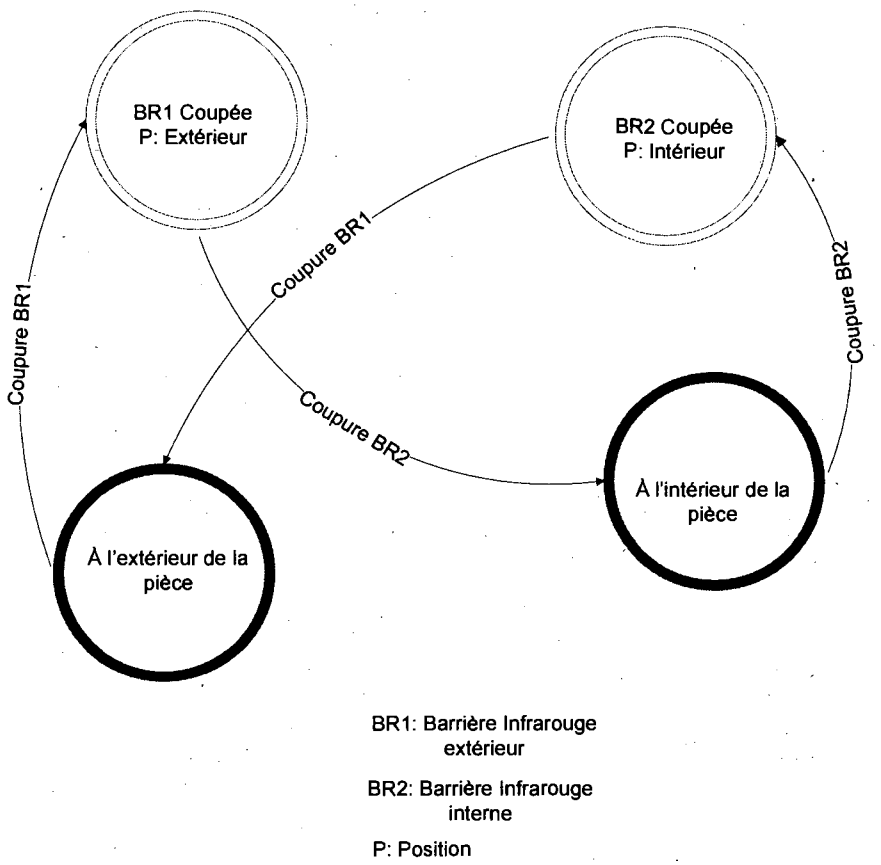


Figure 18: Automate de localisation du patient dans une pièce

Il y a des comportements du patient qui peuvent être ambigus pour notre système. Par exemple, si le patient s'arrête sous le cadre de la porte, il n'est ni dans la pièce, ni hors de la pièce et la détection d'une telle position n'est pas étudiée dans notre système. En conséquence, notre système reste à améliorer. De même, il y a des actions qui sont sans effet, à savoir plusieurs impulsions successives ei1 ou ei2 que nous ajoutons dans la version de l'automate à implanter au chapitre 7. En plus des actions sans effet, nous ajoutons un état initial en vue de l'implantation de l'automate.

Le choix des barrières infrarouges est important parce que les détecteurs de mouvement qui fonctionnent avec les ondes électromagnétiques posent problème dans le cadre de notre travail. Puisque les ondes électromagnétiques traversent le mur, quelqu'un derrière le mur pourrait déclencher les capteurs alors que cette personne n'est pas dans la pièce. C'est une des raisons pour laquelle les points d'accès sans fil ne sont pas utilisés pour la détection de présence dans une pièce.

L'installation de notre système de détection du patient à l'intérieur d'une pièce de l'habitat nécessite certaines précisions. Concernant les pièces ayant plusieurs entrées, toutes ces entrées doivent être munies de détecteurs afin de rendre les capteurs opérationnels. Les détecteurs internes ou externes de ces pièces sont connectés en «OU logique». Pour savoir si le patient est hors de l'habitat, les entrées principales sont équipées de détecteurs dont les signaux sont traités séparément.

5.6 Le modèle d'accointance

Comme il est décrit dans la présentation de la méthodologie Gaia, le modèle d'accointance est un graphe orienté dont les nœuds représentent les agents et les arcs orientés, les directions dans lesquelles les messages sont transmis. Ce graphe représente aussi la topologie logique d'un réseau de communication déployé par notre système. C'est notre réseau virtuel situé au niveau applicatif. Ce modèle de communication est présenté à la figure 19. Ce schéma représente aussi la visibilité entre les agents de la plateforme. L'agent résident peut envoyer des messages à l'agent de localisation, à l'agent pages blanches et à l'agent pages jaunes. L'agent de localisation, pour sa part, envoie des messages à l'agent pages blanches, à l'agent pages jaunes et à l'agent aidant. Il faut mentionner que l'agent aidant n'attend que les ordres de l'agent de localisation pour agir.

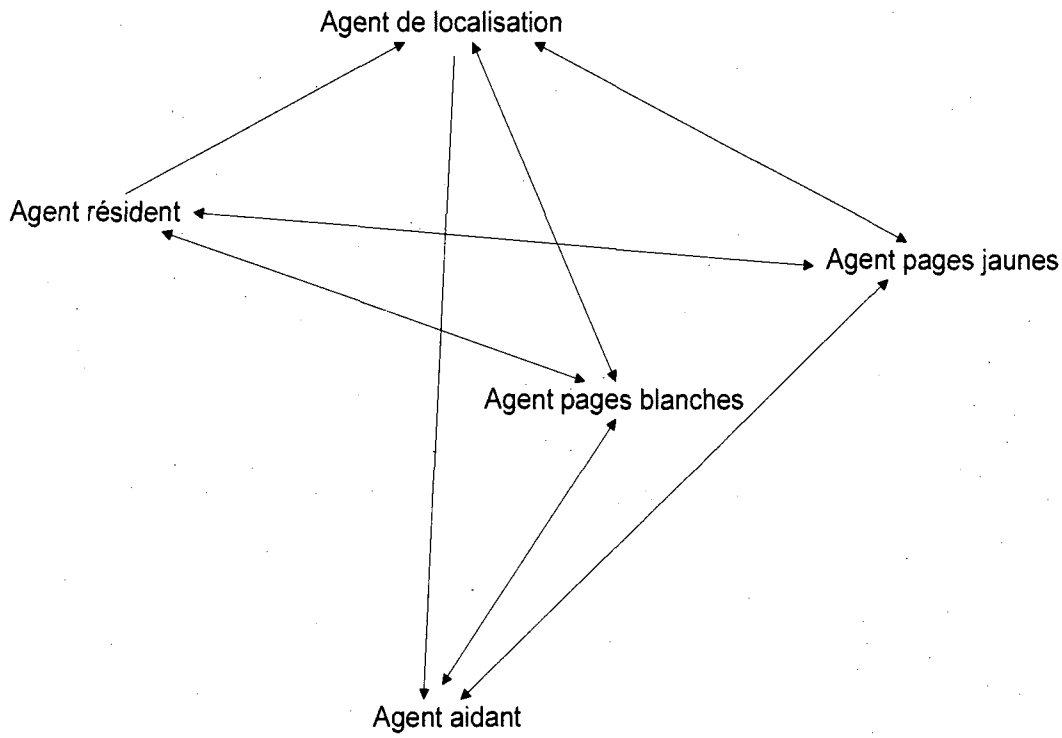


Figure 19: Le modèle communicationnel

5.7 Le modèle des services

Le modèle des services est un tableau qui présente l'ensemble des services offerts dans notre système. Les services dans un système multi-agents sont l'ensemble des méthodes de tous les agents du système. Étant donné que ces méthodes ne sont pas directement accessibles de l'extérieur, l'agent reçoit en entrée des requêtes qui sont des messages contenant des demandes de services spécifiques et les réponses à ces requêtes (sorties) peuvent être soit des messages contenant les réponses soit directement des actions qu'entreprend l'agent. Le tableau 2 présente les services, à savoir le modèle des services qui comprend deux colonnes importantes, les entrées et les sorties. Une autre colonne représente les pré-conditions qui sont les conditions préalables pour que la demande soit valide. La post-condition est une autre colonne qui

présente les conditions nécessaires pour que la demande de service reste toujours valide une fois la demande lancée.

En somme, le modèle des services constitue un document permettant de construire le registre de l'agent pages jaunes, d'une part et de lui intégrer des aspects en conformité avec des règles bien définies par les pré-conditions et les post-conditions, ce qui va nous permettre de tester les pages jaunes efficacement dans la phase des jeux d'essais, d'autre part. Le modèle des services nous donne aussi une vue d'ensemble du but que les agents de notre système atteignent en collaborant tous ensemble. Chaque agent qui possède un service figurant dans le modèle des services s'enregistre auprès des pages jaunes. Pour faciliter la recherche d'un service, l'agent doit, en s'enregistrant, lui donner un nom, «Localisation» en est un exemple. Par exemple, quand un agent demande un service de localisation, il cherche simplement le nom « Localisation » dans le registre des pages jaunes. Il n'a pas besoin de connaître l'identifiant de l'agent (AID) pour utiliser le service, ce qui rend plus flexible l'utilisation.

Services	Entrées	Sorties	Pré-conditions		Post-conditions
Détection du patient	Signalement(Patient)	Envoie (Message, Agent de Localisation))	Présence (Patient)		Présence (Patient)
Localisation	Reçoit(Message, Agent_Résident)	Mise_à_jour(Références(Patient) .affichage(localisation(patient))	Références (Patient)/= Null		Référence(Patient)/=Null
Pages jaunes	demande (adresse ,service), demande (enregistrement(service, agent)) demande(desenregistrement(service,agent)	envoie (adresse (service, demandeur)). notification (enregistrement (service))) notification (desenregistrement(service))	Existe (Service) sauf pour s'enregistrer	Not(Existe (Service), que pour l'enregistrement	Adresse(service) /= Null , tant que l'agent est enregistré
Pages blanches	enregistrement (Adresse (agent))	Notification	Not (Existe (Adresse(Agent))		Existe (Adresse(Agent))
Aide	Demande(migrerVers(AgentAidant),patient)	MigrerVers(Patient)	Besoin(Aide)		Besoin(Aide)

Tableau 3: Le modèle des services

5.8 Conclusion

Le processus de développement basé sur Gaia est léger, il ne demande pas une documentation excessive. Les développeurs peuvent se concentrer sur l'essentiel, c'est-à-dire l'analyse, la conception et le développement, ce qui augmente leur efficacité.

Le choix d'une méthodologie facilite l'analyse si elle est bien adaptée aux besoins. Étant donné que Gaia nous met dans le cadre d'une organisation humaine à laquelle nous sommes déjà familiarisés, il est plus simple pour nous de déterminer les rôles que jouent les individus. Du coup, le processus de développement devient plus intuitif donc plus simple à appliquer.

Chapitre 6

Présentation des aspects de JADE essentiels à la construction de notre système

La présentation de JADE dans le chapitre 2 montre un aspect plus général alors que le présent chapitre aborde des fonctionnalités de façon plus approfondie. Notre système est basé sur des agents et ceux-ci communiquent via des messages basés sur des langages bien définis et des symboles ontologiques. Ainsi, dans ce chapitre, nous présentons les agents de la plateforme JADE et leur rôle. Nous abordons le langage «Agent Communication Language » (ACL) qui constitue un standard utilisé dans les messages échangés entre les agents. Nous terminons par la présentation des ontologies.

6.1 Description des agents de la plateforme JADE

La plateforme JADE, comme illustrée à la figure 20, possède trois grands types d'agents : l'«Agent Management System» (AMS), le «Directory Facilitator» Agent (DF) et les autres types d'agents, notamment, les agents résidents, les agents aidants et l'agent de localisation que nous implantons. L'AMS, autorité de la plateforme qui joue aussi le rôle des pages blanches, est unique dans une plateforme JADE et contient la description de tous les agents de la plateforme. Quand un agent démarre, il s'enregistre auprès de l'AMS avec son AID («Agent IDentifier »). Si l'AID de l'agent existe déjà dans le registre de l'AMS, il n'est pas autorisé à s'exécuter, il est détruit. Il faut noter qu'un agent doit être autorisé par l'AMS pour joindre une plateforme JADE.

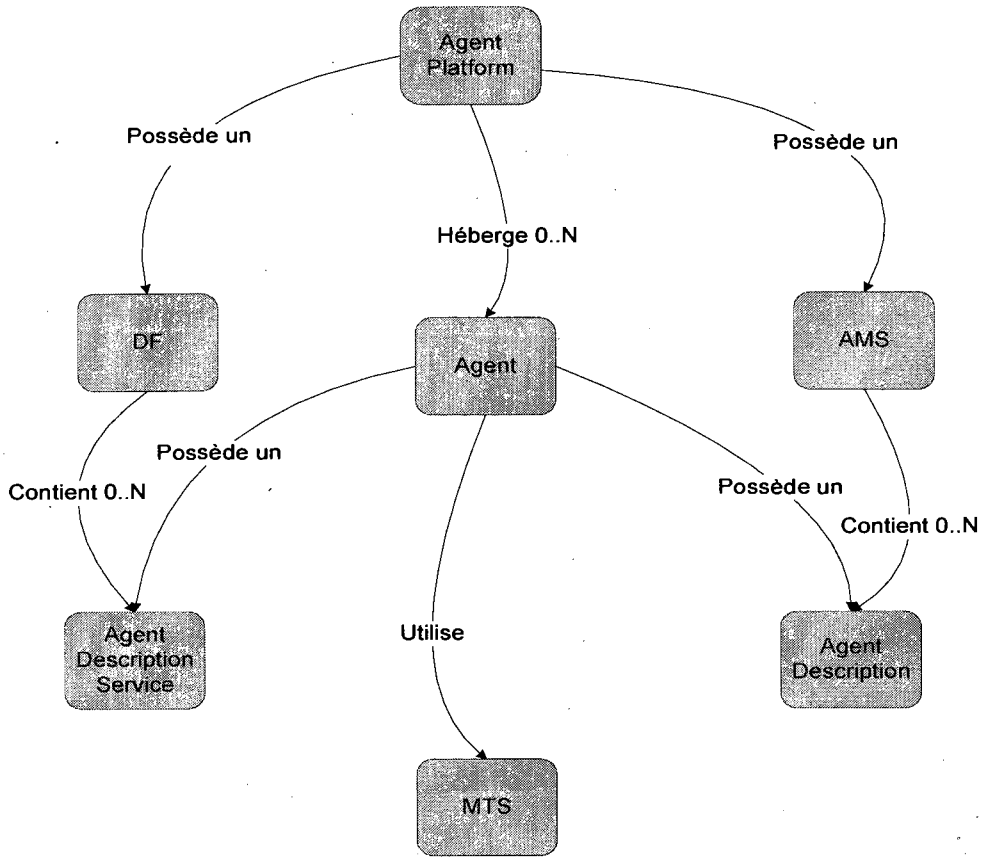


Figure 20: Les agents de la plateforme JADE

Le DF est un agent qui joue le rôle des pages jaunes. Il est aussi unique pour une plateforme JADE, il contient la description de tous les services offerts par les agents de la plateforme. Il faut noter que l'enregistrement des agents auprès de l'AMS est obligatoire, par contre l'enregistrement auprès du DF est facultatif. Cependant, l'enregistrement auprès du DF accroit l'efficacité des interactions entre les agents, car tout agent enregistré peut être rejoint dynamiquement, cela signifie que, pour demander un service, un agent recherche seulement le nom du service dans les pages jaunes et il n'a pas besoin de connaître l'AID de l'agent au préalable.

Tous les agents sans exception communiquent via des messages. Ils utilisent l'infrastructure de transport des messages, le « Message Transport Service » (MTS) implanté dans le « Framework » JADE.

6.2 Les messages « Agent Communication Language »

Les agents communiquent via les messages « Agent Communication Language » (ACL). L'ACL est un format standard auquel tous les agents doivent se conformer pour se comprendre. Ce format standard est définit par la FIPA. Hormis le format ACL, il existe des langages proprement dit qu'utilise les agents de la plateforme JADE, notamment le langage SL (« Semantic Language ») pour les plateformes standards et le langage LEAP pour les plateformes embaquées. Si le langage SL est lisible par l'humain, le LEAP ne l'est pas pour des raisons d'optimisation de l'utilisation des ressources. Les agents utilisent aussi des symboles ontologiques qui sont implantés sous forme d'objet logiciel java. Il existe des ensembles de symboles ontologiques qui existent dans l'infrastructure JADE, notamment « Ontology », qui est une ontologie de laquelle tous les autres sont dérivées et « MobilityOntology », une ontologie qui est utilisée dans la migration des agents. Pour être capable de communiquer entre eux, les agents doivent partager le même format de message, le même langage et la même ontologie. Le format de message, le langage et l'ontologie font partie des propriétés de l'agent que le programmeur implante. L'interprétation et l'écriture d'un message dans un langage sont appelées respectivement encodage et décodage. Les processus de codage et de décodage sont réalisés par des sous-systèmes de l'infrastructure JADE appelé CODEC (CODeur DECodeur). Chaque langage possède un codec spécifique que les agents utilisent, soit pour écrire les messages à envoyer, soit pour interpréter les messages reçus.

Le transport des messages se fait via le réseau si les agents ne sont pas sur la même machine. Pour cela, le MTP, une infrastructure implantée à cet effet, transfert les messages entre deux «Agent Communication Channel» (ACC). L'ACC est le service de transport des messages offert par l'application aux agents pour être capable d'accéder au MTP.

Pour mieux comprendre les communications par messages ACL, nous présentons la structure d'un message ACL à la figure 21.

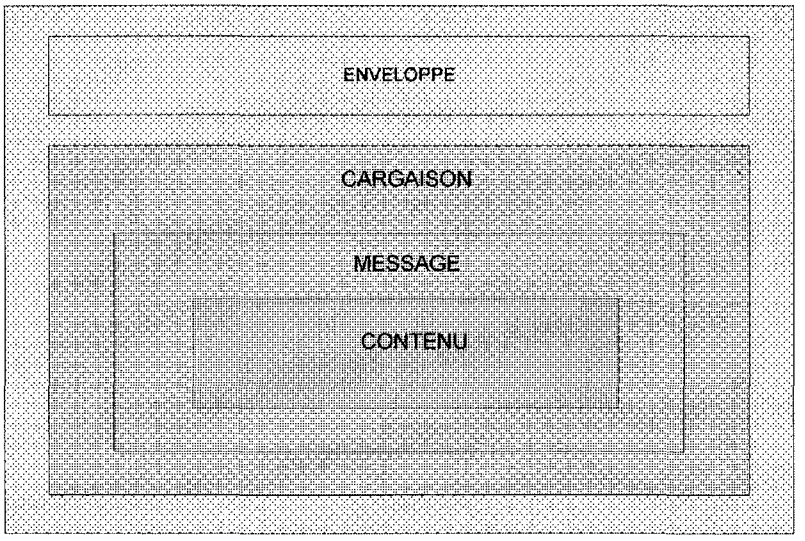


Figure 21: Structure d'un message ACL

Un message ACL est constitué de deux grandes parties : l'enveloppe et la cargaison (« Payload »). L'enveloppe contient les paramètres du protocole permettant d'indexer et d'identifier le contenu de la cargaison comme décrite à la figure 22.

Enveloppe :	contient les paramètres de transport du message
Cargaison:	contient le message encodé dans le langage spécifié par un agent
Message :	paramètres du message
Contenu :	contenu du message

Figure 22: Champs d'un message ACL

La cargaison, quant à elle, est composée de deux parties, le message et le contenu du message. Comme l'enveloppe, le message est un en-tête permettant d'identifier les éléments du contenu du message comme illustré à la figure 23. Il faut noter que dans l'infrastructure JADE, les messages ACL sont implantés par la classe Java *ACLMessage*.

Obligatoire	
To	spécifie le destinataire du message
From	spécifie la source du message
ACL-représentation	représentation ACL, ex XML, String, Bit-Efficient ...
Date	date de création de l'enveloppe
Optionnel	
Payload-length	indique taille en octet de la cargaison « payload »
Payload-encoding	type d'encodage du message ACL, ex us-ascii, utf-8
Received	marque de l'évidence de la réception
Security-object	informations d'encryption et de certificat

Figure 23: En-tête d'un message

6.2.1 Exemples de messages ACL

En prenant notre système en exemple, on suppose que le patient est dans la cuisine et qu'il a besoin d'aide. L'agent résident envoie un message pour informer l'agent de localisation. La cargaison de ce message ACL est présentée à la figure 24. Après avoir pris connaissance du message de besoin d'aide du patient, l'agent de localisation envoie un message d'action à l'agent aidant pour que ce dernier se déplace vers le

patient. La cargaison du message envoyé par l'agent de localisation est présentée à la figure 25.

```
(Inform
        :sender      ( agent-identifier:name agent.resident@domus.usherbrooke.ca:8080)
        :receiver    (agent-identifier;name agent.localisation@domus.usherbrooke.ca:6600)
        :ontology    CommOntology
        :language    FIPA-SL
        :protocol    FIPA-request
        :content
                     (predicate
                              (patientIsHere
                     (location :(:name cuisine@domus.usherbrooke.ca:6600)
                     (AID   : (agentAID agent.resident@domus.usherbrooke.ca:8080)
                     (needhelp : (yes)
                     )
        )
)
```

Figure 24: Exemple d'un message envoyé à l'agent de localisation par un agent résident

```
(Request
        :sender      (agent-identifier:name agent.localisation@domus.usherbrooke.ca:8080)
        :receiver    (agent-identifier :name agent.aidant@domus.usherbrooke.ca:6600)
        :ontology    CommOntology
        :language    FIPA-SL
        : protocol   FIPA-request
        :content
                     (action
                     (moveAction( cuisine@domus.usherbrooke.ca:6600)
                     )
)
```

Figure 25: Exemple d'un message envoyé à l'agent aidant par l'agent de localisation

Pour simplifier, nous omettons les enveloppes dans les présentations des exemples de messages aux figures 24 et 25, car elles ne sont pas nécessaires pour la compréhension. Par ailleurs, nous faisons remarquer que les mots réservés «*inform*» et «*request*» (figures 24 et 25) sont utilisés pour designer respectivement que le message contient soit des informations soit une requête.

6.3 Les ontologies

Lors de la communication, les agents utilisent les symboles ontologiques pour désigner des concepts, des termes et des prédicats. Ces symboles ontologiques sont implantés au travers d'objets logiciels java qui sont classés en plusieurs groupes, ces groupes sont les ontologies que nous pouvons implanter au travers de classes java. Pour créer une ontologie différentes de celles existantes dans l'infrastructure JADE, il faut définir tous les concepts, termes et prédicats à utiliser dans la communication entre les agents et ensuite les déclarer dans une classes java qui représente l'ontologie que tout agent peut utiliser. Les ontologies au niveau du code Java sont des objets transmis via des messages ACL par sérialisation. Pour notre travail, nous avons créé une ontologie, « CommOntology ». Elle regroupe les deux ontologies suivantes, « Ontology » et « MobilityOntologie » en plus d'intégrer d'autres symboles, notamment, un prédicat, «patientIsHere » (figure 26). Ce prédicat permet de transmettre à l'intérieur d'un message que le patient est présent avec les paramètres tels que les références du patient et la date de sa présence. La figure 26 nous montre l'ontologie que nous utilisons.

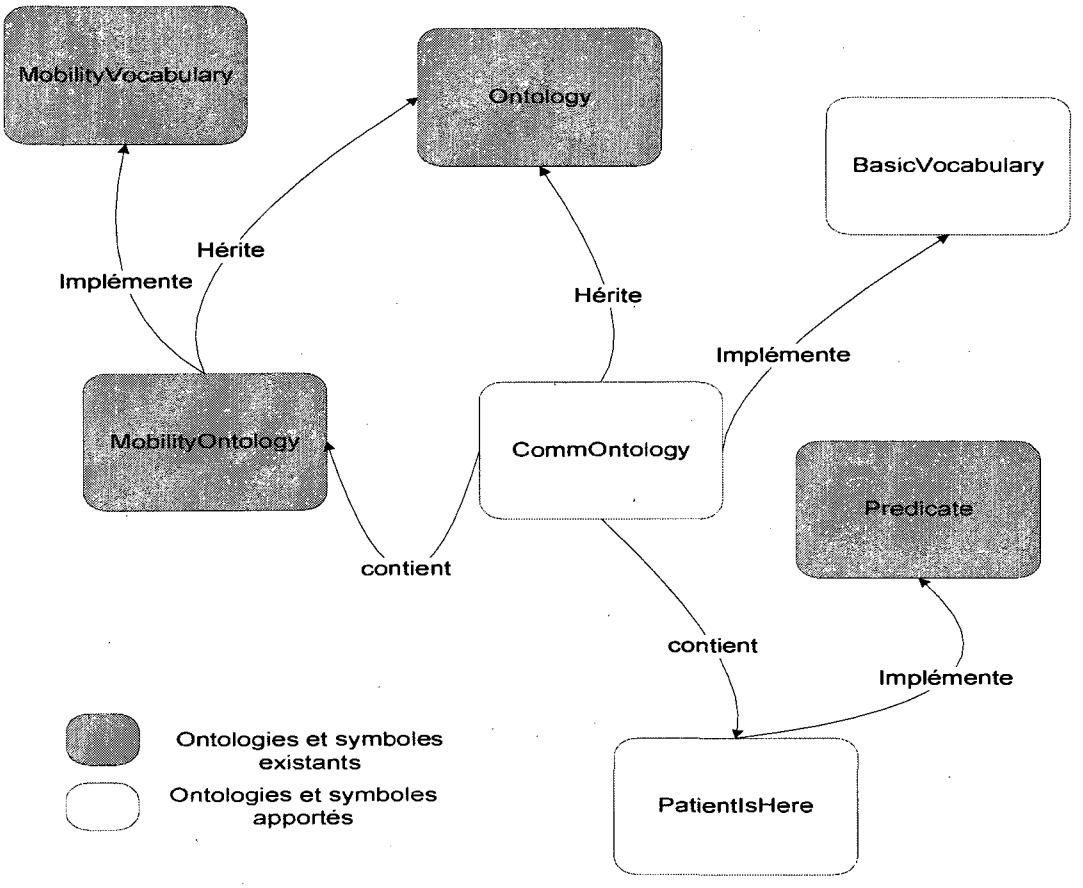


Figure 26: Ontologies utilisées dans notre système

Notre ontologie, à savoir « CommOntology », est constituée, pour la version J2SE, de deux symboles (Terme), un prédicat logique (« patientSignal ») et un concept (« Movement »). Ces deux symboles implémentent respectivement les interfaces *Predicat* et *Concept* de l'API JADE. « BasicVocabulary » est une interface Java qui contient des constantes utilisées dans les interactions avec les agents. Nous identifions les types d'événements liés aux interfaces graphiques des agents mobiles par des nombres entiers constants et les services offerts par des chaînes de caractères.

« MobilityOntology » est une ontologie qui est utilisée pour la migration des agents, elle hérite de la classe Java « Ontology » qui est une ontologie générale définie dans JADE.

« *MobilityVocabulary* » implémente une interface Java où les constantes utilisées dans le processus de migration des agents mobiles sont déclarées.

6.4 Les comportements

JADE offre la possibilité de définir les différentes tâches des agents de façon très modulaire. Il implante une abstraction appelée « behaviours » qui définit le comportement des agents, c'est un objet Java. Quand un agent démarre, il ajoute les comportements («behaviours ») dans la liste des comportements à exécuter en appelant la méthode *addBehaviour(Behaviour b)* avec le comportement à ajouter en paramètre. Une fois l'exécution du comportement terminée, il est retiré de la liste. Il faut rappeler que chaque comportement peut être ajouté ou retirer dynamiquement de l'ensemble des comportements d'un agent. Si nous souhaitons qu'un agent se comporte d'une façon bien définie, alors nous implantons ce comportement via un « behaviours ». À cet effet, nous utilisons une des classes de type « behaviour » qui existe déjà ou nous créons une classe qui va hériter d'un type (« behaviour ») selon nos besoins. Une fois ce comportement créé et instancié, il doit être ajouté à l'ensemble des comportements de l'agent. En exemple, si nous voulons qu'un agent se comporte comme une machine à états finis alors nous définissons une classe de type « FSMBehaviour » à cet effet.

6.5 Conclusion

La compréhension des API de messages tels que *ACLMessage* est une condition indispensable pour réussir un projet basé sur JADE, car la communication, la migration et l'enregistrement des agents se font au moyen de transmissions de messages. Quand les agents utilisent une ontologie qui n'est pas conforme à la structure définie dans JADE, la migration est impossible. Cette impossibilité est due au fait que l'initiation de la

migration se fait par transmission de symboles ontologiques via les messages ACL. Par exemple, une ontologie contenant un symbole défini comme une classe Java dont le constructeur est défini avec un paramètre rend invalide l'ontologie à laquelle il appartient. C'est aussi une des faiblesses de JADE, car le message d'erreur affiché à la suite de cette mauvaise définition du symbole est obscur. On ne peut pas imaginer que l'erreur provient du constructeur paramétré de notre symbole.

Chapitre 7

Notre solution

Ce chapitre est dédié à la présentation de l'implantation de notre système. Il présente la majeure partie des aspects du système, à savoir les agents et les algorithmes sur lesquels se basent leurs comportements. Pour faciliter la lecture, nous remplaçons le code par des algorithmes sous forme de diagrammes.

Nous commençons par la présentation de l'architecture logicielle de notre système qui est une architecture en couche. Ensuite nous abordons les agents que nous avons implantés et enfin nous traitons le sujet de la sécurité de notre système.

Pour plus d'informations sur l'implantation de notre système, nous fournissons le code source sur un disque compact et nous présentons les interfaces graphiques dans les annexes pour une meilleure compréhension du système.

7.1 Architecture logicielle

Notre architecture est une architecture en couches illustrée par la figure 27. Nous présentons les couches en deux grandes parties, à savoir, les couches fournies et les couches que nous apportons. Les couches, de la plus basse à la plus haute, sont les suivantes :

Couches fournies

- La machine virtuelle Java :

  Chaque machine de bureau ou système embarqué intégré à notre plateforme possède une machine virtuelle unique (JVM) qui représente un conteneur. Selon

l'environnement cible, il existe une machine virtuelle adaptée comme c'est les cas pour les machines de bureaux et les systèmes embarqués.

- Les bibliothèques Java :

  Les bibliothèques Java fournissent les APIs de base nécessaires à la construction de notre système. Si nous sommes sur une machine de bureau, les APIs sont fournies par la version standard de Java (J2SE) ; par contre, si nous sommes dans un environnement embarqué, la version J2ME «Connected Device Configuration» (CDC) les fournit. Il faut noter que dans les environnements embarqués, nous utilisons une bibliothèque supplémentaire, à savoir l'« Advance Graphics and User Interface » (AGUI), c'est une version adaptée de «swing» pour les systèmes embarqués, elle est optionnelle.

- Les bibliothèques JADE :

  Elles fournissent les APIs nécessaires à l'implantation des abstractions telles que agent, message, conteneur, CODEC, mobilité, ontologie.

Couches apportées

- Resident :

  Cette bibliothèque fournit les APIs pour les agents qui n'intègrent pas la mobilité, à savoir, l'agent de localisation et les autres agents résidents.

- « Mobility » :

  Située sur la même couche que «resident», cette bibliothèque contient les codes sources et les classes des agents mobiles, à savoir les agents aidants.

- Communication :

Située sur la même couche que «resident», cette bibliothèque fournit les APIs liées à la communication, à savoir, les ontologies que nous avons définies pour notre système.

- Util :

  Située aussi sur la même couche que «resident», cette bibliothèque fournit les classes utilitaires pour tous les agents.

- Client :

  Elle constitue la couche la plus haute de notre architecture. Cette bibliothèque fournit les interfaces graphiques des agents que nous implantons.

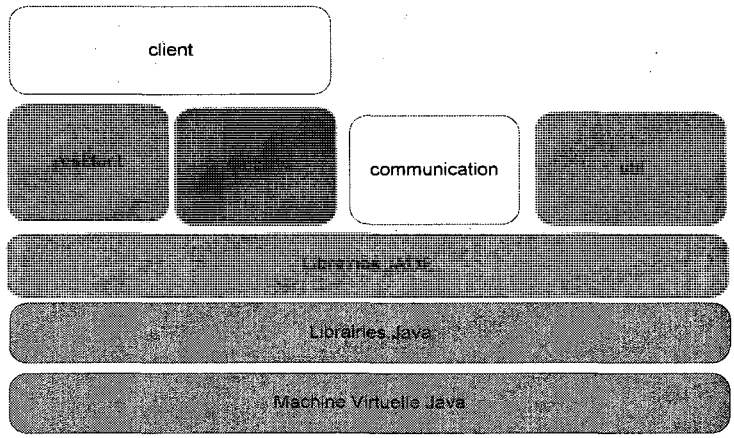


Figure 27: Architecture logicielle de notre système

7.2 Les messages

Nous utilisons les messages de type ACL avec un CODEC en langage SL pour la version J2SE de notre système. Les messages ACL encodés en langage SL sont dans un format lisible par l'humain. Les figures 24 et 25 présentent des exemples de messages en langage SL. Les messages que nous utilisons dans la version J2ME CDC de notre système sont aussi des messages de type ACL cependant ils sont encodés

avec le CODEC LEAP avec lequel les messages sont encodés en octets pour des raisons d'efficacité et de contrainte de ressources, les messages ne sont donc pas dans un format lisible par l'humain.

Pour qu'un agent puisse décoder un message, il faut que le langage du message soit connu. Quand bien même le langage utilisé dans le message est connu, il faut que l'agent qui le reçoit soit configuré pour lire ce message. Tous les agents de notre plateforme utilisent le même langage, le même CODEC et la même ontologie afin de se comprendre. Si nous sommes dans un environnement embarqué, nous utilisons le LEAPCodec et dans le cas contraire, le SLCodec.

7.3 Les types d'agents

Notre système comprend trois types d'agents que nous avons implantés et deux types d'agents qui viennent avec JADE (l'AMS et le DF). Voici une brève description des agents que nous implantons :

- les agents résidents sont tous du même type, à savoir, de la classe « ResidentAgent » ;
- l'agent de localisation, qui possède une instance unique dans le système, représente une instance de la classe « LocalizationAgent» ;
- les agents mobiles, qui sont au nombre de trois, sont tous du même type, soit de la classe « CareGiverAgent ».

7.3.1 Agent résident

Les agents résidents intègrent deux types de comportements cycliques, une machine à états finis qui permet de savoir si le patient est entré dans une pièce ou en est sorti et un autre pour détecter que le patient est devant un appareil.

7.3.1.1 Comportements : Implantation de la machine d'état de localisation dans une pièce

La figure 28 nous montre l'implantation du graphe d'états servant à détecter l'entrée ou la sortie du patient d'une pièce. Dans la conception, par rapport à la section 6.4, nous ajoutons un état supplémentaire qui permet de garder la machine d'états dans un état initial qui est un état de repos lorsqu'aucun signal n'est reçu.

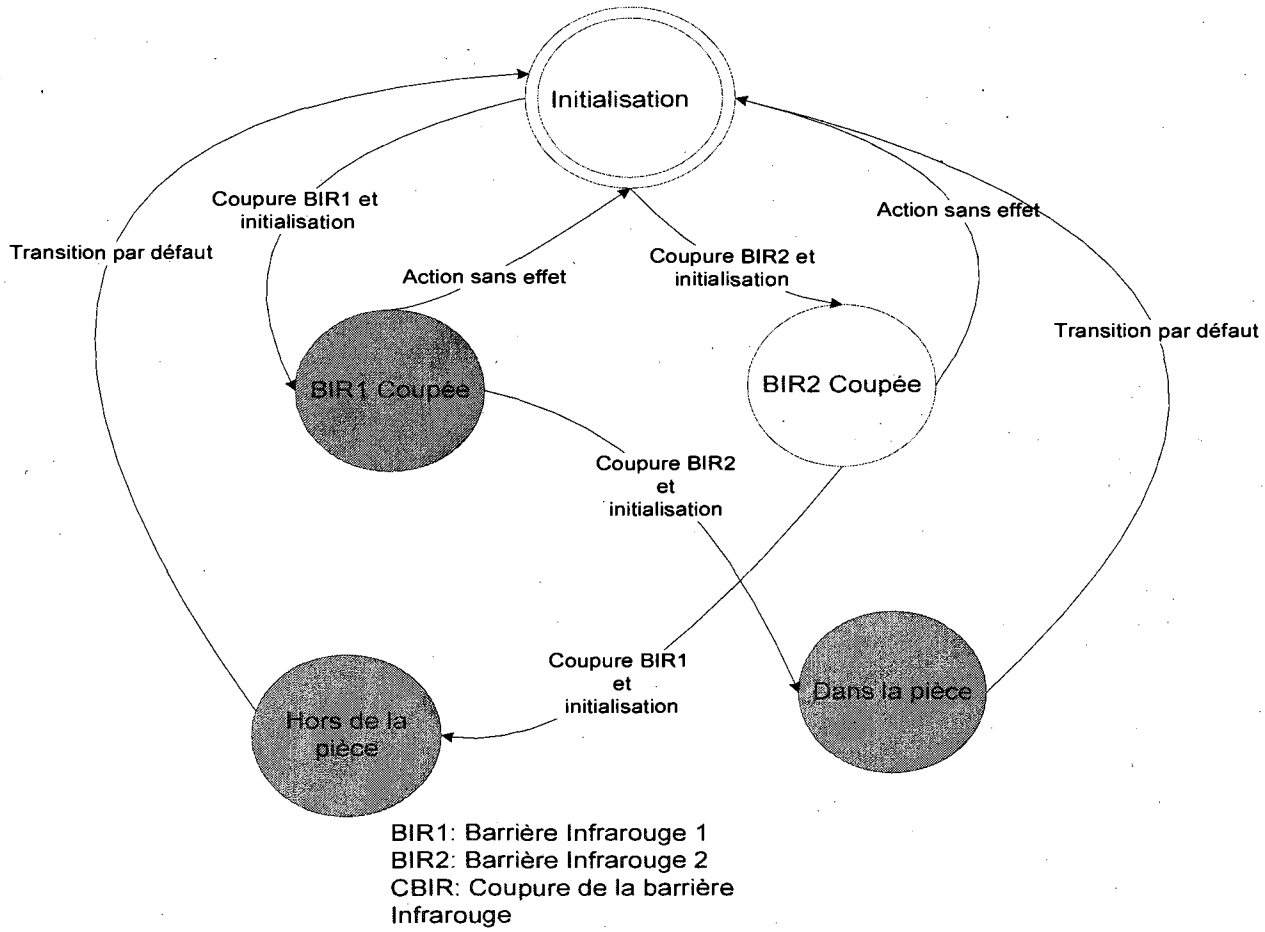


Figure 28: Machine d'états finie implantée pour la détection de l'entrée ou de la sortie d'une pièce de l'habitat

Avec cet état supplémentaire, il nous faut ajouter une variable supplémentaire, *initialisation*, pour marquer les transitions qui passent par l'état initial. Si une transition est générée à partir de l'état initial, la variable *initialisation* prend la valeur « vraie ». Les

transitions par défaut sont celles qui se font automatiquement sans qu'une variable ne change de valeur. Par exemple quand on passe à l'état « hors de la pièce », on retourne à l'état d'initialisation sans condition.

Dans l'état initial, si aucun signal n'est perçu, on ne sait pas où le patient se trouve. Si le patient traverse la barrière pour la première fois, à partir de cet instant, le patient est localisé par les états correspondants.

7.3.1.2 Comportement : implantation de la détection du patient devant un appareil

Pour détecter le patient devant un appareil, nous observons en permanence l'objet logiciel *Contexte*. Cette observation du *Contexte* nous emmène à utiliser une classe qui implante les comportements de type cyclique, à savoir «CyclicBehavior» qui est une classe existant dans JADE et une sous-classe de la classe « Behaviour ». Contrairement aux autres classes de types comportementales dont l'exécution a une fin, celle-ci exécute sa tâche en boucle. Il faut noter que les classes de type comportementales qui implantent des machines à états finie telles que «FSMBehaviour» sont aussi de type cyclique, à savoir, «CyclicBehaviour».

7.3.2 Agent de localisation

Du point de vue de la communication réseau, l'agent de localisation est le point central de notre système. Il est aussi le centre de traitement des informations. Lorsqu'un agent résident détecte le patient devant un appareil, il en informe l'agent de localisation via un message ACL. De même, quand le patient rentre dans une pièce de l'habitat ou en sort, un agent résident informe l'agent de localisation via un message ACL.

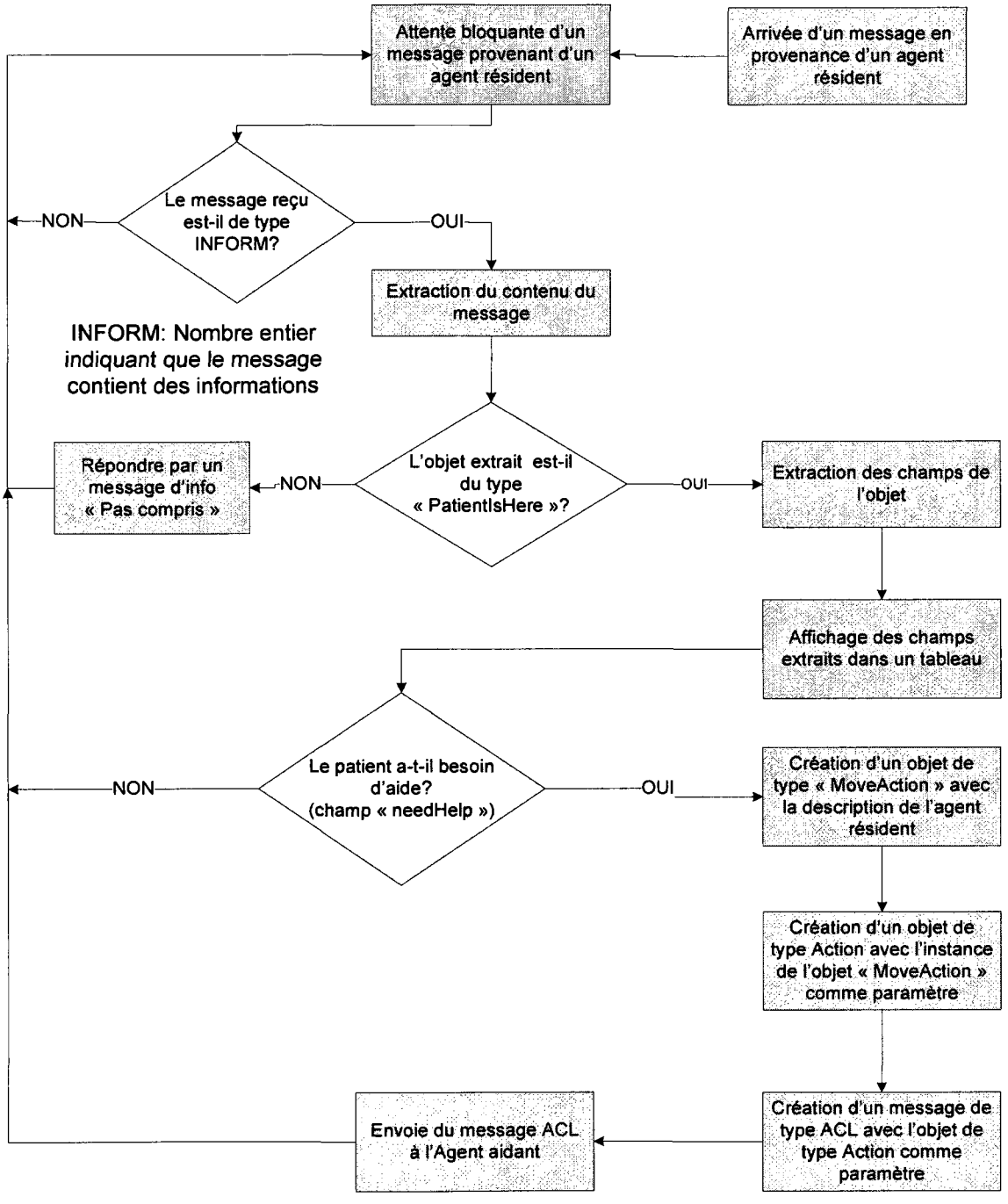


Figure 29: Algorithme implanté dans l'agent de localisation

Afin que l'agent de localisation soit trouvé dynamiquement par les autres agents du système, il s'enregistre auprès du DF. C'est par le nom du service offert que les agents retrouvent son AID chez le DF. Le nom du service enregistré est unique et est formé par une chaîne de caractères constante dont la valeur est «Localisation».

Il faut mentionner que l'agent de localisation intègre une interface graphique. C'est un tableau à deux dimensions dans lequel il affiche l'identifiant du conteneur JADE de l'agent résident qui est la source du message, l'AID de l'agent résident en question, la date de l'événement, le nom de la pièce, l'information selon laquelle le patient a besoin d'aide ou non et le type de mouvement effectué, à savoir, sur place, sortie ou entrée.

L'agent de localisation possède un seul comportement cyclique dans lequel il réalise les activités décrites plus haut. L'algorithme de ces activités est présenté à la figure 29.

7.3.3 Agent aidant

Les agents aidants sont un autre type d'agent que nous implantons dans notre système, à savoir, «CareGiverAgent». Ce type d'agent, à savoir l'agent aidant, est représenté par trois instances. Une première instance a pour rôle de suivre les événements d'entrée ou de sortie du patient d'une pièce donnée de l'habitat, une deuxième instance est chargée de suivre les événements liés aux appareils et enfin la dernière instance est liée à tous les événements concernant le patient. Il faut noter que la troisième instance migre toujours avec l'une ou l'autre des deux premières instances, c'est comme un message d'acquittement de la migration des agents mobiles. En fait, il faut savoir que, si un agent résident détecte la présence du patient, il envoie un message dont le contenu est en relation avec la situation éprouvée par le patient à l'agent de localisation. Si l'événement est un événement d'entrée ou de sortie, l'agent de localisation envoie le message à la première instance de l'agent aidant, dans le cas où l'événement est lié à un appareil, l'agent de localisation contacte la deuxième instance de l'agent aidant si cela s'avère nécessaire.

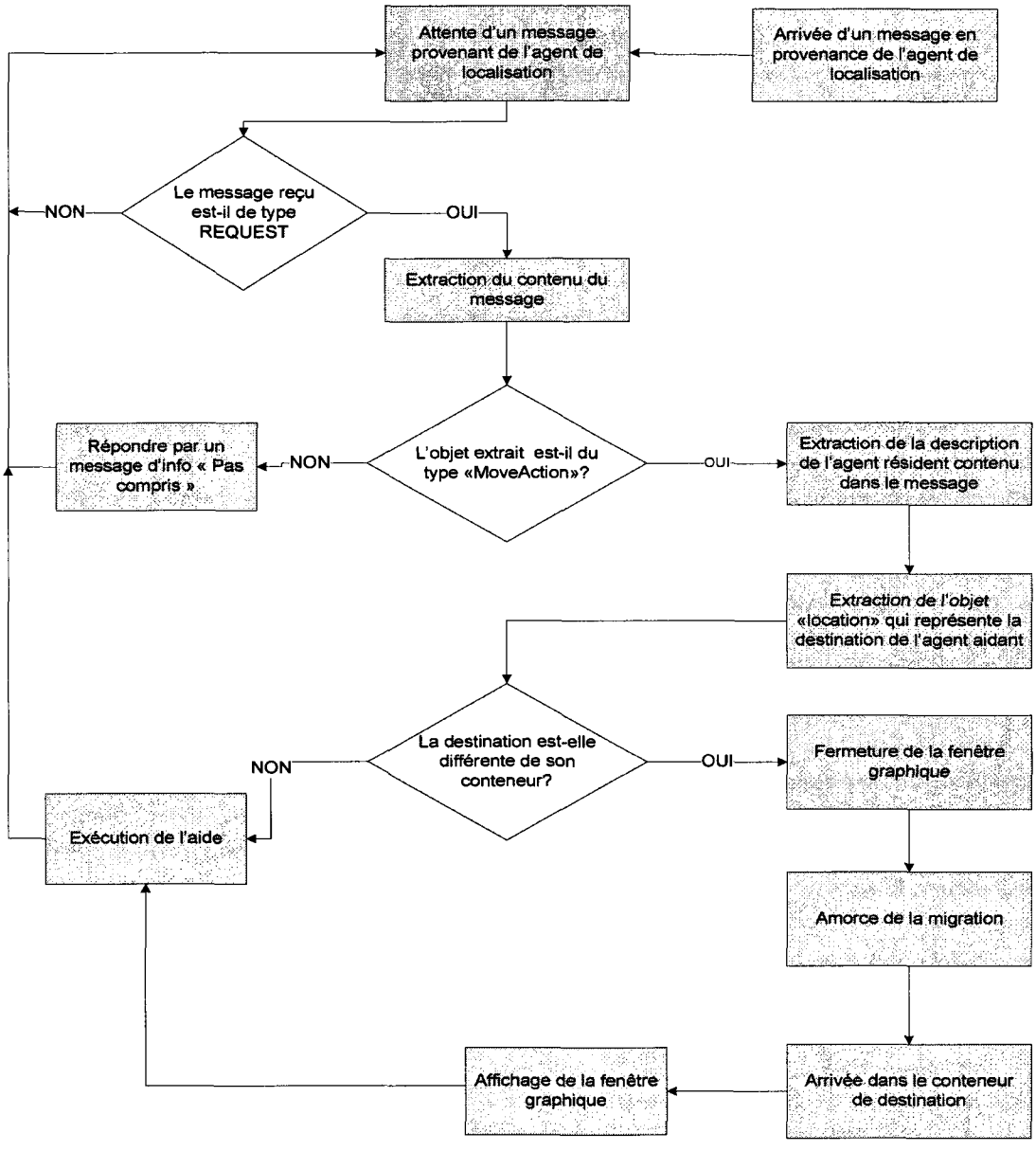


Figure 30: Algorithme de l'agent aidant

Lorsque les trois instances de l'agent aidant sont démarrées, nous pouvons leur attribuer les rôles susmentionnés en les configurant à travers leur interface graphique respective.

Le type «CareGiverAgent» étant le type de base de tous les agents aidants, ceux-ci possèdent tous le même comportement de base. Chaque agent aidant observe en permanence sa boîte de message qui est configurée pour ne recevoir que les messages

de l'agent de localisation. Si un message provient de l'agent de localisation et que ce message est de type «Request» alors le contenu du message est extrait. Si le contenu du message est un objet de type «MoveAction» alors l'objet de type «Location» qui représente l'adresse de destination de l'agent y est extraite. L'agent teste alors si la destination donnée par l'objet de type «Location» est différente de l'adresse où il se trouve. Si c'est le cas, il migre vers la destination mentionnée pour exécuter l'aide, dans le cas contraire, l'agent reste sur place pour porter assistance.

L'algorithme de l'agent aidant est présenté à la figure 33.

7.4 Sécurité

Dans le contexte actuel où l'accès Internet soulève des questions de sécurité informatique, il est difficile de concevoir un logiciel sans tenir compte de cet aspect. Notre système est basé sur la communication, d'où un risque élevé d'intrusion. Une tierce personne peut démarrer un conteneur et joindre notre plateforme pour y lancer un agent intrus en vue de s'emparer des données sensibles en se faisant passer pour un agent légitime.

JADE propose une architecture de sécurité intégrée [46] basée sur le modèle de sécurité de Java [47] que nous n'avons pas implantée. Elle est valable aussi bien pour les machines de bureau que pour les systèmes embarqués. Cette architecture permet de définir les permissions de chaque agent, notamment sur les ressources auxquelles il accède. En plus d'offrir des types d'authentification avec des certificats basés sur « Secure Socket Layer » (SSL), chaque conteneur JADE qui rejoint une plateforme ayant déployé une politique de sécurité exigeant l'authentification doit s'authentifier avant d'être autorisé à rejoindre la plateforme. Chaque administrateur d'une plateforme

a la possibilité de définir sa propre politique de sécurité, par exemple en utilisant des messages ACL encryptés avec SSL. Bien entendu, il existe d'autres moyens pour mettre en œuvre une politique de sécurité en utilisant par exemple un réseau privé virtuel (RPV, «Virtual Private Network, VPN»). JADE étant une infrastructure écrite en Java, on peut aussi utiliser directement des aspects de sécurité de Java pour mettre en place une politique de sécurité, notamment les zones de permission et les objets protégés [47].

Concernant l'accès au réseau de l'habitat intelligent, nous recommandons une architecture de sécurité basée sur un serveur de type « Authentication, Authorization and Accounting » (AAA) que nous avons déployé pour les expérimentations. Un exemple de serveur de type AAA est le serveur « freeradius ». Pour les terminaux sans fils, nous recommandons une architecture « Wireless Protected Access » (WPA) entreprise basée sur une authentification de type EAP/IEEE 802.1x («Extensible Authentication Protocol») dont un schéma de l'architecture est présenté à la figure 31. Il faut noter que la technologie WPA entreprise repose sur un serveur de type AAA, « freeradius » dans notre cas.

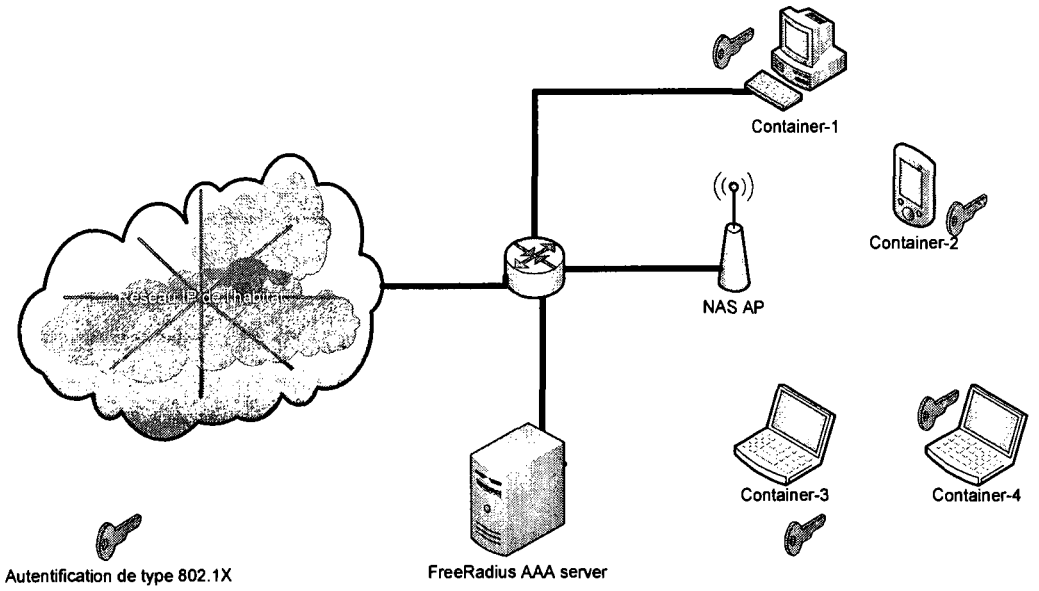


Figure 31: Architecture de sécurité basée sur un serveur de type AAA

L'architecture WPA que nous proposons est constituée d'un réseau filaire et d'un autre réseau sans fil. Les points d'accès sans fils (AP) étendent le réseau filaire en question pour la mobilité des utilisateurs. Les utilisateurs accèdent au réseau filaire via les points d'accès sans fil qu'on appelle « Network Access Server » (NAS). Le cœur de cette architecture est le serveur de type AAA qui authentifie les utilisateurs et les autorise à accéder au réseau. Cette authentification a lieu à deux niveaux. Le premier niveau, c'est avec le NAS, qui va ensuite transférer les informations au serveur AAA. Dans l'architecture WPA, les utilisateurs accèdent au réseau via un tunnel crypté. Pour autoriser une personne à accéder au réseau via cette architecture de sécurité, l'administrateur doit lui créer un compte sur le serveur AAA (« Accounting ») associé à un mot de passe. Lorsque cette personne souhaite se connecter, elle doit activer le protocole IEEE 802.1x sur son interface réseau. C'est ce protocole qui met en œuvre la méthode d'authentification auprès du serveur AAA selon le protocole que va déployer le serveur AAA pour identifier les informations d'authentification. Les points d'accès aussi appelé NAS doivent être configurés avec « WPA entreprise » comme méthode de sécurité. Dans la configuration des NAS, il faut indiquer l'adresse IP du serveur AAA, qui est le serveur d'authentification. Le serveur d'authentification que nous utilisons est «freeradius», une implantation du protocole « Remote Authentication Dial In User Service » (RADIUS) et un logiciel libre. Il fonctionne avec une base de données MySQL contenant les comptes des usagers. L'authentification au niveau du serveur *freeradius* est possible via le protocole EAP. Le serveur *freeradius* possède un module EAP qui nous sert à déployer le protocole EAP. Lorsqu'un utilisateur veut se connecter au réseau, il est d'abord invité à s'authentifier («Authentication») auprès du NAS avec son nom d'utilisateur et son mot de passe. Une fois que ces informations sont entrées par l'utilisateur, le NAS suit le schéma du protocole EAP en vérifiant ces informations auprès

du serveur freeradius, en d'autres termes le NAS authentifie l'utilisateur auprès du serveur freeradius. Si les informations sont en adéquation avec un compte utilisateur enregistré dans la base de données MySQL, alors l'utilisateur est autorisé («Authorization») à se connecter au réseau via un tunnel crypté. Le tunnel crypté est mis en place par le protocole «Tunneled Transport Level Security » (EAP-TTLS). Une fois connecté, l'utilisateur peut faire une requête « Dynamic Host Configuration Protocol » (DHCP) pour obtenir une adresse IP. Le déploiement des architectures de sécurité de JADE et de «WPA entreprise» permettent d'assurer la sécurité à deux niveaux :

- Sécurité d'accès : Toutes les personnes s'authentifient afin d'accéder au réseau via un tunnel crypté.
- Sécurité interne : Une fois les personnes authentifiées via leur machine, elles peuvent démarrer un conteneur JADE, cependant pour connecter le conteneur au conteneur principal qui se trouve dans le réseau, l'usager doit encore s'authentifier. Une fois le conteneur JADE authentifié, les agents communiquent via des messages cryptés par le protocole SSL.

7.5 Conclusion

JADE est une infrastructure logicielle qui offre de nombreux aspects (concepts implantés) aux développeurs et permet une implantation rapide des systèmes multi-agents dû au fait que les composants de base sont déjà présents. Cependant, le fait que nous abordons un autre concept, à savoir celui des agents, une bonne connaissance de Java et une bonne documentation sur l'infrastructure JADE s'avère nécessaire pour implanter un système multi-agents.

Concernant la migration des agents au niveau des plateformes embarquées. La migration de deux agents sur un PDA pose problème, c'est-à-dire, si un agent migre sur un PDA où un autre agent est entrain de s'exécuter, la migration est avortée.

Chapitre 8

Les jeux d'essais

Dans un projet de génie logiciel, les tests font partie intégrante du processus de développement. Les activités de tests commencent en amont pour tester le modèle jusqu'en aval pour tester les livrables. Ici, dans ces jeux d'essais, il s'agira essentiellement des tests en aval.

Nous aurions pu mettre en annexe les jeux d'essais mais vu leurs importances dans le processus, nous estimons que ce chapitre constitue une partie intégrante de notre mémoire. Ces jeux d'essais n'étant pas exhaustifs, nous testons seulement les éléments essentiels de notre système.

Dans ces tests, *Junit (« Java Unit Testing Framework »)* n'est pas à l'ordre du jour, tout simplement parce qu'il a été conçu pour tester les systèmes et les applications conçues avec une approche orientée objets. Comme mentionné dans le chapitre 3, les objets et les agents sont fondamentalement différents. De ce fait, les méthodes de tests sont aussi différentes. À cet effet, nous utilisons une bibliothèque de tests fournie par JADE, JADE Test Suite.

Nous procédons en trois phases, d'abord nous définissons notre stratégie de test pour ensuite aborder l'environnement dans lequel nous exécutons les tests. Finalement, nous définissons les différents tests implantés.

Vu le caractère particulier du test des agents, nous consacrons un point spécial pour expliquer la construction des tests automatisés.

8.1 Stratégie de test

La définition de notre stratégie est la première phase des activités de tests. Elle consiste à tester d'abord les composants les plus importants, à savoir l'infrastructure de communication sans laquelle les agents ne peuvent communiquer, pour ensuite tester le contenu des messages. Finalement, nous testons les agents qui sont les composants de base de notre système. Il faut noter que le test des agents se résume essentiellement au test de leurs comportements. Pour évaluer le degré de portabilité de notre système, nous allons faire les tests susmentionnés dans les environnements Linux, Windows XP et Windows Vista.

8.2 Définition de notre environnement de tests

La définition de notre environnement de tests représente la deuxième phase des activités de tests. Dans cette partie, nous présentons l'infrastructure du réseau, les systèmes d'exploitation ainsi que les machines utilisées.

Comme infrastructure réseau, nous utilisons deux réseaux IP (LAN), un réseau câblé et un réseau sans fil qui sont interconnectés via un routeur sans fil équipé de cinq ports Ethernet commutés. Le réseau câblé est basé sur Ethernet (IEEE 802.3) et le réseau sans fil est un réseau WLAN (IEEE 802.11 b/g).

Rappelons que les systèmes d'exploitation que nous utilisons sont Linux, Windows XP et Windows Vista.

Concernant les machines, nous utilisons trois ordinateurs dont un ordinateur de bureau et deux ordinateurs portables. L'ordinateur de bureau est équipé d'un processeur de type Intel Pentium 3 et une mémoire vive d'une capacité de 512 Mo. Les deux ordinateurs portables sont équipés de processeurs Intel Pentium 4 (*Centrino)* à double cœurs et d'une mémoire vive de capacité de 1024 Mo.

8.3 Définition des essais

Cette étape est la dernière phase des tests où nous définissons les essais à implanter. Nos jeux d'essais sont constitués d'un ensemble de tests présentés au tableau 4. Dans les essais E1 à E12, nous testons l'infrastructure de communication et le contenu des messages pour savoir si la communication de nos agents est bien implantée. Ces tests consistent à lancer tous les agents du système, à savoir, les agents résidents, l'agent de localisation et les agents mobiles. Nous simulons la détection du patient devant les appareils et l'entrée ou la sortie du patient d'une pièce de l'habitat. Cette simulation suscite l'envoi de messages à l'agent de localisation par les agents résidents. Lorsque l'agent de localisation reçoit les messages, selon la circonstance, il donne l'ordre aux agents aidants de migrer vers le patient. De cette façon, nous faisons migrer les agents d'un poste à un autre du réseau local.

Les essais E15, E16 et E17 testent respectivement les comportements des agents lors de leur enregistrement auprès du DF, de l'agent de localisation et de l'agent aidant pour vérifier effectivement que l'agent pages jaunes fait sont travail comme il faut. Les essais E13 à E15 se concentrent sur le comportement des agents résidents à savoir la détection de présence devant un appareil et la détection des mouvements d'entrées ou de sorties du patient d'une pièce de l'habitat. Les résultats de ces comportements peuvent être constatés dans le tableau d'affichage de l'agent de localisation comme nous pouvons le voir à la figure 32. Cette dernière nous donne les informations suivantes :

- AID : AID de l'agent qui envoie le message;
- Container-ID : conteneur dudit agent;
- Place : nom de la pièce où a lieu l'événement;

- Date : date à laquelle a lieu l'événement;
- Assistance : informe l'agent de localisation du besoin d'assistance;
- Déplacement : informe du mouvement d'entrée ou de sortie de la pièce;

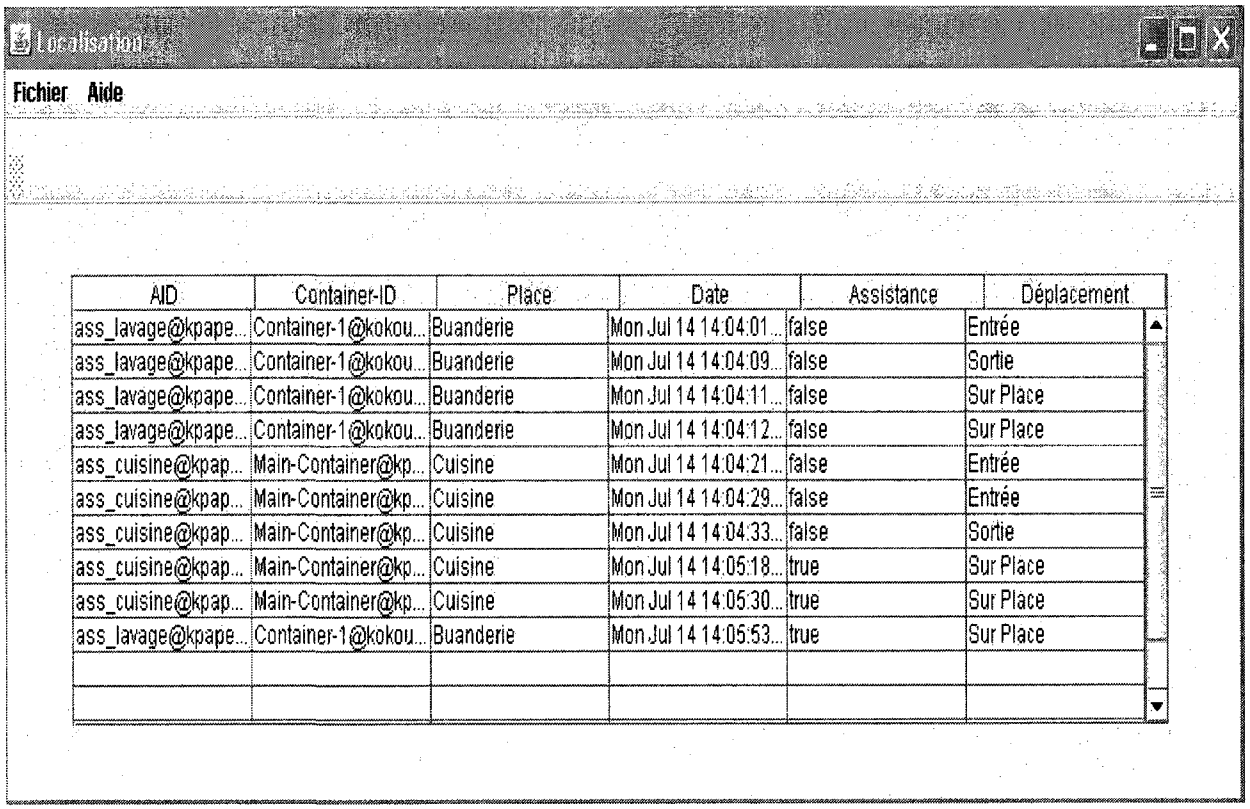


Figure 32: Affichage de l'agent de localisation

La figure 32 nous présente l'interface graphique de l'agent de localisation. Dès que la présence du patient est détectée, l'agent de localisation met à jour en temps réel, le tableau de son interface graphique pour nous informer de la date à laquelle le patient a été détecté, le lieu, l'agent qui l'a détecté et s'il a besoin d'assistance. On aurait pu mettre à la place de ce tableau un plan de l'habitat pour signaler de façon visuelle le point de localisation du patient avec les autres paramètres de localisation susmentionnés.

Essais \	Type	Nombre de fois	Types de réseau	Agents	**Plateformes**
E1	Communication entre agents	7	WLAN/LAN	1 Agents Résident	**Linux Ubuntu**
E2	Communication entre agents	7	Idem	2 Agents Résident	**Linux Ubuntu**
E3	Communication entre agents	7	Idem	3 Agents Résident	**Linux Ubuntu**
E4	Communication entre agents	7	LAN	1 Agents Résident	**Windows XP**
E5	Communication entre agents	7	LAN	2 Agents Résident	**Windows XP**
E6	Communication entre agents	7	LAN	3 Agents Résident	**Windows XP**
E7	Communication entre agents	7	WLAN/LAN	1 Agents Résident	**Vista**
E8	Communication entre agents	7	Idem	2 Agents Résident	**Vista**
E9	Communication entre agents	7	Idem	3 Agents Résident	**Vista**
E10	Communication entre agents	7	Idem	1 Agents Résident	**XP, Ubuntu**
E11	Communication entre agents	7	Idem	2 Agents Résident	**XP, Ubuntu**
E12	Communication entre agents	7	Idem	3 Agents Résident	**XP, Ubuntu**
E13	Test Agent Résident : détection dans une pièce	7	LAN	1 Agents Résident	**XP**
E14	Test Agent Résident : détection devant un appareil	7	LAN	1 Agents Résident	**XP**
E15	Test d'enregistrement	7	LAN	Tous les types agents	**XP**
E16	Test Agent Localisation	7	LAN	1 Agent de Localisation	**XP**
E17	**Test Agent Aidant**	**7**	**LAN**	**3 Agents Aidant**	**XP**

Tableau 4: Définition des essais

8.4 Synthèse des résultats

Le tableau ci-dessous, le tableau 5, présente les résultats des tests automatiques de notre système. Ces tests s'effectuent dans plusieurs environnements, à savoir, Linux et Windows. Nous testons tous les agents de la plateforme au moins sept fois ainsi que son infrastructure de communication.

Essais	Commentaires	Résultats
E1	RAS	Fonctionne
E2	RAS	Fonctionne
E3	RAS	Fonctionne
E4	En-tête du tableau de l'agent de localisation décalé par rapport au corps du tableau, lève une exception de type graphique	Fonctionne
E5	Idem	Fonctionne
E6	Idem	Fonctionne
E7	RAS	Fonctionne
E8	RAS	Fonctionne
E9	RAS	Fonctionne
E10	En-tête du tableau de l'agent de localisation décalé par rapport au corps du tableau, lève une exception de type graphique sur les nœuds XP	Fonctionne
E11	Idem	Fonctionne
E12	Idem	Fonctionne
E13	RAS[1]	Fonctionne
E14	RAS	Fonctionne
E15	RAS	Fonctionne
E16	RAS	Fonctionne
E17	RAS	Fonctionne

Tableau 5: Synthèse des résultats de test

RAS : Rien À Signaler

La réussite des tests E1 à E3 signifie que notre système fonctionne bien sur les plateformes Linux et aussi dans un réseau LAN et WLAN. Le problème à l'issue du test E4 signifie que l'interface graphique de l'agent de localisation peut rencontrer des problèmes de fonctionnement sur les plateformes Windows. Les tests E5 à E17 nous confirment aussi un bon fonctionnement de notre système sur les plateformes Linux et Windows.

8.5 Tests automatiques

Les tests automatiques constituent une partie importante des jeux d'essais parce qu'ils permettent de tester un grand nombre de fonctionnalités en un temps record et sans intervention extérieure. Dans notre cas, les tests automatiques servent à tester les agents de notre plateforme JADE.

Étant donné que nous avons une machine d'état implantée dans le comportement des agents résidents, il est intéressant de passer par tous les états de la machine afin d'observer le comportement de l'agent.

Avec JADE Test Suite, il existe deux types de tests, les tests fonctionnels et les tests atomiques. Les tests fonctionnelles doivent être décrits dans un fichier XML nommée «testerList.xml» et les tests atomiques dans un autre fichier XML que nous sommes libre de nommer. Les emplacements de ces fichiers de descriptions XML doivent se trouver aux emplacements illustrés par la figure 36. Les tests fonctionnels permettent de tester le « comportement » des agents et les tests atomiques de tester des types, c'est-à-dire la valeur des types primitifs et la classe des objets. Les tests doivent hériter de la classe *Test* comme illustré à la figure 34. Tous les tests doivent être regroupés dans une classe qui hérite de la classe *TesterAgent* comme illustré à la figure 35. Il faut noter que la classe *Test* est un composant de la classe *TesterAgent,* c'est elle qui regroupe tous les

tests d'un agent. Pour tester l'agent de localisation, par exemple, nous créons un dossier que nous nommons «localization». Ce dossier contient d'autres sous-dossiers de tests que nous présentons plus loin, le test fonctionnel et le groupe de tests de l'agent de localisation. En principe, dans le sous-dossier «test» du dossier «localization» devraient être créés les tests atomiques et leurs descriptions dans le fichier XML *lzTesterAgent.xml*. Cependant, dans le comportement («behaviour») de l'agent de localisation, nous testons essentiellement le contenu des messages, c'est-à-dire les types d'objets contenus dans les messages provenant des agents résidents, alors, la création des tests atomiques n'est donc pas nécessaire.

8.5.1 Configuration et démarrage des tests

La mise en œuvre des tests automatiques requiert un ensemble de configurations et de la programmation. Nous abordons, dans nos explications, la configuration ensuite la programmation des tests des agents.

La figure 33 présente notre fichier XML de configuration, à savoir, «TesterList.xml». Les sections délimitées par les balises «<Tester>» et «</Tester>» correspondent à la description des tests de différents agents. Il y a autant de sections dans la balise «Tester» qu'il y a d'agents à tester. Tous ces tests sont regroupés à l'intérieur de la section délimitée par les balises racines «<TesterList>» et «</TesterList>». À l'intérieur de la description des tests, la balise «ClasseName» nous indique le nom de la classe java qui implante le test de l'agent. La balise «Description» donne une brève description du test à exécuter. Enfin, la balise «TesterListRif» indique le chemin d'accès au fichier XML de configuration des tests atomiques.

La figure 34 nous présente le code d'une classe Java qui a pour but de charger le «comportement» de l'agent de localisation. La première méthode nommée, «load», crée

une instance du comportement de l'agent et la retourne. La deuxième méthode, «load», appelle la première et retourne le comportement d'agent. La création de la deuxième méthode a pour objectif de nous conformer à la signature exigée par l'infrastructure de test.

La figure 35 nous présente le code d'une classe Java, *TesterAgent*. Elle regroupe l'ensemble des tests d'un agent. Une classe de ce type doit être créée pour chaque agent à tester. Dans le corps de la méthode *getTestGroup*, nous créons une instance de la classe *TestGroup* avec, comme paramètre, le nom du fichier XML qui décrit les tests atomiques de l'agent. Ensuite, nous créons une instance de la classe présentée à la figure 35, à savoir, « LocalizationAgentTester» et nous l'initialisons dans la méthode « initialize».

La figure 36 illustre l'arborescence du répertoire de tests auxquels nous devons nous conformer pour que les programmes puissent s'exécuter. Le dossier « common » contient les classes de JADE Test Suite qui sont utilisées pour programmer les tests. Le dossier « localization » contient les tests de l'agent de localisation et le fichier XML de ce dossier contient la description des tests atomiques.

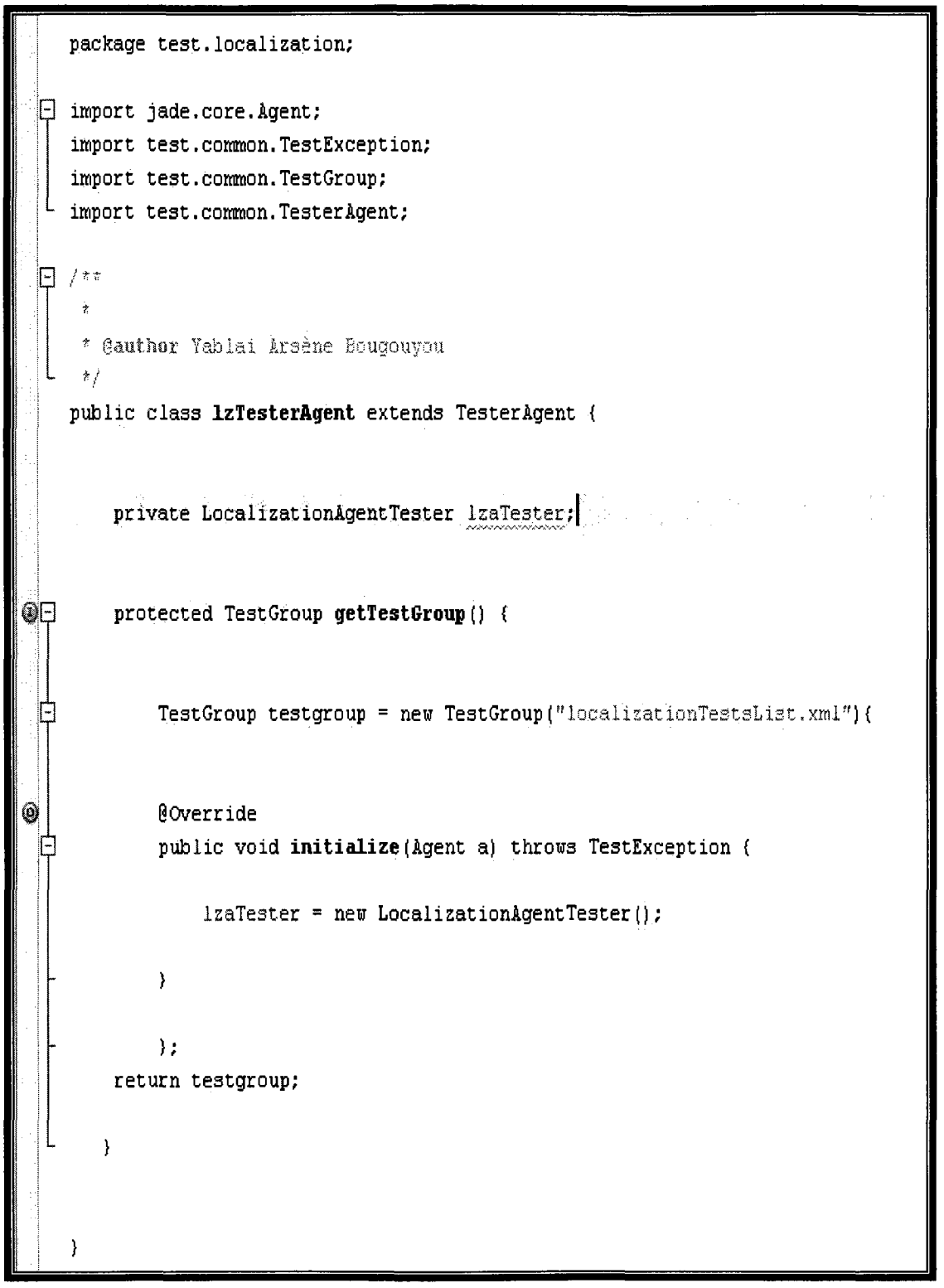
La figure 37 nous présente les résultats des tests automatiques. La barre de progression en couleur verte indique que les tests ont réussi. Les agents de la plateforme sont testés par un seul agent, dont l'AID est «test-suite@testPlateform» qui s'exécute dans un conteneur auxiliaire, le *conteneur-1*. Nous pouvons voir que tous les agents de la plateforme que nous avons testés figurent dans la liste des tests «TesterList». Le conteneur principal *Main-container* contient l'AMS, le DF et le RMA (Remote Management Agent). Le RMA est l'agent qui nous montre l'interface graphique de la plateforme avec les conteneurs et tous les autres agents. C'est par lui que nous pouvons manipuler graphiquement les agents d'une plateforme JADE.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Document   : testerList.xml
    Created on : 15 juillet 2008, 11:59
    Author     :Yablai Arsène Bougouyou
    Description:
        Purpose of the document follows.
-->

<TesterList>

    <Tester name="testAgentAidant" skip="false">
        <ClassName>test.mobility.mobilityTesterAgent</ClassName>
        <Description> test de l'agent aidant</Description>
        <TestListRif>test\\terterList.xml</TestListRif>
    </Tester>
    <Tester name="Agent de Localisation" skip="false">
        <ClassName>test.localization.lzTesterAgent</ClassName>
        <Description> test de l'agent de localisation ACL</Description>
        <TestListRif>test\\terterList.xml</TestListRif>
    </Tester>
    <Tester name="AgentResidentUn" skip="false">
        <ClassName>test.resident.one.RsdTesterAgent</ClassName>
        <Description> Test de la FSM</Description>
        <TestListRif>test\\terterList.xml</TestListRif>
    </Tester>
    <Tester name="AgentResidentDeux" skip="false">
        <ClassName>test.resident.two.RsdTesterAgent</ClassName>
        <Description> test patient devant les appareil</Description>
        <TestListRif>test\\terterList.xml</TestListRif>
    </Tester>
    <Tester name="Enregistrement" skip="false">
        <ClassName>test.registerindf.RegisterInDFTesterAgent</ClassName>
        <Description> test d'enregistrement aupres du DF</Description>
        <TestListRif>test\\terterList.xml</TestListRif>
    </Tester>

</TesterList>
```

Figure 33: Contenu du fichier TesterList.xml

```
package test.localization;

import jade.core.Agent;
import jade.core.behaviours.Behaviour;
import resident.LocalizationAgent;
import test.common.Test;

/**
 *
 * @author Yablai Arsène Bougouyou
 */
public class LocalizationAgentTester extends Test {


    public Behaviour load(LocalizationAgent lza) {

         LocalizationAgent.
                 ReceivingMessageBehaviour
                           behaviour = lza.
                                        new ReceivingMessageBehaviour(lza);

            return behaviour;
    }



   @Override
   public Behaviour load (Agent a){

       Behaviour b = load(new LocalizationAgent());

       return b;
   }
```

Figure 34: Test du comportement de l'agent de localisation



```
package test.localization;

import jade.core.Agent;
import test.common.TestException;
import test.common.TestGroup;
import test.common.TesterAgent;

/**
 *
 * @author Yablai Arsène Bougouyou
 */
public class lzTesterAgent extends TesterAgent {


    private LocalizationAgentTester lzaTester;


    protected TestGroup getTestGroup() {


        TestGroup testgroup = new TestGroup("localizationTestsList.xml"){


        @Override
        public void initialize(Agent a) throws TestException {

            lzaTester = new LocalizationAgentTester();

        }

        };
    return testgroup;

   }


}
```

Figure 35: Groupe de tests pour l'agent de localisation

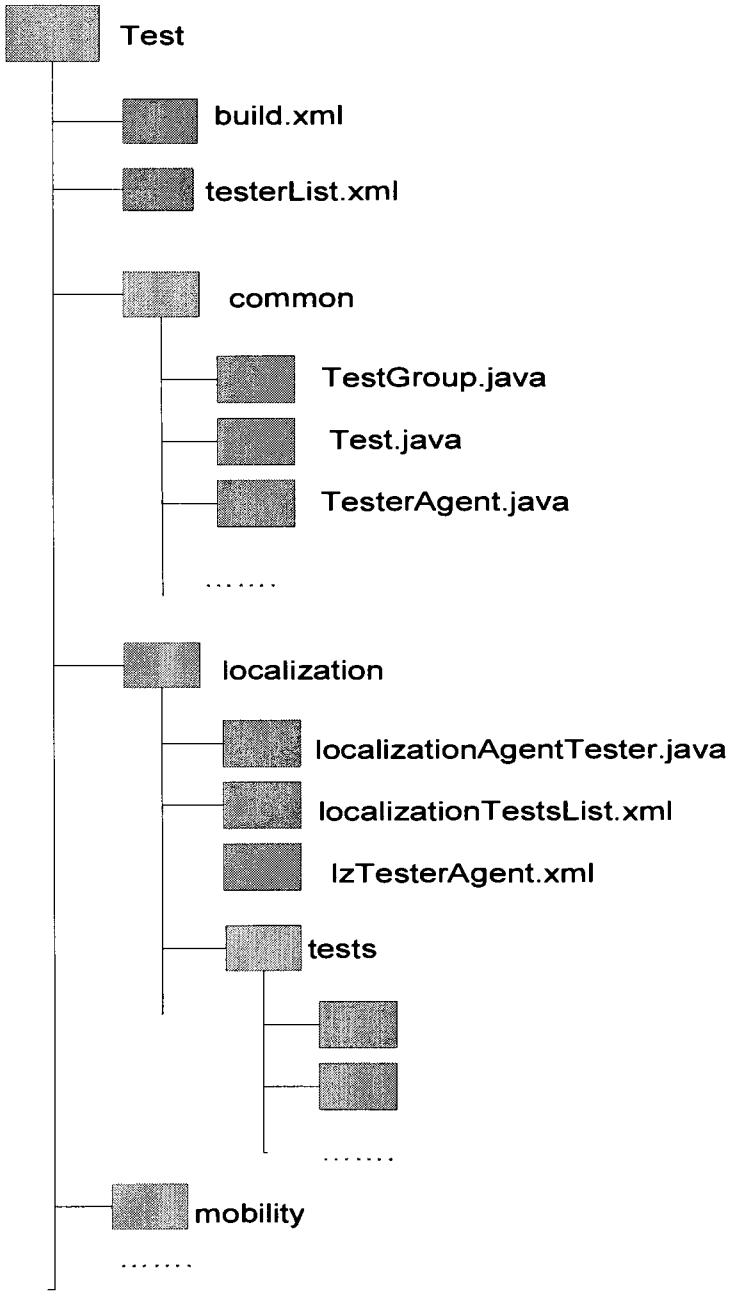


Figure 36: Arborescence du dossier de test

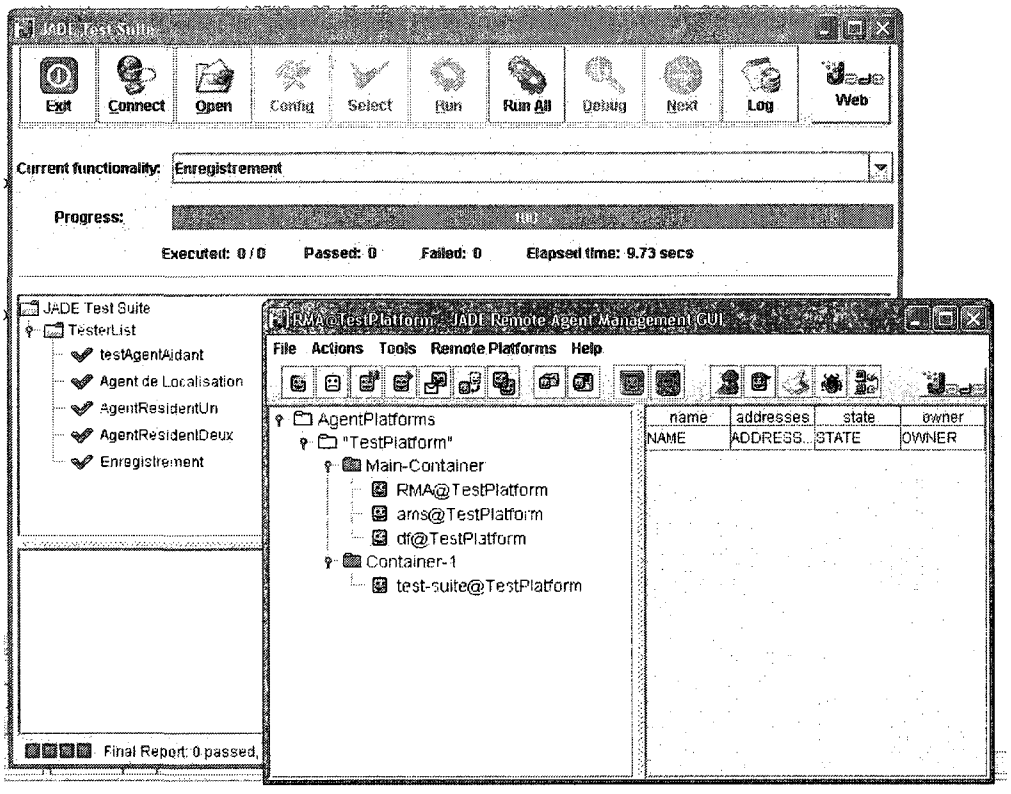


Figure 37: Interface graphique des tests

8.6 Conclusion

L'infrastructure de tests proposée par JADE est plus simple à programmer mais difficile à exécuter à cause des problèmes techniques rencontrés lors de l'exécution. Un des problèmes embêtants que nous avons éprouvé était la présence d'un caractère d'espacement dans le chemin d'accès au répertoire de l'environnement de développement. Finalement, tous les tests automatiques ont été exécutés. Le test du système sur des plateformes hétérogènes s'effectue sans problème majeur, cependant avec Linux, le démarrage des plateformes est impossible si le nom de domaine « Domain Name System » (DNS) n'est pas configuré ou n'existe pas, cela s'explique par le fait que la requête RMI, émise par les conteneurs JADE auxiliaires en vue de se connecter au conteneur principal, donne lieu à une résolution de nom DNS.

Conclusion

Ce projet sur lequel nous avons eu l'opportunité de travailler nous a permis d'acquérir une démarche scientifique rigoureuse, d'apprendre et d'approfondir le concept des systèmes multi-agents, en particulier celui des agents mobiles. Il nous a aussi donné l'occasion d'appliquer une nouvelle méthodologie de développement, à savoir, Gaia. Comme savoir-faire, ce projet nous a permis d'approfondir le langage Java, de s'en servir pour implanter un système multi-agents et surtout d'acquérir les compétences nécessaires à la programmation des agents en se servant de l'infrastructure JADE.
Il faut aussi noter que ce projet a été l'occasion pour nous d'apprendre à travailler sous la direction d'un superviseur dans un cadre académique.

Contributions

Notre travail est un composant (sous-système) qui vient s'ajouter au système de l'habitat intelligent qui tend à améliorer l'autonomie des personnes en déficience cognitive. Le système que nous avons développé et implanté intègre les fonctionnalités de localisation du patient à tout moment afin de lui porter assistance quand cela s'avère nécessaire. Cette localisation du patient devant les appareils et à l'intérieur d'une pièce quelconque de l'habitat est possible grâce à la collaboration des agents de notre plateforme. Il faut aussi noter que notre système affiche les références de localisation du patient en temps réel dans l'interface graphique de l'agent de localisation. La détection des personnes à l'intérieur d'une pièce est implantée au travers d'une machine à états finis que nous avons conçue. Elle évolue en fonction des signaux qu'envoient les capteurs à infrarouge que nous mettons en place. L'apport de l'aide au patient se fait par l'intermédiaire des agents mobiles qui migrent vers lui quand cela s'avère nécessaire.

La conception d'un tel système de localisation des personnes à l'intérieur d'une pièce permet au corps médical de suivre le déplacement quotidien des patients dans leurs habitats et certaines de leurs activités.

Critique du travail

Dans notre système, toutes les communications passent essentiellement par l'agent de localisation ce qui fragilise le système entier, car si ce dernier est détruit, notre système est paralysé. Il est vrai que l'infrastructure JADE intègre la possibilité de mise en œuvre de la redondance, cependant améliorer notre concept reste une option non négligeable. Concernant les conteneurs JADE, nous ne contrôlons pas leur capacité, c'est-à-dire le nombre d'agents qu'ils peuvent accueillir en fonction des ressources de la machine sur lesquelles ils sont installés. Cette situation peut entraîner un dysfonctionnement du système si un agent migre vers un conteneur qui n'est plus capable de l'accueillir, la migration d'un agent dans ce cas est avortée.

Travaux futurs de recherche

Dans le cadre de l'assistance aux personnes en déficience cognitive, plus précisément dans le cadre de notre projet, plusieurs extensions sont possibles, notamment la logique décisionnelle sur laquelle nous nous basons pour faire migrer les agents aidants reste à être améliorée. Cette logique est la suivante, si le patient se trouve devant un appareil pendant quarante secondes sans agir, nous supposons que celui-ci a besoin d'aide. Cette logique reste un point fondamental et complexe de l'habitat, elle peut faire l'objet d'un autre travail, à savoir, étudier le comportement du patient en vue d'établir des règles d'inférences logiques et construire une base de connaissances permettant de déduire si le patient a besoin d'aide dans une situation donnée.

La construction d'une telle logique décisionnelle intégrant une base de connaissances est possible en restant dans le cadre de l'infrastructure JADE. La bibliothèque JSA («JADE Semantics Add-ons») est une bibliothèque JADE conçue à cet effet [29]. L'utilisation d'un autre moteur de règle intégrable à JADE est aussi possible, notamment JESS [21,22]. Quand bien même JESS n'est pas gratuit, il est disponible dans le cadre de la recherche universitaire sous certaines conditions.

Une autre extension possible serait de concevoir un module de gestion de la capacité des conteneurs JADE en fonction des ressources des systèmes sur lesquels ils sont installés. Une fois le conteneur lancé, le module de gestion intégré au système doit pouvoir informer les agents de leur possibilité d'y migrer, ou de le quitter quand les ressources commencent à faire défaut.

Perspectives

Au-delà des habitats intelligents pour les personnes souffrant de déficience cognitive, la question de sécurité des personnes a une place très importante dans la société, de ce point de vue, notre système pourrait être utilisé pour la surveillance des personnes dans les bâtiments, voire des enfants.

Par ailleurs, notre système peut aider à fournir des services dans les réseaux de télécommunications. Par exemple, un fournisseur d'accès Internet peut déployer le système dans ses réseaux pour offrir des services personnalisés aux clients tels que la surveillance des habitations des clients ou le contrôle de l'habitation des clients en leur absence.

Annexes

Annexe A

Tous les agents résidents de notre système intègrent une interface graphique qui sert à simuler le contexte. Cette interface graphique intègre deux boutons dont les événements de souris (« clic ») servent à simuler la coupure des barrières infrarouges et deux autres boutons qui servent à simuler la présence du patient devant un appareil de l'habitat. Un champ de type texte permet de saisir le temps en millisecondes que passe le patient devant un appareil. L'interface graphique est présentée à la figure 38.

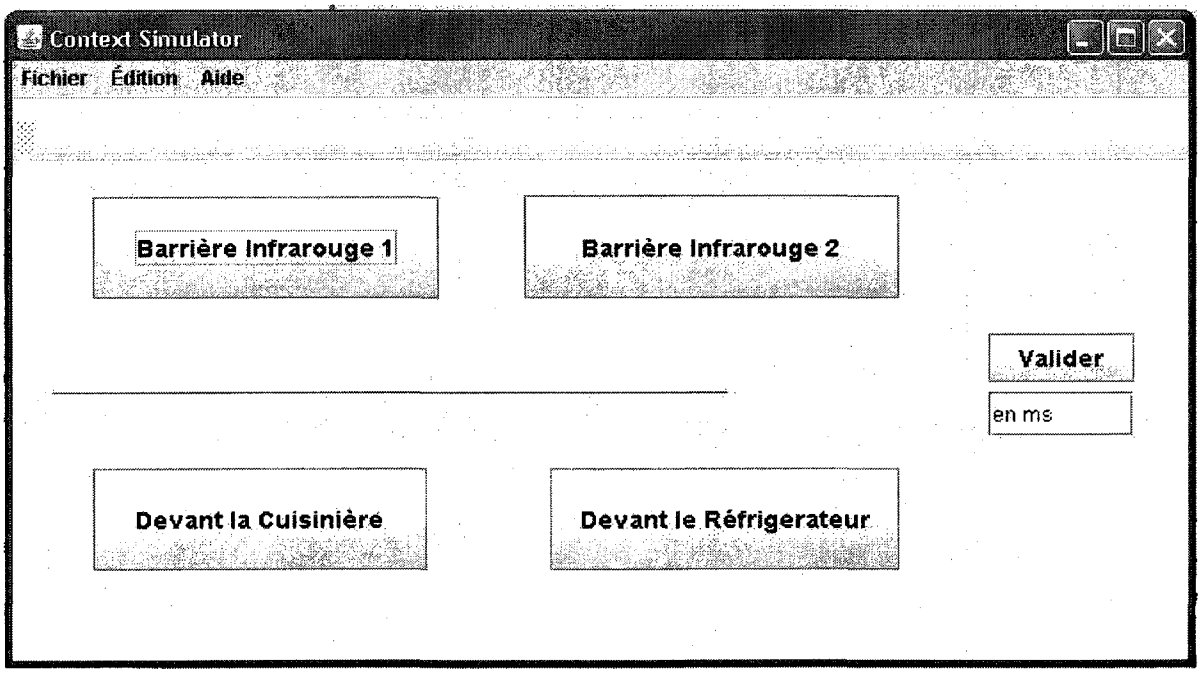


Figure 38: Interface graphique d'un agent résident

Annexe B

Étant donné qu'un agent résident peut être dans une pièce quelconque de l'habitat, nous avons intégré la possibilité de configurer le nom de la pièce, le nom du service et de nommer deux appareils de la pièce en question. Les noms des deux appareils servent à identifier les boutons qui vont simuler la présence du patient devant ces appareils. Cette interface graphique de configuration est présentée à la figure 39.

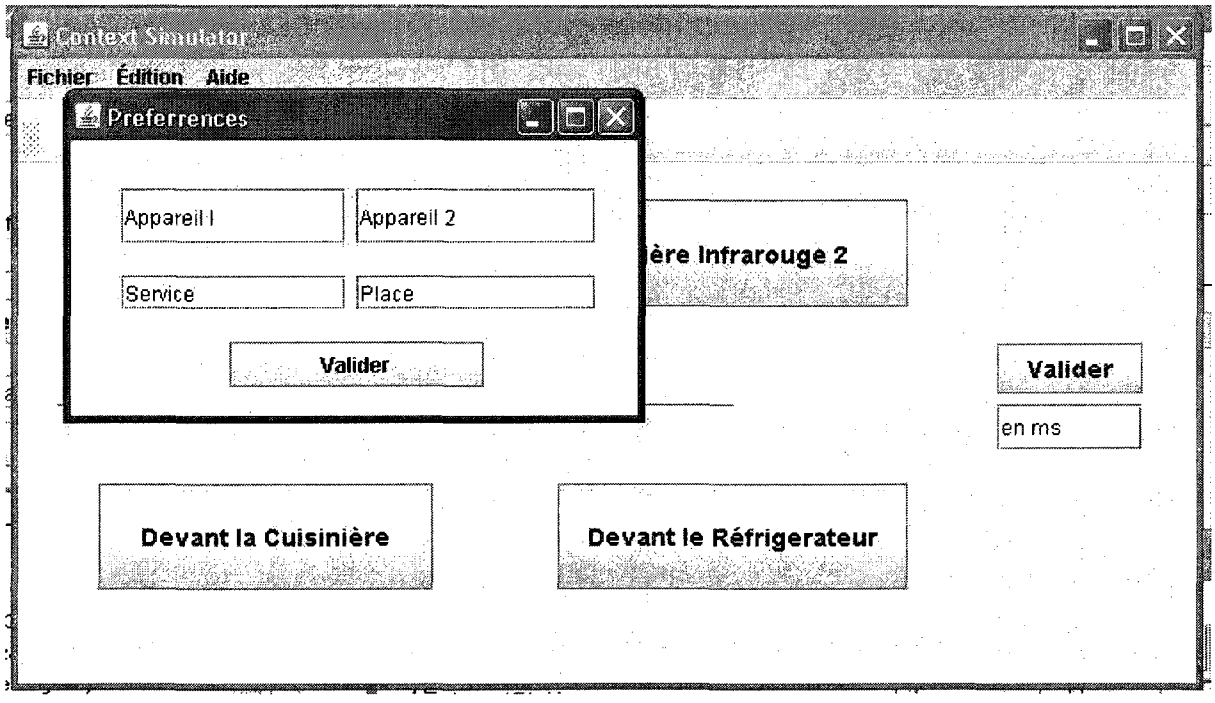


Figure 39: Configuration du service d'un agent résident

Annexe C

La figure 40 présente l'interface graphique des agents aidant. Toute interface permet de configurer les services en choisissant pour chaque agent un service parmi trois. Ces services sont les suivants, « Assistance aux appareils », pour les événements liés aux appareils, « Détection des entrées », pour les événements liés à l'entrée du patient dans une pièce ou « Service de traçage », pour la migration vers l'un des deux premiers agents cités. Nous pouvons insérer une signature à chaque fois que l'agent passe sur un poste dans le seul champ inscriptible de l'interface.

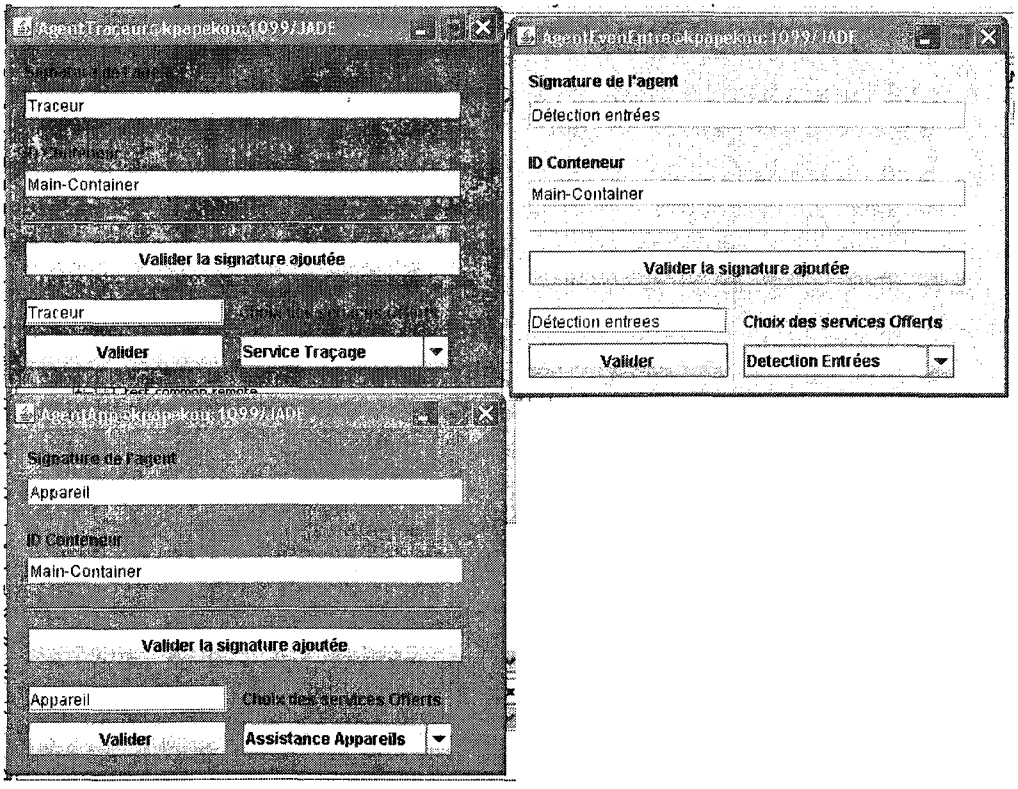


Figure 40: Interface graphique des agents mobiles

Annexe D

La figure 41 nous présente, à droite, les agents de notre système. L'agent « rsdAgent » est un agent résident hébergé sur un téléphone intelligent émulé, le Nokia S80. L'interface graphique à gauche est celui de l'agent de localisation (lzAgent) dans la version embarquée de notre système et en « splitter-mode ». L'interface graphique, présentant une arborescence des agents est l'interface de gestion des agents. Tous les agents de la plateforme y sont listés. L'administrateur peut, instancier, nommer ou détruire un agent de la plateforme à travers cette interface. Dans la rubrique « tools», nous pouvons lancer l'interface graphique de l'agent pages jaunes pour voir le contenu de son registre.

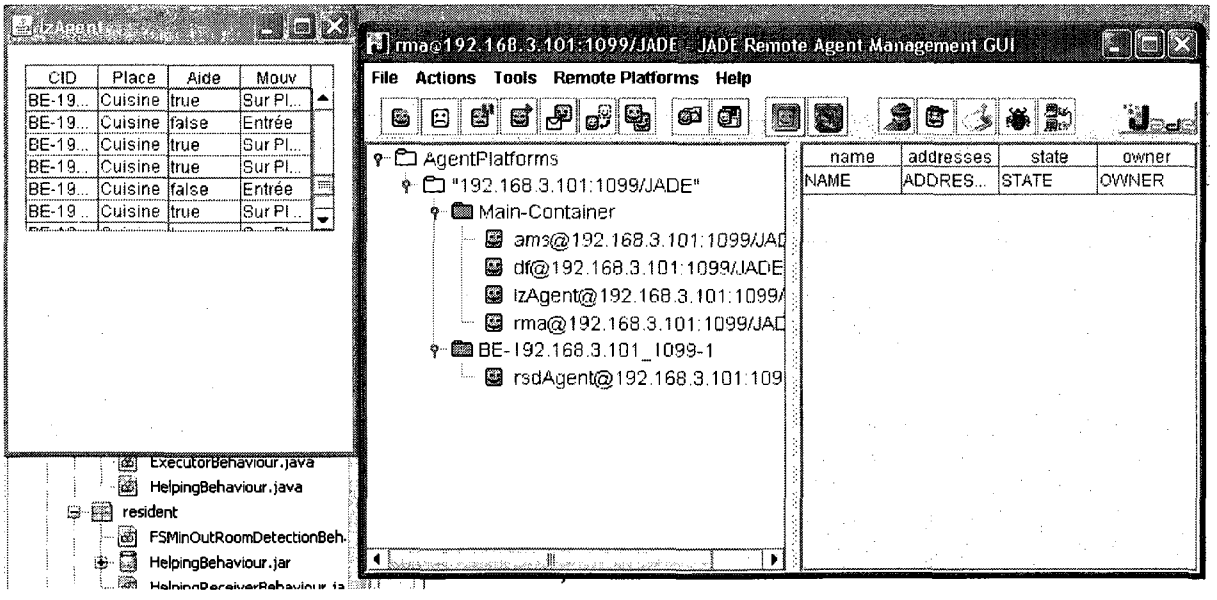


Figure 41: Notre système en "splitter mode"

Annexe E

La figure 42 montre les interfaces graphiques de notre système en version embarquée. Le téléphone à gauche nous montre l'interface graphique d'un agent résident et celle de droite est celui de l'agent de localisation.

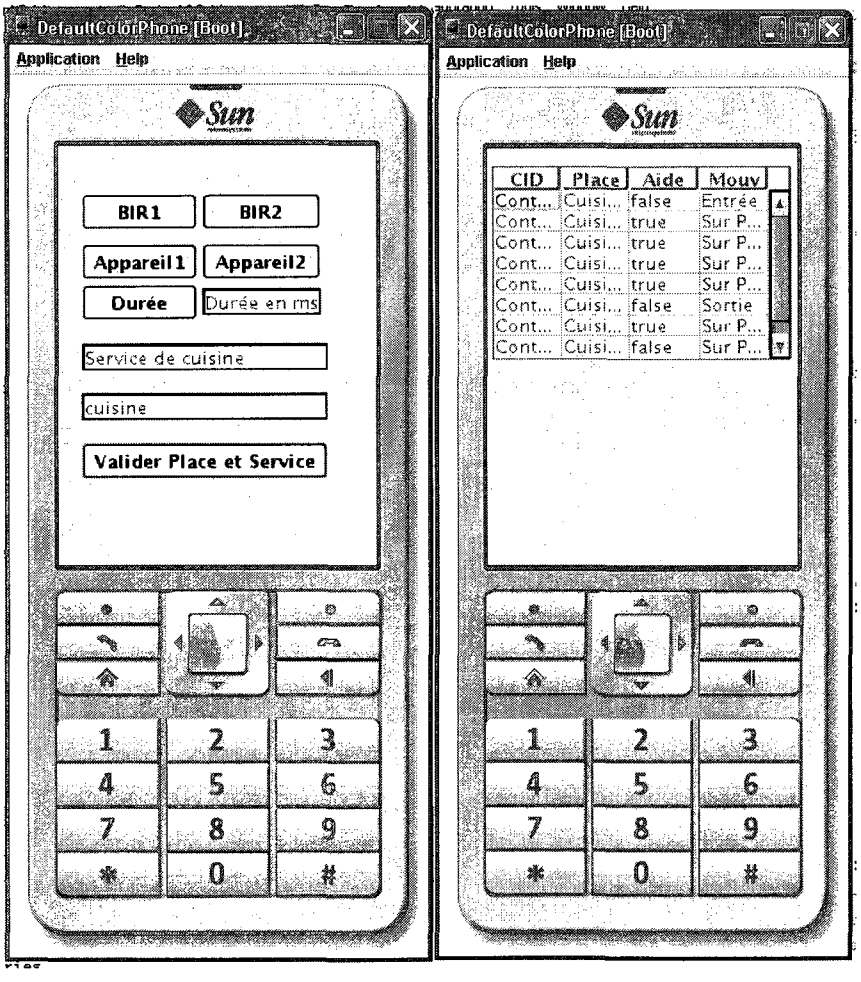


Figure 42: Notre système en version embarquée

Annexe F

La figure 43 nous présente l'interface graphique d'un agent résident dont le conteneur est hébergé dans un téléphone intelligent, c'est l'agent associé à l'événement d'entrée du patient dans une pièce de l'habitat.

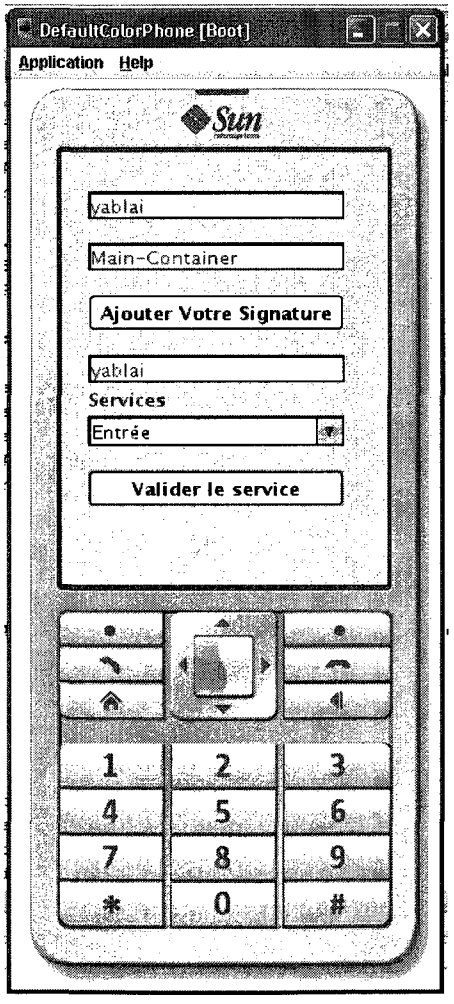


Figure 43: Interface graphique d'un agent aidant

Bibliographie

[1] Agha, G., "A Model of Concurrent Computation in Distributed Systems", *MIT Press,* Cambridge, MA, 1987.

[2] Ametller, J., Robles, S. et Borrell, J. « Agent migration over FIPA ACL Messages », http://jade.tilab.com/

[3] Baumann, J., Hohl, F., Rothermel, K., Strasser, M., « Mole, Concepts of a Mobile Agent System » (à paraître), *WWW Journal, Special Issue on Applications and Techniques of Web Agents*, Baltzer Science Publishers.

[4] Bellifemine, F. Claire, G. Poggi, A. Rimassa, G. « JADE, a white paper » *EXP in search of innovation*, exp - Volume 3, no 3, September 2003.

[5] Brookshier, D. « The Voyager SOA platform », *Recursion software*, http://www.recursionsw.com/About_Us/wp_mobileagents.html?var1=sc2

[6] Caire, G., Garijo, F. Gomez, J. *et al.,* Agent Oriented Analysis using MESSAGE/UML AOSE, 2001.

[7] Chess et Al, 4. D.M. Chess *et al*, « Itinerant Agents for Mobile Computing », *IEEE Personal Comm.*, Vol. 2, No. 5, Oct. 1995, pp. 34-49.

[8] Coulouris, G., Dollimore, J. et Kindberg, T. « Mobile and ubiquitous computing », *Distributed Systems Concepts and Design*, 4e édition, Addison-Wesley, New York, 2005, pages 657-717.

[9] THE FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS. *FIPA Specifications,* [en ligne], 2008, http://www.fipa.org/.

[10] Frappier, M. et St-Denis, R. « EB^3: an entity-based black-box specification method for information systems », *Software and System Modeling,* Vol.2, no. 2, 2003, p. 134 – 149.

[16] Gray, Robert S., « Agent TCL: A flexible and secure mobile-agent system », *Proceedings of the 4th conference on USENIX Tcl/Tk Workshop*, Volume 4, Monterey, California, 1996, p. 2.

[17] Jacobsen, K. et Johansen, D. « Mobile Software on Mobile Hardware - Experience with TACOMA on PDAs », *Technical Report 97-32*, Department of Computer Science, University of Tromsø, Norway, Décembre 1997.

[18] SUN MICROSYSTEMS, *Java Object Serialization Specification,* [en ligne], 1997-1999, http://java.sun.com/j2se/1.3/docs/guide/serialization/spec/serialTOC.doc.html

[19] SUN MICROSYSTEMS. *Java Platform, Micro Edition Connected Device Configuration*, [en ligne], 2005, http://www.java.sun.com.

[20] SANDIA NATIONAL LABORATORIES. *Jess, the rule engine for Java platform,* [en ligne], mis à jour le 11 novembre 2008, http://www.jessrules.com/jess/index.shtml.

[21] « The rule Engine for Java platform », http://herzberg.ca.sandia.gov.

[22] http://jade.tilab.com/doc/tutorials/jade-jess/jade_jess.html, un exemple d'intégration de Jess à JADE

[23] Johansen, D., Jacobsen, K., Sudmann, N. P., Lauvset, K. J., Birman, K. P., Vogels, W., *Using software Design Patterns to build distributed environmental Monitoring Applications.* Cornell Computers Science Technical Report TR97-1655.

[24] Johansen, D., van Renesse, R. et Schneider, F., « Operating system support for mobile agents », *Proc. of the 5th. IEEE HOTOS Workshop*, Orcas Island, USA, 4 et 5 mai 1995.

[25] Kotz, D., et al., « Mobile Agents for Mobile Internet Computing », *IEEE Internet Computing*, vol 1, no 4, juillet et août 1997, p 58-67.

[26] Kotz, D., Gray, R. Nog, S., Rus, D., Chawla, S., Cybenko, G., « Agent TCL: Targeting the Needs of Mobile Computer » *IEEE Internet Computing,* Volume 1, numéro 4, juillet 1997, p. 58–67.

[27] Lange, D. B. et Oshima, M., « Mobile Agent with Java: The Aglet API », Volume 1, numéro 3, 1998, p. 111-121, ISSN:1386-145

[28] Lo Piccolo, F., Bianchi, G. et Salsano, S. « A measurement study of the Mobile Agent JADE Platform », proceeding of the 2006 International Symposium on a World of the Wireless, Mobile and Multimedia Networks (WoWMoM'06), IEEE Computer Society.

[29] Lopes Cardoso, H. « Jade semantic Add-on & Jade semantic Agent », *Integrating JADE and Jess,* http://jade.tilab.com/doc/tutorials/jade-jess/jade_jess.html

[30] Moreno, A., Valls, A., Viejo, A. *Using JADE-LEAP to implement agents in mobile devices*, [en ligne], http://jade.tilab.com/

[31] Morreale, P. « Agent on the Move » *IEEE Spectrum*, Vol. 35, no 4, avril 1998, p.34- 41, ISSN:0018-9235

[32] Nikraz, M., Caire, G. et Bahri, P.A., *A methodology for the Analysis and Design of Multi-Agent Systems using JADE*

[33] Odell, J., « Objects and Agents Compared », *Journal of Object Technology*, Vol 1, numéro 1, mai 2002.

[34] Peine, H. et Stolpmann, T., « The Architecture of the Ara Platform for Mobile Agents », *Proc of the First Intl Workshop on Mobile Agents MA'97,* Berlin, April 7-8, Springer Verlag, http://www.uni-kl.de/AG-Nehmer/Ara

[35] RECURSION SOFTWARE, *Recursion Software .inc*, [en ligne], 2008, http://www.recursionsw.com.

[36] Shoch, J. F, Hupp, J. A., « The "worm" programs —early experience with a distributed computation », *Communications of the ACM*, volume 25, numéro 3, mars 1982, p.172-180.

[37] Stamos, J.W. « Remote Evaluation », *Technical Report 354,* MIT Laboratory for Computer Science, janvier 1986.

[38] « The Tacoma project », http://www.tacoma.cs.uit.no/index.html

[39] RECURSION SOFTWARE, « Voyager Technical Overview, ObjectSpace », http://www.recursionsw.com/Products/voyager.html

[40] White, J., « Telescript Technology: Mobile Agents », *General Magic White Paper* http://www.genmagic.com/Telescript/Whitepapers/wp4/whitepaper-4.html.

[41] White, J. *Mobile Agent White Paper*, MIT Press, Menlo Park, 1996.

[42] Wong, D. *et al.,* Concordia: « Concordia: An Infrastructure for Collaborating Mobile Agents » *Proc. of Workshop on Mobile Agents MA'97*, Berlin, 7 et 8 avril. LNCS 1219, Springer Verlag.

[43] Wooldridge, M. et Jennings, N. R., « Pitfall of Agent-Oriented Development », *Proceedings of the 2nd International Conference on Autonomous Agents* (Agents'98)

[44] Wooldridge, M. *et al.,* « The Gaia Methodology for Agent-Oriented Analysis and Design » *Journal of Autonomous Agents and Multi-Agent Systems,* volume 3, numéro 3, 2000, p. 285-312.

[45] Bouchenak, Sara et Hagimont, Daniel. « Pickling threads state in the Java system », *TOOLS Europe (TOOLS Europe'2000),* Mont-Saint-Michel / Saint-Malo, France, 5 au 8 juin 2000.

[46] Giosuè Vitaglione, « Security Administrator Guide », (TILAB, formerly CSELT), *JADE Tutorial*, [en ligne], http://jade.tilab.com.

[47] SUN MICROSYSTEMS, *Java Security Architecture*, http://www.java.sun.com.

[48] Fabio Bellifemine, Giovanni Caire, Dominic Greenwood, Bellifemine, Fabio Luigi. *Developing multi-agent systems with JADE*, Edition Wiley, 2007, section 6.1.

[49] Sara Bouchenak, « Conference on Object-Oriented Technology and Systems », *Proceedings of the 6th conference on USENIX Conference on Object-Oriented Technologies and Systems*, Volume 6, page 12.

[50] JXTA. *JXTA » Community,* [en ligne], https://jxta-jxme.dev.java.net

[51] Ortiz, Enrique C. An *Introduction to Java Card Technology – Part 1*, [en ligne], 2003, 2007, http://www.java.sun.com.

[52] Ort, Ed. *Writing a Java Card Applet*, [en ligne], 2001, 2007, http ://www.java.sun.com.

[53] Pfaeffle, Tom. *Java Card Interoperability*, [en ligne], 2002, 2007 http://www.java.sun.com

[54] IEEE, *IEEE Computer Society*, [en ligne], 2008, http://www.computer.org.

[55] THE APPACHE PROJECT, [en ligne], mis à jour le 18 janvier 2009, http://ant.apache.org/

[56] OSGi Alliance, [en ligne], http://www.osgi.org/Main/HomePage

[57] RUNES Project, [en ligne], http://www.ist-runes.org/introduction.html

[58] Abdelgadir Ibrahim, Liping Zhao, « Supporting the OSGi Service Platform with Mobility and Service Distribution in Ubiquitous Home Environments », the computer Journal, Vol. 52 No. 2, 2009

[59] Appache Felix, [en ligne], 2009, http://felix.appache.org/

[60] Eclipse Equinox, [En ligne], 2009, http://eclipse.org/equinox/

[61] Knopflerfish, [En ligne], 2009, knopflerfish.org/

[62] Concierge, [En ligne], 2009, http://concierge.sourceforge.net/

[63] André Bottaro, Anne Gérodolle, Philippe Lalanda, « Pervasive Service Composition in Home Network », 21st International conference on Advance Networking and Applications (AINA'07)

[64] Chao-Lin Wu; Chun-Feng Liao; Li-Chen Fu, "Service-Oriented Smart-Home Architecture Based on OSGi and Mobile-Agent Technology," *Systems, Man, and*

[65] Murphy, A., Picco, G. and Roman, G. (2002) On global virtual data structures, In Marinescu, D and Lee, C. (eds) «Process Coordination and Ubiquitous computing, pp. 11-29. CRC Press