FSD N°ordre

Université Saad Dahleb Blida

FACULTE DES SCIENCES

DEPARTEMENT INFORMATIQUE

Mémoire présenter par :

- DJILALI BEN FREDJ OMAR
- HADJ SADOK ALLAA EDDINE

Pour l'obtention du diplôme de Master en Informatique

Domaine :MI

Filière : informatique

Spécialité : informatique

Option : Génie logiciel

Thème

Une plateforme sécurisée pour les agents mobiles

JADE-MP

Setenue le : 02/07/2012 devant le jury composer de :

-Melle Boustia Présidente

-Mr Chikhi Examinateur

Promoteur : Sidoumou

Promotion : 2011 / 2012

MA-004-80-1

Dédicaces

Je dédie ce modeste travail A mes très chers parents qui m'ont toujours soutenu et encouragé à tout moment, que Dieu vous garde pour moi.

Omar

A mes très chers Parents, A tous ceux que j'aime, et tous ceux qui m'aiment, Je dédie ce modeste travail.

Allaa eddine

RESUMÉ

Les applications réparties actuelles reposent souvent sur le modèle client – serveur et ses variantes. L'essor fulgurant de l'Internet et le développement des technologies de l'information et de la communication (TIC) ont généré de nouveaux besoins en termes de performance, d'efficacité, de fiabilité et de sécurité des informations échangées. Des applications telles que la recherche d'information sur le Web ont besoin de performance (en temps de réponse) et de pertinence des résultats retournés. Des applications telles que le commerce électronique exigent de la fiabilité, de la confidentialité et la sécurité des transactions. La consommation de la bande passante est un autre facteur décisif. En effet, le nombre croissant des utilisateurs du Web (1,3 milliards en 2007) a induit un volume d'information énorme sur le réseau Internet, d'où un impact sur les temps de réponses et la disponibilité des services.

Par sa nature synchrone, le modèle client – serveur semble ne pas répondre à ces nouvelles contraintes. Plusieurs approches ont été proposées pour répondre à ces nouvelles exigences, parmi lesquelles la mobilité des codes. Cette dernière a vu l'émergence du paradigme des agents mobiles. Ces derniers sont certes prometteurs (notamment par leur nature asynchrone) mais souffrent du manque de fiabilité et de sécurité.

Nous proposons un modèle fiable et sûr pour les agents mobiles : le modèle acheteur – vendeur (Seller – Buyer ou SB)[1]. Ce modèle applique des mécanismes market comme interactions entre un client (acheteur) et un serveur (vendeur). [Men 04]

Ce modèle devra être implémenté à travers la plateforme Jade existante et évalué.

Mots clés : Application reparties, client-serveur, sécurité, mobilité du code, agents mobiles, modèle acheteur vendeur, JADE.

REMERCIEMENTS

Nous remercions dieu tout puissant qui nous a donné le courage et la patience pour mener à terme ce mémoire pour ses faveurs,. Au terme de ce travail nous adressons notre profonde gratitude à Mr Mohmed Redha Sidoumou pour sa disponibilité et son soutien tout au long de cette année, et pour avoir assumé si bien son rôle de promoteur.

A Mr Benguassmia Bahri pour ses conseils tant que encadreur et son dévouement nous ont permis d'avancer et d'évoluer dans un domaine qui était totalement inconnu pour nous, et aussi de l'originalité du sujet et de tout l'intérêt et l'attention qu'il a portés à ce travail du début jusqu'à la fin. Nous remercions tous ceux qui ont contribué de prés ou de loin à la réalisation de ce modeste travail, Nous remercions également tous les membres du jury qui ont accepté d'évaluer notre travail.

Abstract

The current distributed applications often rely on the client - server model and its variants. The explosive growth of the Internet and the development of information technologies and communication technologies (ICTs) have generated new needs in terms of performance, efficiency, reliability and security of information exchanged. Applications such as information retrieval on the Web need performance (response time) and relevance of results returned. Applications such as e-commerce require the reliability, confidentiality and security of transactions. The consumption of bandwidth is another critical factor. Indeed, the growing number of Web users (1.3 billion in 2007) led to an enormous amount of information on the Internet, which has an impact on response times and service availability. By its nature synchronous, the client - server model does not conform to these new constraints. Several approaches have been proposed to meet these new requirements, including mobility codes. The latter has seen the emergence of mobile agent paradigm. These are certainly promising (especially by their asynchronous nature) but suffer from lack of reliability and safety.

We offer a reliable and safe for mobile agents: the model buyer - seller (Seller - Buyer or SB) 1. This model uses market mechanisms such as interactions between a client (buyer) and a server(seller).This model will be implemented through the existing platform Jade and evaluated through application of information retrieval (IR) and simplified by using a suitable benchmark. [Men 04]

Keywords: Distributed applications, client-server, security, code mobility, mobile agents, seller-buyer model, JADE

Liste des figures

Fig-1.1 Schémas d'organisation Client / Serveur 4

Fig-1.2 Modèle Client serveur Deux Tier.....6

Fig-1.3 Logique applicative réside dans le client 7

Fig-1.4 Logique applicative réside dans le Serveur.....7

Fig-1.5 Exemple d'un serveur qui agit comme un client [DIS 06] 7

Fig-1.6 Modèle N tiers (Multi-tiers) 8

Fig-1.7 Fonctionnement du RPC.....10

Fig-1.8 architecture interne du RMI Architecture interne de RMI 13

Fig-1.9 Fonctionnement de RMI.....14

Fig-1.10.RMI avec échange d'objets 16

Fig-1.11 Evaluation à Distance (1) 17

Fig-1.12 Evaluation à Distance (2) 17

Fig-1.13 Code à la demande.....18

Fig-1.14 Schéma d'interaction par Agent Mobile 19

Fig- 2.1 structure d'un agent mobile..... 26

Fig-2.2 Modèle des Langages de Communication entre Agents.....35

Figure 1Fig.-3.1Les services fournis par FIPA.....48

Figure 2Fig3.2 Architecture d'une plateforme JADE [JADE 07].....52

Figure 3Fig-3.3 Le modèle Agent Groupe Rôle 54

Figure 4Fig-3.4 Architecture Partielle de Telescript 58

Figure 5Fig-3.5. Architecture D'Agents 60

Figure 6Fig-3.6 L'architecture d'Aglets dans un Serveur 62

Fig-4.1. Modèle SB: Interaction des agents mobiles à travers une place de marché 70

Fig-4.2 Modèle de la sécurité de la plateforme avec l'infrastructure PKI [MEN 04] 73

Fig4.3 Schémas général de l'architecture JADE-MP.....75

Fig-4.4. Le diagramme d'hiérarchie de classe des agents du Framework 79

Fig-4.5.Diagramme de classe AUML de l'agent AF_AP.....80

Fig-4.6 Diagramme de classe AUML de l'agent AF_MP.....81

Fig-4.7. Diagramme de classe AUML de l'agent AF_Eshop.....81

Fig-4.8. Diagramme de classe AUML de l'agent xMA 82

Fig-4.9 Interactions de l'agent mobile avec l'agent AP_FA et l'agent MP_FA.....83

Fig-4.10. Interactions Au niveau d'un MP avec les Agents MP_FA et Eshop_FA.....84

Fig-4.11 Interactions Au niveau d'un E-shop entre BMA et SMA.....85

Fig-4.12.Diagramme de classe de MPNS_DB 88

Fig-4.13.Diagramme de classe de MPDS_DB 88

Fig-4.14.Diagramme de classe de la base de données au niveau de l'AP.....89

Fig-4.1. Modèle SB: Interaction des agents mobiles à travers une place de marché.....70

Fig-4.2 Modèle de la sécurité de la plateforme avec l'infrastructure PKI [MEN 04] 73

Fig4.3 Schémas général de l'architecture JADE-MP.....75

Fig-4.4. Le diagramme d'hiérarchie de classe des agents du Framework 79

Fig-4.5.Diagramme de classe AUML de l'agent AF_AP.....80

Fig-4.6 Diagramme de classe AUML de l'agent AF_MP.....81

Fig-4.7. Diagramme de classe AUML de l'agent AF_Eshop.....81

Fig-4.8. Diagramme de classe AUML de l'agent xMA 82

Fig-4.9 Interactions de l'agent mobile avec l'agent AP_FA et l'agent MP_FA.....83

Fig-4.10. Interactions Au niveau d'un MP avec les Agents MP_FA et Eshop_FA.....84

Fig-4.11 Interactions Au niveau d'un E-shop entre BMA et SMA.....85

Fig-4.12.Diagramme de classe de MPNS_DB 88

Fig-4.13.Diagramme de classe de MPDS_DB 88

Fig-4.14.Diagramme de classe de la base de données au niveau de l'AP.....89

Liste des tableaux:

Tab-2.1 Les différents messages de FIPA ACL 37

Tab.3.1. Comparaison entre les différentes Framework- Protection de La Plateforme.....62

Tab.32. Comparaison entre les différentes Framework- Protection d'agent Mobile.....64

Tab-4.1 Descriptions des classes au niveau d'AP 86

Tab-4.2Descriptions des classes au niveau d'AP 88

Tab-5.1 types de mobilité dans jade.....98

Tab-6.1. Différents types d'attaque et leurs contremesures. 118

LISTE DES ABREVIATIONS

ACL :	Agent communication Language
AGR :	Agent Groupe Rôle
AID :	Agent Identifier
AMS :	Agent Management System
API :	Application Programming Interface
AP :	Agent Provider
AP_F	Agent Provider Facilitator
BAL :	Boîte Aux Lettres
BDI :	Beliefs Desires and Intentions
BVC :	Base Virtuelle de Connaissances
CA :	Certification Authority
CBR:	Case Based Reasonning
CDPS:	Cooperative Distributed Problem Solving
CMU:	Carnegie Mellon University
CNP :	Contract Net Protocol
CFP :	Call For Proposals
CPU :	Central Unit Processing
DF :	Directory Facilitator
DoS	Denial of Service
ED :	Environnement de Développement
EDMA :	Environnement de Développement Multi-Agent
FQDN:	Fully Qualified Domain Name
FIPA:	Foundation For Intelligent Physical Agent
FIPA ACL :	Foundation For Intelligent Physical Agent Agent Communicating Language
GUID :	Global Unique Identifier
HTTP	Hypertext Transfer Protocol
IAD :	Intelligence Artificielle Distribuée
ILP :	Inductive Logic Programming

Sommaire

RESUMÉ I

Abstract II

Listes des figures et des tableaux: III

LISTE DES ABREVIATIONS IV

Sommaire VI

Introduction générale XI

Partie I : Etat de l'art Chapitre 1 : Evolution du Client/Serveur

Introduction générale 3

1. L'approche Client / Serveur : [ICS 09] 7

2. Un client serveur lourd ou léger : 8

2.1 Architectures Clients Lourds : 8

2.2 Architectures Serveurs Lourds : 8

3. Classification des systèmes client/ serveur : 9

3.1 Le modèle Deux tiers : 9

3.2 Le modèle Trois tiers : 9

3.3 Le modèle N tiers (Multi-tiers): 10

4. Evolution du modèle Client / Serveur 11

4.1 Les différents types de requêtes 12

4.1.1 L'envoi de message 12

4.1.2 Appel de procédure à distance (RPC) 12

4.1.3 Appel de méthode à distance 14

4.1.4 Java RMI : 15

4.1.4.1 Architecture interne de RMI (structure des couches RMI) 15

5. Introduction à la Mobilité 18

4.1 L'exécution à distance (REV, Remote Evaluation) :.....20

4.2 Code à la Demande :.....21

4.3 Les Agents mobiles 21

Chapitre 2: Les système multi-agents

1)Les Agents : 23

2)Agent mobile.....25

3)SYSTEMES MULTI AGENT - SMA - :.....29

3.1 Définitions d'un Système Multi-Agent 29

3.1.1Définition 1[FERB 95].....29

3.1.2Définition 2 :.....30

3.1.3Définition 3.....30

3.2 Les Caractéristique d'un System Multi-Agents [GutMich 04] [HBMAS 09] :.....30

3.3 Avantages et inconvénients des SMA.....31

3.3.1 Les avantages des SMA [Ferb 95] [IGI 2007].....31

3.3.2 Les inconvénients des SMA :[Ferb 95] [GutMich 04] 32

3.4 Les architectures des SMA : 33

3.4.1 Les architectures à base des tableaux noirs : 33

3.4.2Les systèmes hiérarchisés (ou horizontal) 34

3.4.3L'approche totalement distribuée.....34

3.5 Interaction & communication dans les SMA.....34

3.5.1Introduction.....34

3.5.2Langage de communication : 35

3.5.2.1Knowledge Query and Manipulation Language- KQML: 36

3.5.2.2FIPA-ACL [FIP02a].....37

3.5.3Protocole d'interaction : 38

3.5.4Exemples de protocoles d'interaction 39

3.5.4.1Le protocole Contract Net (réseau contractuel).....39

3.5.4.2Les protocoles d'enchère.....39

3.5.5Classification des protocoles d'interaction 41

3.5.5.1Les protocoles Compétitifs vs. Coopératifs 41

3.5.5.2Les protocoles Unidirectionnels vs. Bidirectionnels 41

3.6.Autres classifications.....42

3.6.1Implémentation des protocoles d'interaction.....42

Chapitre 3 : Plateforme et Framework des systèmes multi agent

Introduction : 43

1. Les Normes Des Agents Mobiles:.....44

1.1 OMG MASIF :.....44

1.1.1 Les standards : 44

1.2 FIPA :.....45

1.2.1 Brève histoire de FIPA : [JADE 07] 45

1.2.2Les standards : 46

1.2.2.1Les agents selon la FIPA.....46

1.2.2.2Les Directory Facilitator (DF) Selon la FIPA 47

1.2.2.3Agent Management System (AMS) :.....47

1.2.2.4Message Transport Service (MTS) ou Agent Communication Language ACL.....47

1.2.3L'interopérabilité entre plateformes FIPA :.....47

1.2.3.1Communication entre Agents :.....47

1.2.3.2Environnement d'exécution d'un agent sur chaque plateforme:.....47

2. Les Plateformes Agent Mobiles : 47

2.1 La Plateforme JADE :.....47

2.1.1Brève histoire de JADE : [JADE 07] 48

2.1. Caractéristiques :.....48

2.1.3L'architecture de JADE :.....49

2.1.4Interaction dans JADE: 50

2.2 Madkit : [Web07, GDM 09] 50

2.2.1Le micronoyau Agent [DevG 09] 50

2.2.2Agentification des services.....51

2.2.2.1Agents, groupes et rôles dans la plateforme MADKIT.....51

2.2.2.2Services agents 52

2.3 La Plateformes Magique [Web 08] [TuT 09].....53

2.4 La Plateforme Jack.....54

3 Les Framework Agents mobiles existantes :.....55

3.1 Telescript :.....55

3.2 AgentTcl et D'Agents :.....57

3.3 Aglets: 58

Conclusion : 61

Partie II: Conception et Réalisation Conception :

Introduction.....64

1. Limites de la technologie des agents mobiles :.....65

1.1 Les Problèmes de sécurité : 65

1.1.1Protection des sites d'accueil 65

1.1.2Protection des agents 66

2. Proposition 67

2.1 Un prototype d'architecture à base du modèle seller-buyer [Men 04] :.....67

2.2.1Les « Agents Providers » (AP): 68

2.2.2Place de marché (Market Places): 69

2.2.3Le Service MPNS : (MP Name Service) 69

2.2.4La sécurité de la plateforme : 70

3. Conception de JADE-MP.....71

3.1 Conception des composants JADE-MP.....72

3.1.L'Agent Provider (AP):.....72

3.1.2Les Place de marché (MP): 72
3.1.3L'E-Shop : 72
3.2 Conception des agents JADE-MP 74
3.2.1Les agents du système 74
3.2.2Les agents statiques :....74
3.2.3Les agents mobiles 76
4 Les classes d'agents :....76
4.2 Les agents facilitateurs (statiques):....76
4.3 Les agents mobiles:....77
5 Les diagrammes de classes des Agents du système :....77
5.2 Les agents facilitateurs : 78
5.2.1L'agent facilitateur d'AP....78
5.2.2L'agent facilitateur d' MP....79
5.2.3L'agent facilitateur d'E-shop 79
5.3 Les agents mobiles xMAs :....80
6 Communication entre les agents du système: 81
6.2 Au niveau d'AP : 81
6.3 Au Niveau d'MP :....82
6.4 Au Niveau d'E-shop : 83
7. Conception des bases des Connaissances du système 84
7.1 La base de données au niveau de l'AP 84
7.1.1La base de connaissance MPNS_DB :....84
7.1.2La base de données TSA_DB : 85
7.2 La base de données au niveau de la MP....85
7.2.1La base de connaissance MPDS_DB....85
7.3 La base de données au niveau de l'E-Shop 85
Conclusion : 90

Implementation

Introduction : 93

1. Technologies utilisées....94

1.1 Langage de programmation 94

1.2 Serveur de bases de données....94

1.3 Serveur Web :....94

1.4 Plateforme d'agents : 95

1.4.1L'environnement d'exécution des agents : 95

1.4.2Architecture logiciel de la plate-forme JADE 95

1.4.3Le protocole Contract-Net de JADE....97

2. La Mobilité des agents JADE:....97

3. Réalisation du Framework :....98

3.1 L'Agent Provider (AP): 101

3.2 Localisation des MPs :....102

3.3 Les Place de marché (MP):....103

3.4 L'E-Shop :....105

3.5 Les agents du système....106

Evaluation de la sécurité

Introduction : 110

1. Présentation de quelques Attaques possible sur notre Plateforme : 112

2. Les Contremesures utilisées: 115

2.1 Détection de la falsification des agents [MEN 04]: 117

Conclusion : 119

Conclusion générale 120

Bibliographie

Introduction générale

En raison de la demande incessante pour l'amélioration et l'augmentation de la capacité et du rendement des différents systèmes informatiques, une grande diffusion des réseaux informatisés et des systèmes distribués a était lancée. De plus, les exigences et les préférences des utilisateurs qui ne cessent d'accroître, ainsi que la nécessité d'une évolutivité progressive des applications font qu'une vision centralisée, rigide et passive atteint ses limites. En fait, Il y a quelques années la plupart des applications étaient conçues pour fonctionner dans un environnement d'exécution relativement bien connu et maîtrisé.

Cependant, cela n'est plus possible dans le contexte actuel, où les conditions d'exécution pour une application donnée sont diversifiées et souvent variables. En conséquence, concevoir de telles applications est de plus en plus complexe.

En revanche, les progrès technologiques récents ont permis l'émergence d'une panoplie de nouveaux moyens facilitant la tâche de la conception et du développement de ces applications. En effet, l'essor de l'informatique distribuée et nomade s'est accompagnée de l'apparition du paradigme des agents mobiles. Ce dernier offre une plus grande souplesse et efficacité de traitement.

Un agent mobile est généralement défini comme une entité logicielle autonome ayant une activité propre, agissant pour le compte d'une personne ou d'une organisation et pouvant migrer d'un site à un autre au sein d'un réseau informatique. L'une des motivations les plus importantes de ce paradigme est la minimisation des communications distantes permettant ainsi d'économiser la consommation de la bande passante. La technologie des agents mobiles présente d'intéressantes perspectives pour divers domaines d'applications de l'ère actuelle. On trouve, parmi ces domaines, le commerce électronique, la télécommunication, la recherche d'informations dans le web, la gestion des bases de données et des réseaux, etc.

Cependant, les environnements particuliers dans lesquels ces applications sont déployées font que leur construction représente un vrai défi pour les programmeurs. Leur conception impose la

prise en compte non seulement des aspects purement fonctionnels mais aussi des problèmes déterminant leur qualité de service comme le temps de réponse, la tolérance aux pannes et en particulier ***la sécurité*** .

Dans ce mémoire, nous allons réaliser un « Framework » sécurisé ou une plateforme sécurisée pour les agents mobiles en se basant sur le modèle ***acheteur-vendeur (Seller-Buyer ou SB)*** [1]. Ce modèle applique des mécanismes *market* comme interaction entre un client (acheteur) et un serveur (vendeur).

Le mémoire est organisé en deux parties :

La première partie : est un état de l'art contenant trois chapitres

- Chapitre 1 : Évolution du client/serveur

Dans ce chapitre nous allons citer brièvement les caractéristiques principales du modèle client/serveur (communication par messages, RPC et RMI) et son évolution jusqu'à arriver aux agents mobiles.

- Chapitre 2 : Les systèmes multi-agents (les SMAs)

Dans ce second chapitre nous allons présenter les caractéristiques des agents logiciels, la communication entre eux, interaction et négociation et enfin un bref aperçu des système multi-agents.

- Chapitre 3 : les plateformes multi-agents

Dans ce chapitre nous allons parler des différentes plateformes multi-agents existantes et leurs caractéristiques et on termine par une petite comparaison de ces dernières

La deuxième partie : Elle est consacrée à la conception, la mise en œuvre et l'évaluation du travail réalisé.

- Chapitre 4 : Conception

Dans ce chapitre nous allons donner tous l'architecture générale et modulaire de notre Framework. Nous parlerons également de la conception de cette architecture ainsi la modélisation UML et AUML des déférentes classes utilisées.

Chapitre 5 : Implémentation

Dans ce chapitre nous parlerons de l'aspect programmation de l'environnement de travail ainsi que les classes et les bibliothèques utilisées

- Chapitre 6 : Évaluation

  Ce dernier chapitre nous permettra d'évaluer la sécurité de notre Framework.

Nous terminerons par une conclusion et quelques perspectives.

Partie 1 : Etat de l'art

Chapitre 1 :

L'évolution du client/serveur

Introduction :

Le modèle client/serveur est certainement le modèle le plus utilisé pour la construction d'applications réparties. Il est notamment utilisé dans les environnements Corba [OMG 91] et Java [GOS 95] à travers un service d'appel d'objet à distance.

Récemment, la recherche en systèmes répartis a vu l'émergence d'un nouveau modèle pour la structuration d'applications réparties : la programmation par agents [FUG 98] [THO 97]. Dans ce modèle, un agent est un processus possédant un contexte d'exécution, incluant du code et des données, pouvant se déplacer de machine en machine (appelées serveurs) afin de réaliser la tâche qui lui est assignée.

Les agents visent principalement des applications réparties sur des réseaux à grande distance, car ils permettent de déplacer l'exécution vers les serveurs et de diminuer ainsi le coût d'accès à ces serveurs [CHE 94].

Dans ce chapitre, nous présentons une évaluation de performance du modèle client/serveur vers la technologie des Agents.

1. L'approche Client / Serveur : [ICS 09]

L'approche Client serveur est une nouvelle technologie qui offre une solution à plusieurs problèmes de gestion des bases de données. Le terme client / Serveur décrit un modèle de développement des applications distribuées. Il est basé sur la distribution des fonctions entre deux types de processus autonomes et totalement différents : Client et Serveur.

Client:

Le client est tout processus qui envoie une requête au processus serveur en demandant un service spécifié fourni par ce dernier.

Serveur:

Le serveur est un processus fournissant des services demandés par les processus clients.

Les processus client serveur peuvent résider dans différentes machines d'un même réseau, dans ce cas le serveur peut fournir des services à plusieurs clients et le client peut demander des services de plusieurs serveurs dans son réseau. Le schéma suivant illustre un modèle d'organisation client / serveur :

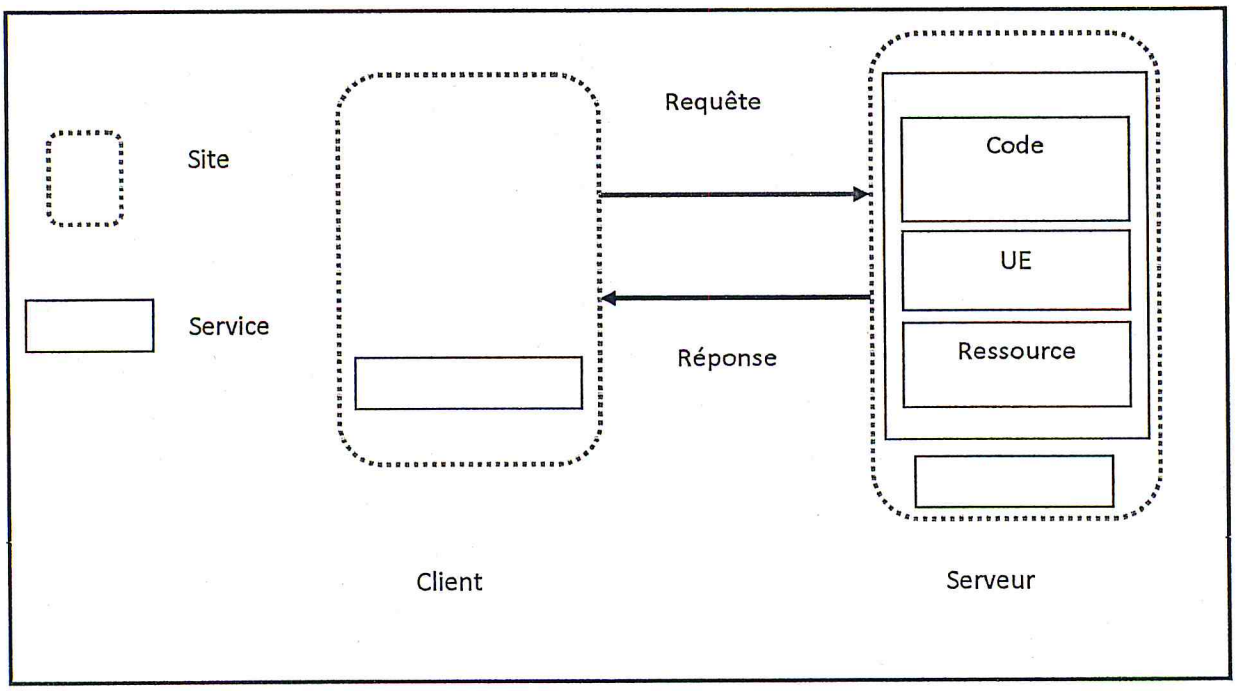


Fig-1.1 Schéma d'organisation Client / Serveur

2. Un client serveur lourd ou léger :

Le terme « *client lourd* » (en anglais « fat client ») désigne une application cliente graphique exécutée sur le système d'exploitation de l'utilisateur. Un client lourd possède généralement des capacités de traitement évoluées et peut posséder une interface graphique sophistiqué.

Le terme « *client léger* » (parfois « client pauvre », en anglais « thin client »), par opposition au client lourd, désigne une application accessible via une interface web (en HTML) consultable à l'aide d'un navigateur web, où la totalité de la logique métier est traitée du côté du serveur. Pour ces raisons, le navigateur est parfois appelé client universel. L'origine du terme lui-même provient de la pauvreté du langage HTML, qui ne permet de faire des interfaces relativement pauvres en interactivité.

2.1 Architectures Clients Lourds :

Dans ce type d'architecture la plupart des traitements se déroulent du côté Client(s). Elle est utilisée dans les modèles clients / serveur classique. La maintenance peut poser des problèmes pour ce type.

2.2Architectures Serveurs Lourds :

Dans ce type d'architecture la plupart des traitements se déroulent du côté Serveur(s).
Le serveur offre des services de plus haut niveau. Les applications actuelles se basent sur des **serveurs lourds.**
L'avantage de ce type d'architecture est que c'est plus facile à gérer parce que la maintenance se fait seulement du côté des serveurs, mais la mise à jours de tous les clients est vraiment un problème.

3. Classification des systèmes client/ serveur :

Il existe trois types de systèmes clients serveurs :

1) Deux-tiers
2) Trois-tiers
3) N-Tiers

Les termes « multi-tiers » sont abusivement traduits de l'anglais *multi-tier* ou *n-tier*. Pour cette raison, il serait préférable d'employer la traduction « *architecture n-tier*» ou « multi-niveaux » ou bien un hybride français-anglais « multi-tier ».

3.1 Le modèle Deux tiers :

Ce modèle est le plus simple car il a deux types de machines:

- Un PC client qui contient l'interface d'utilisateur (Gestion de présentation)
- Un serveur qui contient les bases des données. (Réalisation)[DIS 06]

Le client accède directement aux bases des données [Chan 07]. Dans ce cas on sépare les traitements d'application des requêtes et les mises à jour de la base de données. [ICS 09]

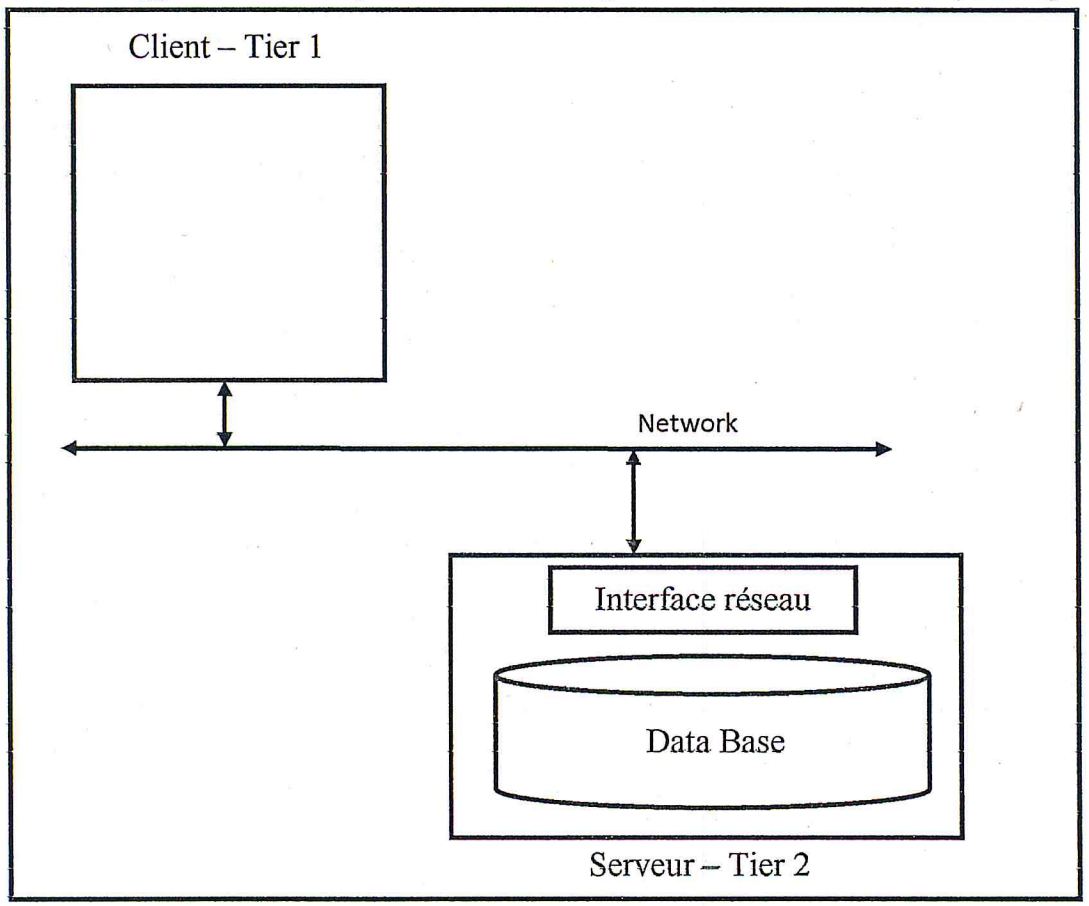


Fig-1.2 Modèle Client serveur Deux Tiers

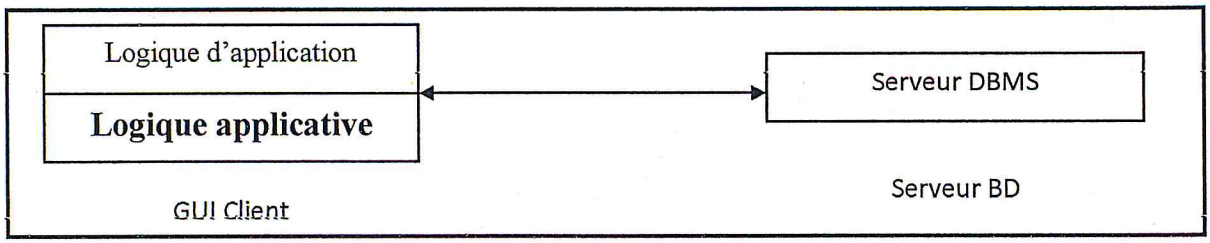


Fig-1.3 Logique applicative réside dans le client

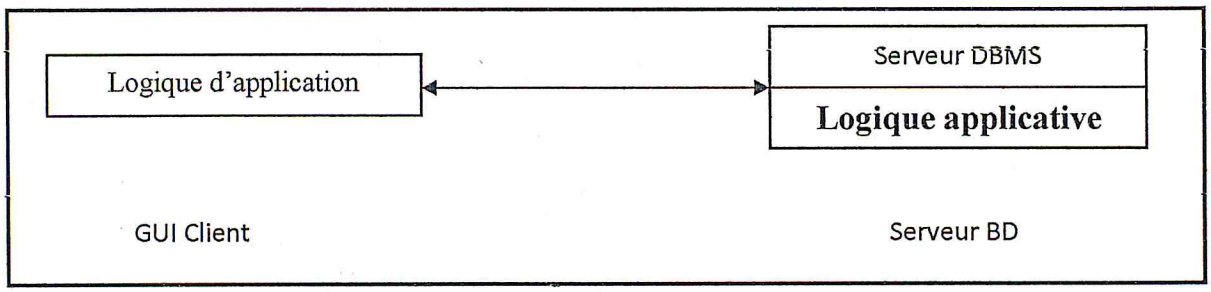


Fig-1.4 Logique applicative réside dans le Serveur

3.2 Le modèle Trois tiers :

Dans ce modèle un tiers est ajouté au milieu des deux tiers du modèle précédent (le client et le server SGBD) qui sera un serveur applicatif ou d'application responsable du lien entre le client et plusieurs serveurs de BD qui vont gérer la gestion d'accès BD.

Ce modèle est utilisé quand un serveur doit agir comme un client, la figure suivante montre ce cas :

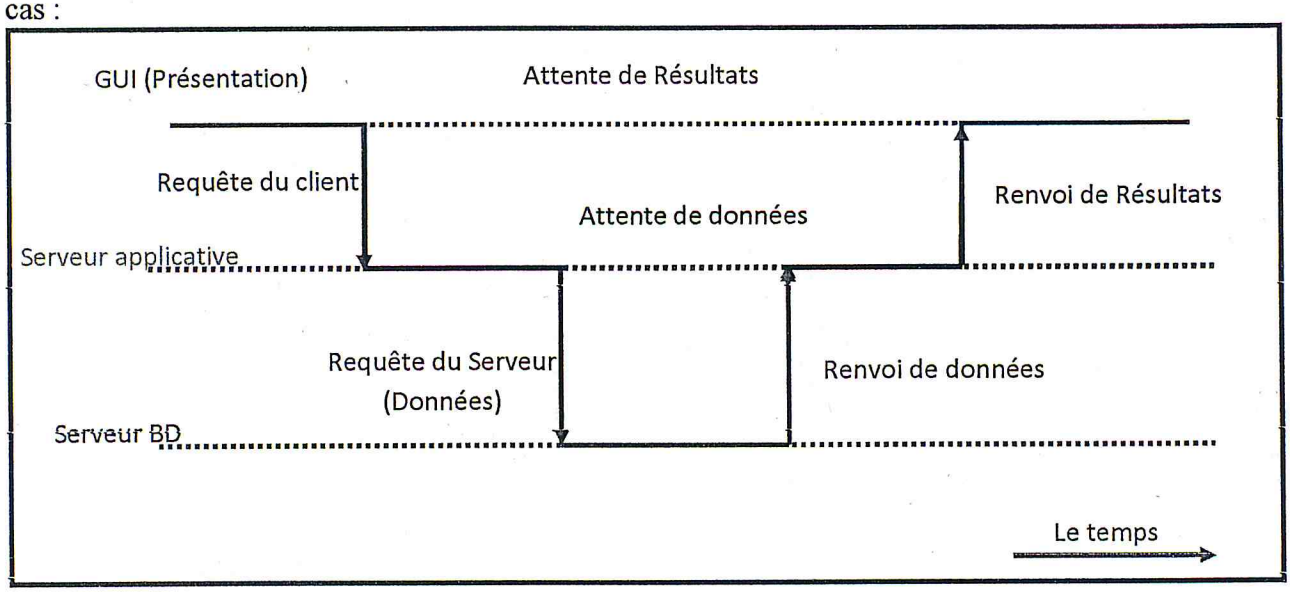


Fig-1.5 Exemple d'un serveur qui agit comme un client [DIS 06]

3.3 Le modèle N tiers (Multi-tiers):

Ce modèle oblige le développeur de grouper les entitées selon leur rôle, règles, relations et de distribuer la fonctionnalité en tiers qui va permettre de partager les ressources et les utiliser d'une meilleure façon.

L'approche la plus utilisée est de créer plusieurs modèles 3 tiers comme le montre la figure suivante :

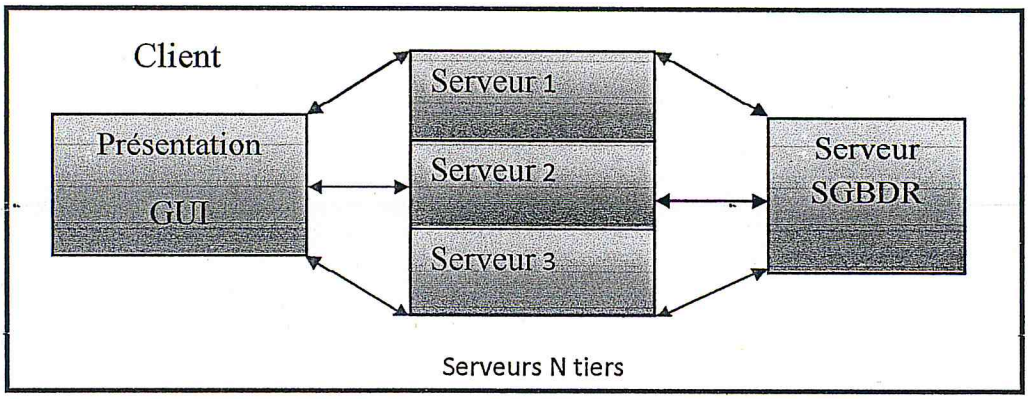


Fig-1.6 Modèle N tiers (Multi-tiers)

4. Evolution du modèle Client / Serveur

Un service est défini en 3 composants élémentaires : le *code*, *l'unité d'exécution* (processeur encapsulant la pile d'exécution et le compteur ordinal) et les *ressources* (applicables au code) qui représentent les données (bases de données, mémoires, fichiers…etc).Ces composants sont nécessaires pour l'exécution d'un service. Pour cela un client souhaitant utiliser un service précis doit identifier, trouver puis utiliser l'ensemble de ces 3 éléments.

Nous commençons par présenter les modèles d'exécution à distance, actuellement les plus répandus, où tous les éléments sont immobiles puis nous abordons des modèles utilisant la mobilité grâce aux déplacements d'un ou plusieurs composants d'un service.

4.1 Les différents types de requêtes

Une **requête** est un message transmis par un client à un serveur décrivant l'opération à exécuter pour le compte du client [GAR99]. Pour cela, nous disposons d'une couche système prenant en charge la localisation des sites, le transport fiable et efficace des communications entre deux partenaires distants.

Nous présentons ici les différents mécanismes (protocoles) permettant l'envoie de ces requêtes. Dans tous ces mécanismes, les trois éléments définis pour un service (code, ressource et UE) *restent concentrés sur le serveur*, la différence est simplement sur la manière d'envoyer la requête.

4.1.1 L'envoi de message

Cette méthode fût la première à être disponible lors de la construction des réseaux d'ordinateurs. Elle repose directement sur le service de communication offert par le réseau. Ce paradigme est réalisable à travers des bibliothèques de bas niveau généralement difficiles à mettre en œuvre [Rif98]. De plus, cette méthode n'est pas directement intégrée aux langages de programmation et nécessite de définir un protocole de communication stricte entre le serveur et ses clients. Ces deux contraintes ne permettent pas une programmation simplifiée aux développeurs. Un exemple d'application utilisant cette méthode est le serveur FTP (File Transfert Protocole) [Gie78].

4.1.2 Appel de procédure à distance (RPC)

Plusieurs langages de programmation offrent la possibilité de décomposer le code en modules (procédures) qui vont être appelés pendent l'exécution selon l'enchainement de l'algorithme. Ce mécanisme permet l'appel de procédure distante (classiquement locale). Cette méthode a été élargie en offrant la possibilité de réaliser des appels à distance grâce au mécanisme d'appel de procédure à distance ou RPC (Remote Procedure Call) [ICS09]. Ce mécanisme est une abstraction permettant aux développeurs de ne pas manipuler directement le service de communication réseau mais d'appeler une procédure distante comme si elle était locale i.e. il permet l'appel de procédure distante, en rendant transparent l'échange de messages entre le site appelant et le site appelé [DIS06].

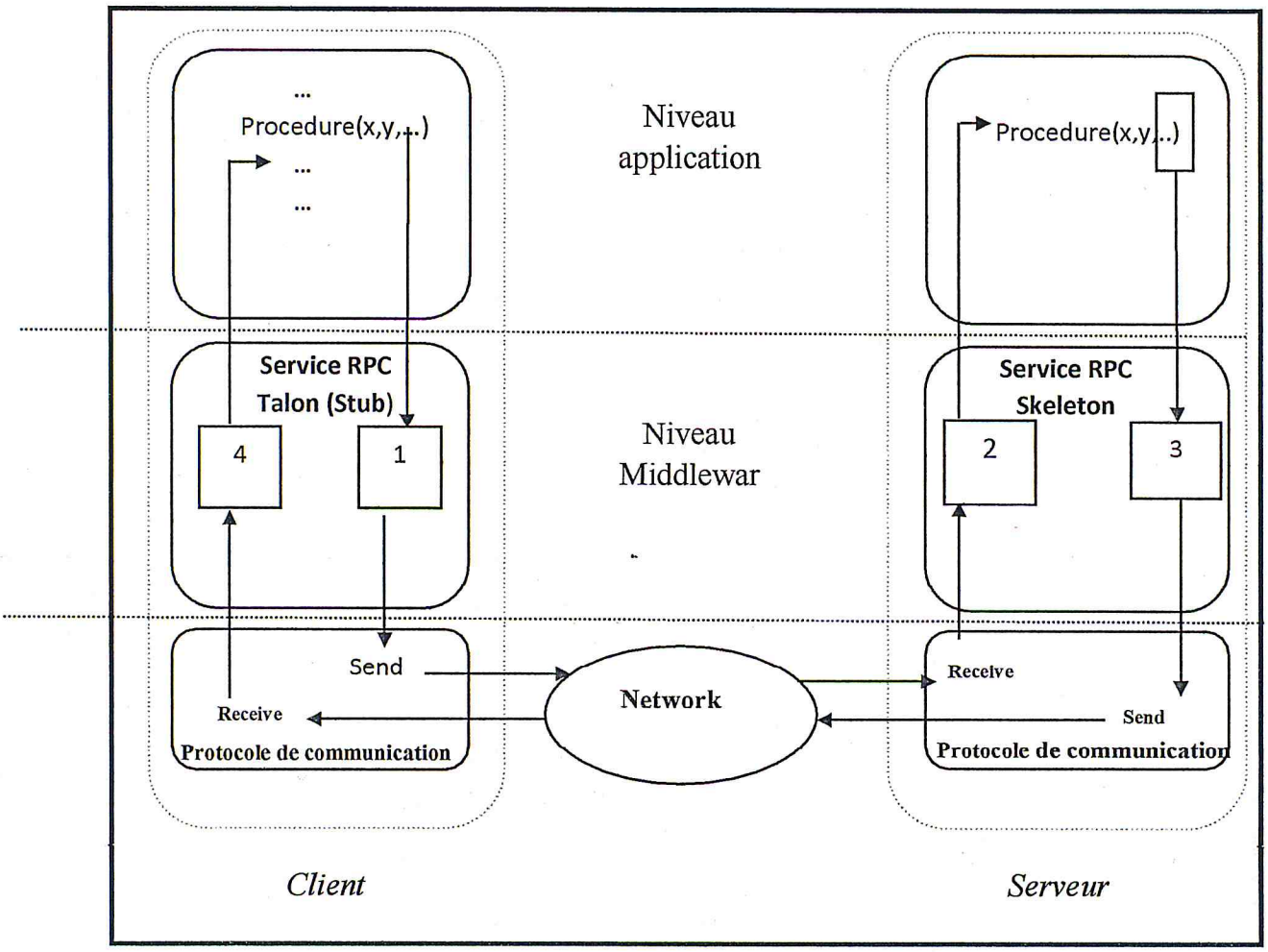


Fig-1.7 Fonctionnement du RPC

La procédure talon client est invoquée lors l'appel de la procédure distante par le client

L'ensemble des arguments est transmis au service RPC

Au point 1 :

- Le talon collecte les arguments et les assemble dans un message.
- Un délai de garde est armé.
- Un identificateur est généré pour RPC et joint au message.
- Le talon transmet les données au protocole de transport pour émission sur le réseau.

Du coté serveur

- Le protocole de transport délivre le message au service de RPC (talon serveur)

Au point 2 :

- Le talon désassemble les arguments.
- L'identificateur de RPC est enregistré.
- L'appel est ensuite transmis à la procédure distante requise pour être exécuté. Le retour de la procédure redonne la main au service de RPC et lui transmet les paramètres résultats (point3)

Au point 3 :

- Les arguments de retour sont empaquetés dans un message.
- Un autre délai de garde est armé.
- Le talon transmet les données au protocole de transport pour mission sur le réseau.

Du coté client

- L'appel est transmis au service de RPC.

Au point 4 :

- Les arguments de retour sont dépaquetés.
- Le délai de garde armé au point 1 est désarmé.
- Un message d'acquittement avec l'identificateur du RPC est envoyé au talon serveur (le délai de garde armé au point 3 peut être désarmé).
- Les résultats sont transmis à l'appelant lors du retour de procédure.

4.1.3 Appel de méthode à distance

Ce mécanisme se place à un niveau plus haut que celui du processus. A savoir la programmation à objet qui a enrichi celle procédurale ou structurée. Un objet est une entité logicielle encapsulant un ensemble de données et de méthodes (interface) i.e. un objet peut être vu comme étant une boite noire (utilisable à travers son interface) capable d'effectuer une tâche définie. Une application est alors un ensemble d'objets et d'interactions entre ces objets.

ORPC : (Object-oriented Remote Procedure Call) a enrichit l'RPC en permettant l'accès à des méthodes d'objet distant à la place de procédure distante. Un objet distant est un objet qui répond aux interactions déclenchées par des composants distants.

Les standards ORPC les plus communs sont :

1) Common Object Request Broker Architecture (CORBA)
2) Java Remote Method Invocation (RMI)
3) Distributed Component Object Model (DCOM)

4.1.4 Java RMI :

Remote Methode Invocation RMI est un ensemble de classes (API) permettant de manipuler des objets sur des machines distantes (objets distants) de manière similaire aux objets sur la machine locale (objets locaux).

C'est un peu du "RPC orienté objet". Un objet local demande une fonctionnalité à un objet distant. RMI apparaît avec Java 1.1 et est complètement intégré depuis Java 1.1 (mars 1997). Comme RMI permet la communication objet à objet entre 2 machines virtuelles Java (JVM), son utilisation requière une plateforme Java.

On dit généralement que RMI est une solution "tout Java", contrairement à la norme CORBA de l'OMG (Object Management Group) permettant de manipuler des objets à distance avec n'importe quel langage. CORBA est toutefois beaucoup plus compliqué à mettre en œuvre, c'est la raison pour laquelle de nombreux développeurs se tournent généralement vers RMI.

Architecture interne de RMI (structure des couches RMI)

Les connexions et les transferts de données dans RMI sont effectués par Java sur TCP/IP grâce à un protocole propriétaire (JRMP, Java Remote Methode Protocol) sur le port 1099. A partir de Java 2 version 1.3, les communications entre client et serveur s'effectuent grâce au protocole RMI-IIOP (Internet-Orb protocol), un protocole normalisé par OMG et utilisé dans l'architecture CORBA [SUN04].

La transmission de données se fait à travers un système de couches, basées sur le *modèle OSI* afin de garantir une interopérabilité entre les programmes et les versions de Java.

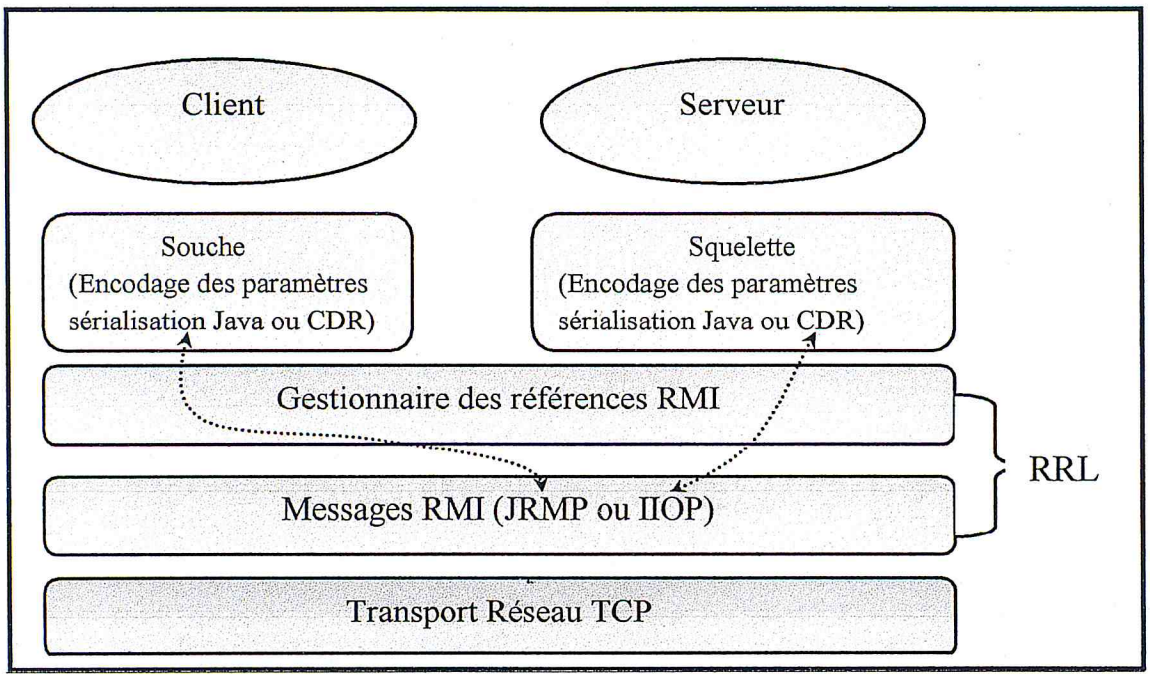


Fig-1.8 architecture interne du RMI

- Souche/squelette : encode/décode les paramètres des méthodes
- Gestionnaire de références (RRL, Remote Reference Layer): associe les mandataires (Remote) aux références distantes + ramasse-miettes réparti. Il est chargé du système de localisation afin de fournir un moyen aux objets d'obtenir une référence à l'objet distant. Elle est assurée par le package *java.rmi.Naming*. On l'appelle généralement **registre RMI** car elle référence les objets.
- Message : définit la structure des messages, leurs identifiants et l'interaction requête/réponse.
- Transport : transporte un message entre deux machines virtuelles Java. Cette couche permet d'écouter les appels entrants ainsi que d'établir les connexions et le transport des données sur le réseau par l'intermédiaire du protocole TCP. Les packages *java.net.Socket* et *java.net.SocketServer* assurent implicitement cette fonction.

Ainsi, une application client-serveur basée sur RMI met ainsi en œuvre trois composantes :

- Une application cliente implémentant le stub
- Une application serveur implémentant le skeleton (*squelette*)

- Une application médiatrice (le registre RMI) servie par un processus tiers (*rmiregistry*)

Fonctionnement d'ensemble de Java RMI

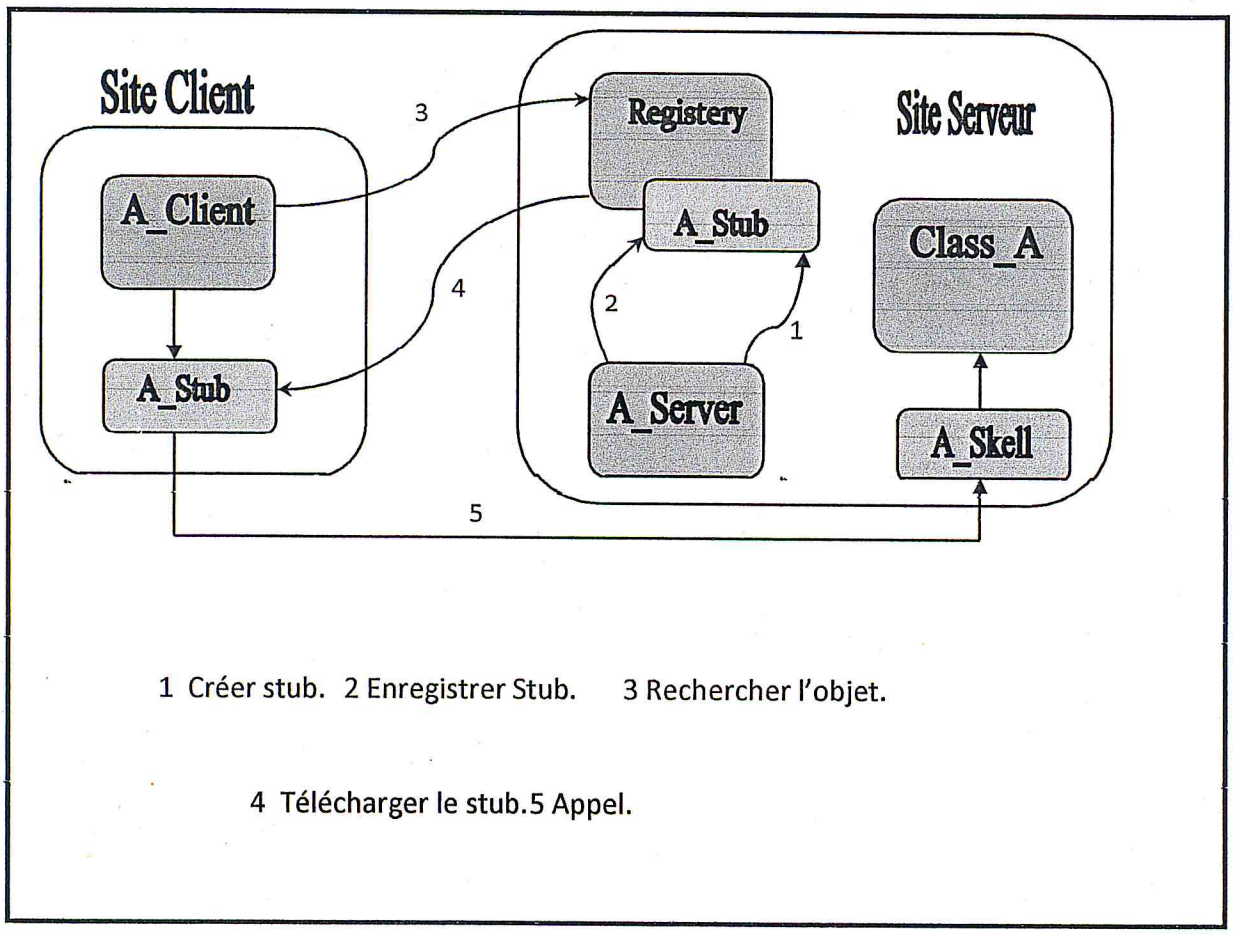


Fig-1.9 Fonctionnement de RMI

Soit un objet instancié sur une machine cliente et désirant accéder à des méthodes d'un objet distant. Il effectue les opérations suivantes :

Considérant que l'objet distant a été crée, son talon (stub) a été défini et enregistrer dans registre RMI.

1. il localise l'objet distant grâce à un service de désignation : le **registre RMI**
2. il obtient dynamiquement une image virtuelle de l'objet distant (appelée **stub** ou *souche* en français). Le stub possède exactement la même interface que l'objet distant.

3. Le stub transforme l'appel de la méthode distante en une suite d'octets, c'est ce que l'on appelle la sérialisation, puis les transmet au serveur instanciant l'objet sous forme de flot de données. On dit que le stub *"sérialise"* les arguments de la méthode distante.
4. Le **squelette** instancié sur le serveur "dé-sérialise" les données envoyées par le stub, puis appelle la méthode en local
5. Le squelette récupère les données renvoyées par la méthode (type de base, objet ou exception) puis les sérialise.
6. Le stub dé-sérialise les données provenant du squelette et les transmet à l'objet faisant l'appel de méthode à distance.

Objet local et objet distant

Comme la distinction entre les objets distants et locaux est difficile au niveau du langage de programmation, Java peut cacher les différences même au niveau de RMI donc une sérialisation est requise. La seule **déférence** qu'on peut faire entre un objet local et un autre distant pendant un RMI c'est que les objets locaux sont passés par valeur et que les objets distants sont passés par références.

Appel RMI avec échange d'objet distant et d'objet ordinaire

Cet exemple est destiné à montrer la différence entre les objets "ordinaires" (i.e. non-distants) et les objets "distants" ("Remote") lorsque des objets sont passés en argument ou en valeur de retour lors d'un appel RMI.

5. Introduction à la Mobilité

Nous présentons ici une autre manière de concevoir les interactions de type client/serveur en introduisant la mobilité qui peut être supportée par une nouvelle famille de langages. Cette Mobilité peut être vue comme une variation du modèle clients/serveur classique.

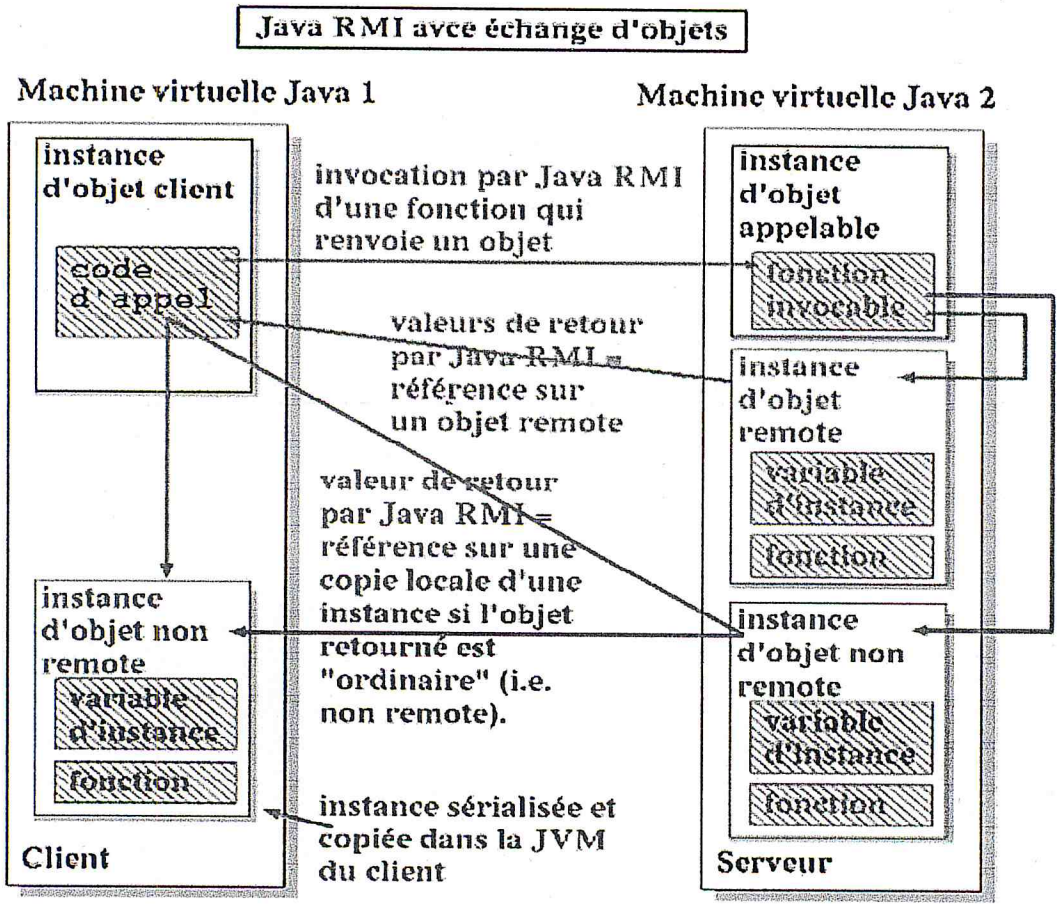


Fig-1.10 RMI avec échange d'objets

Nous présentons les différentes formes existantes, on se basant sur le déplacement d'un ou plusieurs éléments composant le schéma client/serveur, i.e. le code, les ressources ou l'unité d'exécution. Nous prenons comme référence le placement de chacun de ces éléments avant et après l'exécution du service.

Le terme « *code mobile* » réfère à du code qui peut être envoyé d'une machine pour être exécuter sur une autre machine [Cons 01]

4.1 L'exécution à distance (REV, Remote Evaluation) :

Dans une interaction par évaluation distante un client envoi un code à un site distant. Le site récepteur utilise ses ressources pour exécuter le programme envoyé (Le code de l'application). Une interaction additionnelle délivre ensuite les résultats au client ou le site émetteur. Dans ce cas seul le code est transmis au serveur et l'exécution du code se déroule uniquement sur ce dernier. L'UE (l'unité d'exécution) et les ressources sont fixes seul le savoir faire est envoyé.

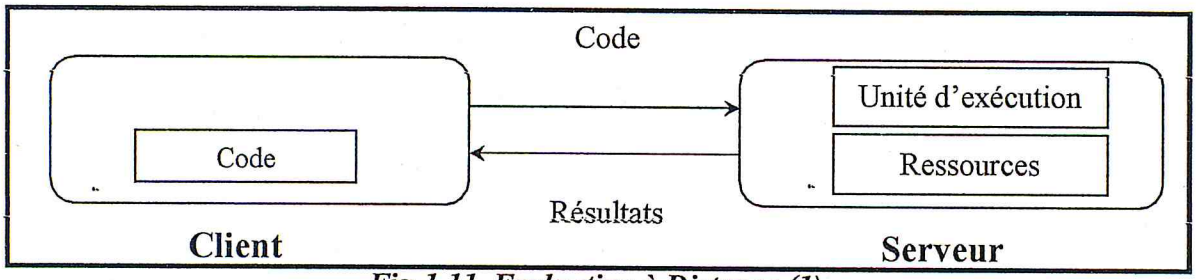


Fig-1.11 Evaluation à Distance (1)

Le code d'une requête SQL émis vers un serveur de base de données représente un exemple d'évaluation distante.

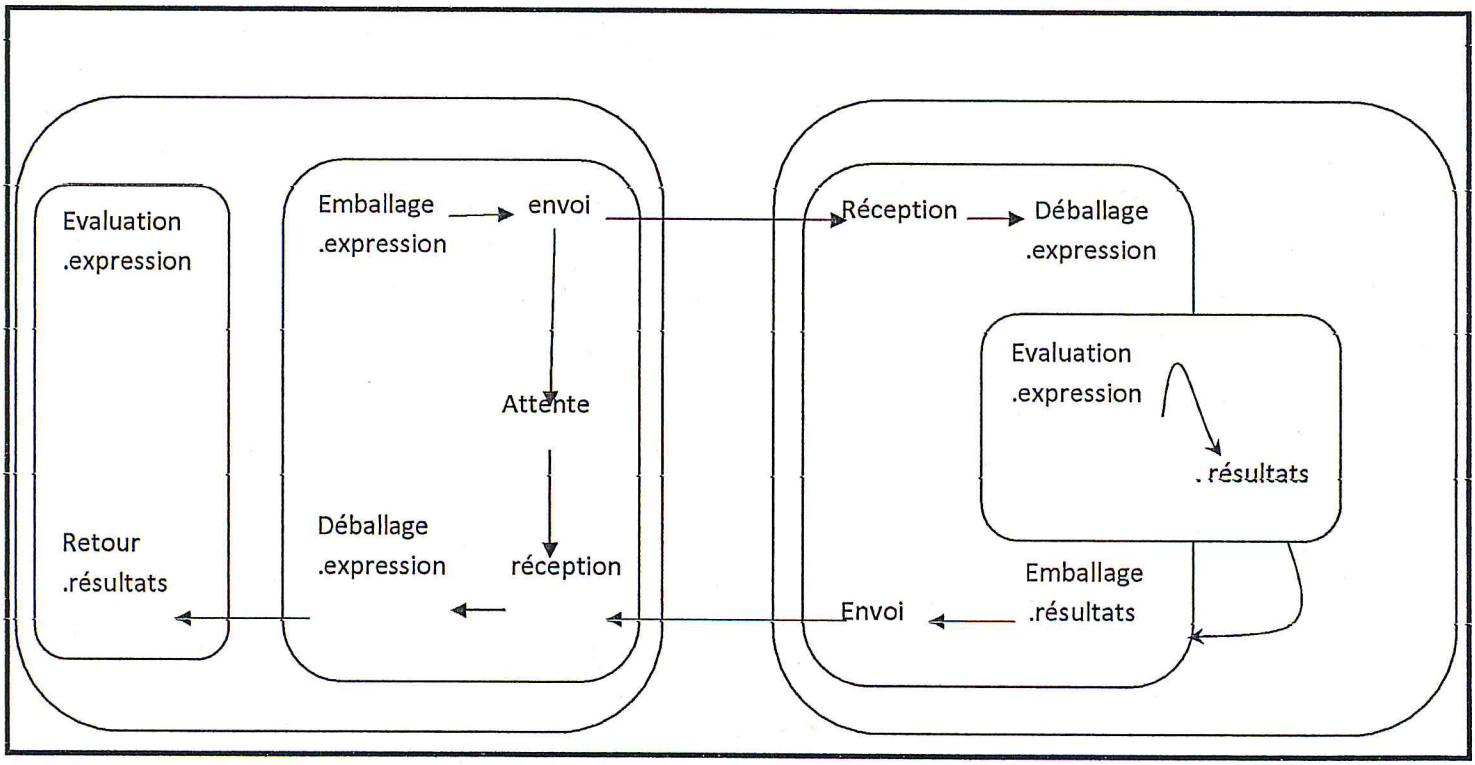


Fig-1.12 Evaluation à Distance (2)

Chapitre 2 :

Les Systèmes Multi-Agents

Introduction :

Dans le chapitre précédent on a vu l'évolution de client serveur qui est la colonne vertébrale des applications reparties vers la technologie des agents qui représentent la forme la plus évoluée de la migration du code.

Dans ce chapitre nous discutons la technologie des Agents, qui fait partie des aspects les plus importants du domaine du code mobile.

1) Les Agents :

Toutes les définitions conviennent qu'un agent est essentiellement un composant logiciel spécial qui a l'autonomie qui fournit une interface interopérable à un système arbitraire et/ou se comporte comme un agent humain, fonctionnant pour quelques clients dans la poursuite de son propre agenda. Ces agents peuvent agir l'un sur l'autre les uns avec les autres indirectement (par l'action sur l'environnement) ou directement (par l'intermédiaire de la communication et de la négociation).Les agents peuvent décider de coopérer pour le bénéfice mutuel ou peuvent concurrencer pour servir leurs propres intérêts. [JADE07]

1.1 Caractéristiques d'un Agent

Il est important de déterminer les différentes caractéristiques que doit posséder un agent et qui font la différence avec les programmes conventionnels. En effet, dépendamment des situations et des circonstances, un agent peut être fortement caractérisé par un sous-ensemble des propriétés indiquées ci-dessous et être difficilement associable aux autres spécifications.

1.2 Autonomie :

Un agent est autonome, parce qu'il est capable d'agir sans l'intervention direct d'un tiers (Humain ou agent) et contrôle ses propres actions ainsi que son état interne. [JADE 07]

1.3 Réactivité :

C'est la capacité qu'un agent puisse percevoir les changements environnementaux et modifier son comportement en élaborant des réponses *dans les temps requis*. [JADE 07] i.e. il répond en temps réel aux changements de l'environnement.

1.4 Pro-activité :

Un agent n'agit pas simplement en réponse à l'environnement, mais peut prendre l'initiative pour atteindre un but. [TOOL 05]

1.5 Sociabilité :

Un agent est capable d'interagir avec d'autres agents (négocier, coopérer, collaborer...etc.), [TOOL 05] pour minimiser les redondances, résoudre des problèmes ou effectuer des tâches.

1.6 Communication :

Un agent peut communiquer avec d'autres agents en échangeant une séquence de messages par un langage de communication que d'autres comprennent. Le domaine du discours est décrit par son ontologie. [IGI 07]

1.7 Adaptabilité :

Un agent peut avoir la capacité d'apprendre les nouvelles informations sur l'environnement dans lequel il est déployé et améliorer dynamiquement son propre comportement. [IGI 07] i.e. qu'il peut changer son comportement en se basant sur ses expériences passées (sur ce qu'il a appris)

1.8 Rationalité [AMA 03]:

Pour chaque perception possible, un agent rationnel doit sélectionner une action qui est sensée maximiser ses mesures de performance, basées sur la séquence perçue et la connaissance intégrée de l'agent. I.e. la rationalité maximise les performances prévues. La rationalité dépend de quatre points :

1- la mesure d'exécution qui définit le critère des succès.

2- la connaissance antérieure de l'agent de l'environnement.

3- les actions que l'agent peut effectuer.

4- l'ordre du percept de l'agent jusqu'ici.

1.9 Mobilité:

Un agent peut migrer d'un système à l'autre d'une manière prédéterminée ou à sa propre discrétion. En conséquence, les agents peuvent être statiques ou mobiles. [IGI 07]

2) Agent mobile

2.1 Définition :

Un Agent mobile est programme qui migre à sa propre initiative d'une machine à une autre dans un réseau hétérogène. Dans chaque machine l'agent interagi avec les services des agents stationnaires et d'autres ressources afin d'accomplir une tache particulière, définie par son propriétaire. [Sec 06]

2.1.1 Agent mobile du point de vue intelligence artificielle :

Le terme d'agent mobile est issu principalement de deux domaines distincts : l'intelligence artificielle et les systèmes distribués avec la migration de processus. L'agent mobile est alors défini comme un agent ayant la capacité de se déplacer de site en site et ayant la conscience de son déplacement. Tel qu'au domaine de l'intelligence artificielle où la programmation tente d'imiter l'esprit humain lors de l'exécution [AS92], on désigne par agent cognitif, un programme qui possède une « intelligence » assez développée pour prendre des décisions face

à des situations complexes. Dans ce contexte, l'agent est isolé, i.e. il n'a aucune interaction avec tout autre agent, et il se contente d'itérer seul son algorithme afin de résoudre le problème qui lui est posé.

Les difficultés rencontrées par les agents cognitifs pour résoudre des problèmes complexes et distribués, à cause de leur approche centralisée principalement, ont poussé à repenser leur fonctionnement en distribuant l'intelligence dans différents éléments communiquant entre eux [San04]. C'est de cette idée d'où viennent les ***systèmes multi-agents***.

2.2 Composants et attributs d'un agent mobile :

D'un point de vue architectural, l'agent mobile est composé d'un code (statique) et d'une configuration courante (Données et Etat) qui inclut des structures de données globales, tas (heap), pile et informations de control (conteur ordinal) [SEC06]. En outre l'agent possède un identifiant, une interface, des informations sur son conteneur (container, c'est agent server qui contrôle l'exécution des agents et leur fournit le minimum des services : communication, migration et sécurité) et des informations concernant son propriétaire (Owner) qui servent à déterminer la loyauté de cet agent. [TOOL 05]

Le code [TOOL05]: c'est la logique de l'agent. Il contient les différentes fonctionnalités de l'agent. Les agents de même type ont le même code. Ce code doit être identifiable, lisible et surtout facilement séparable de sa plateforme locale pour pouvoir être éventuellement transférer à une plateforme distante. Pour cela il est généralement sous forme de code interprété.

Généralement, le code d'un agent se compose de plusieurs fichiers. Cela dépend du langage de programmation dont-t-il à été écrit (ex. En JAVA le code peut être sous forme de plusieurs classes chacune dans un fichier i.e. plusieurs fichiers JAR)

L'état [SECM06] [TOOL05]: l'état général de l'agent mobile est constitué de deux parties

L'état de données : aussi appelé l'état d'objet, il enregistre les valeurs de toutes les variables locales et globales utilisées ou peut être produites par l'agent durant son cycle de vie. Il est important de noter ici que les données accessibles par un agent ne font pas toutes partie de l'état d'objet car certaines d'entre elles peuvent être partagées (ex. les gestionnaire de fichiers, des threads, des interfaces utilisateur... etc.).

L'état d'exécution : il enregistre l'état de tout les processus et threads. La différence entre cet état et le précédent est que les informations de l'état d'objet sont contrôlées par l'agent lui même, par contre celles de l'état d'exécution sont contrôlées par le processeur ou le système d'exploitation.

L'interface : un agent mobile est appelé à interagir avec d'autres agents et d'autres systèmes, pour cela l'agent est doté d'un ensemble de méthodes constituant son interface.

L'identifiant : à sa création l'agent est doté d'un identifiant immutable. Il est nécessaire pour identifier et localiser l'agent d'une manière unique dans la plateforme (ensemble de conteneurs). L'identifiant est, dans certains cas, choisi par le développeur d'agent, dans d'autres cas imposé par le conteneur (les agents midleware)

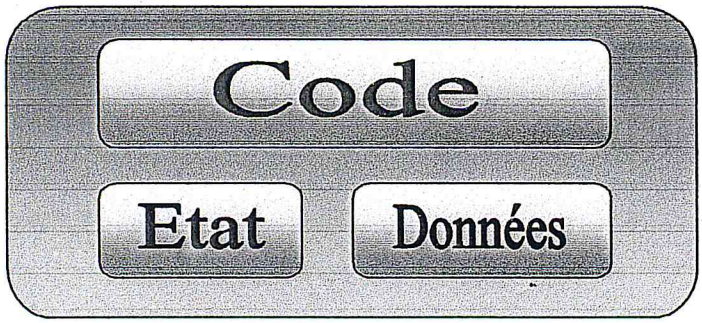


Fig-2.1 Structure d'un agent mobile

2.3 Avantages et inconvénients des Agents Mobiles [JADE 07] [TOOL 05] [SecM 06]

Il y a eu beaucoup de débats sur les divers avantages et inconvénients des agents mobiles, habituellement en comparaison de leurs cousins immobiles.

Quelques avantages typiques sont:

- ***Délégation des tâches :***

Puisque les agents mobiles sont simplement un type plus spécifique d'agent logiciel, un utilisateur peut utiliser un agent mobile comme représentant auquel il peut déléguer des tâches. Au lieu d'utiliser des systèmes comme outils interactifs qui ne peuvent fonctionner que sous le control *direct* de l'utilisateur, les agents logiciels autonomes visent à suivre toutes les tâches et travailler sans contact ou contrôle permanent. En conséquence, l'utilisateur peut consacrer du temps et de l'attention à d'autres choses plus importantes. Ainsi, les agents mobiles sont de bons moyens pour faire face à la surcharge régulière de l'information que nous éprouvons.

- ***Traitement asynchrone et indépendant :*** Une fois qu'ils ont émigré à une nouvelle plate-forme, les agents n'ont pas à contacter leur propriétaire afin d'effectuer leur tâche. Ils

peuvent seulement devoir renvoyer les résultats (C'est utile quand on considère les dispositifs mobiles avec des ressources limitées). Un agent peut être migrée vers une autre machine pour accomplir des tâches complexes et retourner périodiquement des résultats. Ces caractéristiques permettent aux agents mobiles d'être appropriés au « Nomade computing », aussi dans l'utilisation des dispositifs mobiles à ressources limitées. i.e. un utilisateur peut lancer ses agents depuis un dispositif mobile qui offre une connexion réseau limitée et volatile, car l'agent est peu dépendant du réseau, et ça sera plus stable que les applications Client/Serveur.

- ***Tolérance aux fautes physiques :***

En se déplaçant avec leur code et leurs données propres, les agents mobiles peuvent s'adapter facilement aux erreurs systèmes. Ces erreurs peuvent être d'ordre purement physique, disparition d'un nœud par exemple, ou d'ordre plus fonctionnel, arrêt d'un service par exemple. Si on prend le cas d'un site perdant une partie de ses fonctionnalités, un service tombant en panne, l'agent pourra alors choisir de se déplacer vers un autre site contenant la fonctionnalité désirée. Ceci permet une meilleure tolérance aux fautes que le modèle statique classique.

- ***Diminution de l'utilisation du réseau :***

Le déplacement des agents mobiles permet de réduire significativement les communications distantes entre les clients et les serveurs. En privilégiant les interactions locales, l'utilisation du réseau va se limiter principalement au transfert des agents.

Mais les agents mobiles ont également quelques inconvénients.

Comme décrit dans la MIR (2004), les plus pertinents sont :

- ***Scalabilité et performance:***

Quoique les agents mobiles réduisent la charge du réseau, ils tendent également à augmenter la charge du traitement. C'est parce qu'ils sont habituellement programmés avec des langues interprétés et doivent également souvent observer les normes rigoureuses d'interopérabilité qui peuvent encourir des frais généraux de traitement.

- ***Portabilité et standardisation :***

Les agents ne peuvent pas interopérer s'ils ne suivent pas des normes communes de communication. L'adoption de ces normes, telles qu'OMG MASIF (Mobile Agent System Interoperability Facility) ou le FIPA est en général nécessaire, particulièrement pour la mobilité inter-plateforme.

- ***Sécurité :***

L'utilisation des agents mobiles peut provoquer des problèmes de sécurité. N'importe quel code mobile offre une menace potentielle et devrait être soigneusement authentifié avant l'invocation.

La sécurité consiste à empêcher des accès, ou des modifications, non-autorisés aux éléments d'un système informatique qui ne doivent pas être modifiés par des utilisateurs non-autorisés.

Un agent mobile peut être cible de plusieurs types d'attaques du fait de son déplacement à travers des milliers de sites, qui peuvent ne pas être de confiance, et à travers un réseau qui n'est pas toujours sécurisé. En effet, au cours de son déplacement, l'agent interagit avec d'autres agents qui peuvent analyser ou altérer son contenu.

2.4 Domaines d'application

Il existe plusieurs applications pour lesquelles les agents mobiles pourraient être utilisés :

2.4.1 Recherche d'information :

Le domaine le plus populaire d'application d'agent mobile. A la place de déplacer une large base de données on envoie un agent mobile afin de récupérer l'information.

Etant donné le volume important (et en croissance) d'informations disponibles sur l'Internet, la collecte d'informations exige la recherche dans un grand espace de données pour déterminer des portions d'informations pertinentes. Le filtrage des informations non-pertinentes est souvent un processus consommateur de temps.

Les agents mobiles peuvent visiter plusieurs sites web, coopérer et trouver les sites d'intérêts et revenir avec les meilleurs résultats.

2.4.2 Le Commerce électronique :

Ce domaine a simplement traduit le commerce en monde réel en données et des processus électronique Un agent mobile peut être utilisé pour faire des achats (shopping) comme un humain, il peut visiter des magasins, comparer les prix et même faire des enchères

2.4.3 Négociation :

Au lieu de chercher dans des bases de données ou dans des fichiers, les agents peuvent obtenir de l'information en interagissant avec d'autres agents. Si par exemple une personne désire organiser une rencontre avec diverses personnes, elle pourrait envoyer son agent mobile pour interagir avec les agents représentatifs de chacune de ces personnes. Les agents pourraient négocier et décider du temps et du lieu de la rencontre selon les contraintes de chaque

personne. Ces contraintes sont exprimées dans un langage de spécification compréhensible aux agents.

Ainsi que d'autres domaines d'application, comme la surveillance des réseaux et le traitement parallèle.

3) SYSTEMES MULTI AGENTS- SMA - :

3.1 Définitions d'un Système Multi-Agents

3.1.1 Définition 1[FERB 95] :

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

- ✓ Un environnement E, c'est-à-dire un espace disposant généralement d'une métrique.
- ✓ Un ensemble d'objets O. Ces objets sont situés, c'est-à-dire que, pour tout objet, il est possible, à un moment donné, d'associer une position dans E. Ces objets sont passifs, c'est-à-dire qu'ils peuvent être perçus, créés, détruits et modifiés par les agents.
- ✓ Un ensemble A d'agents, qui sont des objets particuliers (A _ O), lesquels représentent les entités actives du système.
- ✓ Un ensemble de relations R qui unissent des objets (et donc des agents) entre eux.
- ✓ Un ensemble d'opérations Op permettant aux agents de A de percevoir, produire, consommer, transformer et manipuler des objets de O.
- ✓ Des opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification, que l'on appellera les lois de l'univers.

3.1.2 Définition 2 :

Un Système Multi-Agents (SMA)-(MAS) Multi Agent System, en Anglais- est une société d'agents coexistants. Chaque agent interagit avec les autres agents et avec l'environnement. Ces agents, selon certaines règles et certaines organisations, coordonnent et changent leurs comportements pour atteindre le but du système [IGI 07]. Les interactions entre ces agents peuvent être coopératives ou égoïstes. C'est-à-dire, les agents peuvent partager un objectif commun (comme dans une colonie de fourmis), ou ils peuvent poursuivre leurs propres intérêts (comme dans l'économie de marché) [CKP09]. Ces agents ont un niveau élevé d'autonomie, hétérogénéité et parallélisme. Les propriétés qui les qualifient d'excellents candidats pour l'élaboration des blocs de programmation, parallèle ou distribuée, concurrente. [IGI 07]

3.1.3 Définition 3

Les Systèmes Multi agents (SMA) sont des applications basées sur des entités logicielles encapsulées et autonomes qui peuvent flexiblement atteindre leurs objectifs en interagissant les uns sur les autres selon des protocoles et langages d'interaction de haut niveau. Les agents équilibrent leur comportement réactif en réponse aux influences de l'environnement avec leur comportement proactif vers l'accomplissement des objectifs de conception.

3.2 Les Caractéristique d'un System Multi-Agents [GutMich 04] [HBMAS 09] :

Un Système Multi-Agents est caractérisé par :

- Un SMA peut être composé d'un seul système ou de plusieurs Sous-systèmes.
- Un agent est responsable de son niveau de sécurité.
- Un agent peut communiquer avec n'importe quel agent dans le Système.
- Un agent fourni certains nombres de services, qui sont disponibles aux autres agents dans le système.
- Un agent « sait » quels sont les agents qui vont interagir avec lui.
- Il possède des informations ou capacités de résolution des problèmes limitées on dit qu'il a une vue partielle
- Les agents devront interagir pour coopérer (contrôle) Collaborer (allocation de tâches) Négocier (résolution de conflits) se coordonner (synchronisation)
- Chaque agent a des ressources limitées pour résoudre un problème donné
- Chaque agent contient (Dans son nom) la manière dont il peut être utilisé de l'extérieur (la notion « Agent ID » très connue par les développeurs des SMAs)
- L'absence de système de contrôle global.
- Les Données sont décentralisées et le calcul est asynchrone.

Un SMA peut-être :

1) Ouvert / Fermé :

Les agents y entrent et en sortent librement /l'ensemble d'agents reste le même

2) Homogène / Hétérogène :

Tous les agents sont issus du même modèle / des agents de Modèles différents, de granularités différentes (ex: un écosystème)

3) Mixte (ou non) :

Les agents « humains » sont partie intégrante du système

3.3 Avantages et inconvénients des SMA

3.3.1 Les avantages des SMA [Ferb 95] [IGI 2007]

Plusieurs avantages sont attribuables aux SMA :

1) Quelques problèmes sont distribués par nature et demandent à être résolus par plusieurs agents et c'est le cas dans le e-business et le e-science.

2) Puisque plusieurs agents peuvent travailler en parallèle, les SMA peuvent améliorer leur performance par un travail parallèle des différents agents.

3) La facilité de « changement d'échelle d'une architecture » (*scalability*) car plusieurs agents peuvent s'ajouter ou se retirer dynamiquement d'un système.

4) La configuration automatique d'un SMA grâce à la plus grande « proximité » du monde réel des agents.

5) Les SMA peuvent améliorer la productivité de chacun des agents.

6) La réduction des coûts de développement de logiciel : plus un logiciel est modulaire, plus la complexité et les coûts de développement diminuent.

7) Flexibilité des systèmes : les systèmes sont composés de plusieurs agents ayant des habiletés différentes, ils peuvent donc résoudre différents problèmes (ils peuvent aussi résoudre des problèmes en coopérant et en se partageant les tâches dépendamment des capacités de chacun).

8) Les caractéristiques des SMA rendent les comme meilleur candidats pour la IT-Security.

9) Les SMA peuvent Améliorer l'utilisation des ressources. En gérant mieux l'utilisation de certaines ressources par collaboration.

3.3.2 Les inconvénients des SMA :[Ferb 95] [GutMich 04]

Malgré tous les avantages des SMA ils ont quelques inconvénients tels que :

1. La sécurité des applications : La possibilité que tous les agents peuvent communiquer sans aucun contrôle externe pose des problèmes de sécurité (un agent peut agir facilement comme un pirate et utiliser le système d'une manière frauduleuse).

2. La Modularité : Dans un programme classique, il existe des entités qui sont groupées dans des « packages » ou des modules. Ces Modules peuvent être visibles ou pas par d'autres modules on les appelle modules privés. Ce concept n'existe pas dans les SMA, tous les agents sont accessibles par tous. Donc il faut grouper les agents actifs qui travaillent ensemble dans

un groupe et les rendre capables de modifier leur groupement dynamiquement selon des règles générales.

3. L'exécution des agents s'effectuant en parallèle (que ce parallélisme soit simulé sur une machine monoprocesseur ou effectif sur une machine multiprocesseur) il est encore plus difficile de comprendre leur fonctionnement à partir de leur code, du fait du non-déterminisme inhérent au parallélisme.

4. Enfin, et surtout, les systèmes multi-agents sont des logiciels complexes, difficiles à appréhender et à concevoir. Il est donc nécessaire de réduire leur complexité en dégageant leur structure et en analysant séparément leurs différents composants sans pour autant perdre de vue l'organisation générale.

3.4 Les architectures des SMA :

3.4.1 Les architectures à base de tableaux noirs :

L'architecture de tableau noir est l'une des plus utilisée dans les systèmes multi-agents Cognitifs symboliques et elle a donné lieu à une abondante littérature. Originellement développée dans le cadre de l'intelligence artificielle traditionnelle (c'est-à-dire, du point de vue de l'IAD, pour réaliser des systèmes mono-agents), pour la reconnaissance de la parole avec le système Hearsay II (Erman et al. 1980), l'architecture de tableau noir s'est rapidement imposée en IAD comme une architecture suffisamment souple et puissante pour pouvoir implémenter les mécanismes de raisonnement et de calculs intervenant à l'intérieur des agents, notamment avec le système DVMT (Lesser et Corkill 1983).

Le modèle de tableau noir est fondé sur un découpage en modules indépendants qui ne communiquent directement aucune information, mais qui interagissent indirectement en partageant des informations. Ces modules, appelés sources de connaissance ou KS (pour Knowledge Sources), travaillent sur un espace qui comprend tous les éléments nécessaires à la résolution d'un problème. L'architecture d'un système à base de tableau noir comprend trois sous-systèmes

- ✓ Les sources de connaissance (KS).
- ✓ La base partagée (le "tableau" proprement dit) qui comprend les états partiels d'un problème en cours de résolution, les hypothèses et les résultats intermédiaires et d'une manière générale toutes les informations que s''echangent les KS. Ces bases sont décomposées en hiérarchies (conceptuelles, partitives, causales etc.) qui structurent la modélisation du domaine d'application comme l'espace des hypothèses/solutions.

✓ Un dispositif de contrôle qui gère les conflits d'accès entre les KS, ces derniers intervenants de manière "opportuniste" c'est-à-dire sans être déclenchés effectivement par un système centralisé de contrôle. C'est cette partie qui a connu le plus de modifications au cours de l''evolution de ces architectures. [Ferb 95]

Les principaux désavantages de ce type d'architecture sont:

➢ L'inefficacité des systèmes et leur manque de tolérance aux fautes.

➢ Comme toutes les données sont gardées sur la même machine, le manque d'espace mémoire devient souvent un problème.

➢ D'un autre côté, lorsque plusieurs agents tentent d'accéder aux mêmes données, un problème de partage des ressources est inévitable.

3.4.2 Les systèmes hiérarchisés (ou horizontaux)

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

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

3.4.3 L'approche totalement distribuée

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

Les avantages de cette approche sont :

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

3.5 Interaction & communication dans les SMA

3.5.1 Introduction

Une des principales propriétés de l'agent dans un SMA est celle d'interagir avec les autres agents. Ces interactions sont généralement définies comme toute forme d'action exécutée au sein du système d'agents et qui a pour effet de modifier le comportement d'un autre agent. Elles permettent aux agents de participer à la satisfaction d'un but global. Cette participation permet au système d'évoluer vers un de ses objectifs et d'avoir un comportement intelligent indépendamment du degré de complexité des agents qui le composent.

En général, les interactions sont mises en œuvre par un transfert d'informations entre agents ou entre l'environnement et les agents. La communication est souvent l'un des moyens utilisés pour échanger des informations entre agents. Donc les communications dans les SMA, comme chez les humains, sont à la base des interactions et de l'organisation des agents. Nous distinguons essentiellement deux modes de communication: la communication indirecte qui est une communication par signaux via l'environnement, et la communication directe qui procède à un échange de messages entre les agents.

Le mode de communication par échange de messages entre les agents permet à un agent de planifier un acte de communication envers un autre agent, on parle alors de communication intentionnelle. Cette approche s'est inspirée de l'interaction sociale telle que nous la trouvons dans d'autres contextes comme la communication entre les humains.

L'interaction inter-agents regroupe et combine plusieurs types de messages. Cette combinaison se traduit par des enchaînements conversationnels qui peuvent être modélisés par deux approches :

– Dans la première approche, l'agent construit son propre plan de communication au moment où il en a besoin pour réaliser sa tâche. Ainsi, l'interaction n'est pas définie a priori, elle est émergente. Les connaissances et les buts des agents régissent l'interaction en spécifiant le message à envoyer (acte de communication, agents destinataires, le contenu du message,...).

Cette approche donne à l'agent plus de flexibilité dans son interaction. Cependant, dans cette approche l'agent doit avoir suffisamment de connaissances sur la sémantique des

messages et doit être doté d'un mode de raisonnement lui permettant de mener ses interactions.

– Dans la deuxième approche, l'interaction se conforme à un enchaînement de messages spécifié à l'avance. Les règles qui régissent l'échange de messages forment le *protocole d'interaction.* De plus, le protocole d'interaction définit le type des messages qui doivent être échangés. L'agent interactif doit se conformer aux règles de conversation dictées dans le protocole. Comparée à la première approche, l'interaction basée sur les protocoles d'interaction ne nécessite pas d'architecture complexe au sein de l'agent [GB99a]. Ainsi, un protocole d'interaction spécifie un ensemble limité de réponses possibles pour un type spécifique de messages.

3.5.2 Langage de communication :

L'intérêt des langages de communication est de faciliter l'échange et l'interprétation des messages et l'interopérabilité entre les agents. Ces langages se focalisent essentiellement sur la manière de décrire exhaustivement des actes de communication d'un point de vue syntaxique et sémantique.

[FFMM94] déclarent que les langages de communication entre les agents doivent satisfaire certaines propriétés :

- ✓ une forme déclarative,
- ✓ une syntaxe simple et lisible,
- ✓ faire la distinction entre le langage de communication et le langage de contenu. Le langage de communication décrit les actes de langage et le langage du contenu permet d'exprimer les informations contenues dans le message,
- ✓ avoir une sémantique du langage ayant un fondement théorique et une description formelle,
- ✓ avoir une implémentation efficace, en vitesse et en bande passante.

Figure 4.1 Modèle des Langages de Communication entre Agents

NB : Une ontologie est une spécification ou une vue simplifié et abstraite du domaine qui sera représenté. En d'autres termes, une ontologie définit le vocabulaire dans un domaine donné pour que les agents puissent se comprendre.

3.5.2.1 Knowledge Query and Manipulation Language- KQML:

A l'origine, KQML a été développé pour échanger des informations et des connaissances entre des systèmes à base de connaissances. Il a été ensuite repris dans [FFMM94] pour décrire les messages échangés entre les agents.

KQML est un langage de communication et un protocole de haut niveau pour l'échange de l'information, orienté messages et indépendant de la syntaxe du contenu et de l'ontologie applicable.

En plus, KQML est indépendant du mécanisme de transport (TCP/IP, SMTP, IIOP ou autre), indépendant du langage du contenu échangé (KIF, SQL, Prolog ou autre) et indépendant de l'ontologie utilisée.

3.5.2.2 FIPA-ACL [FIP02a]

FIPA Agent Communication Language (FIPA-ACL) est proposé dans le cadre d'un travail de standardisation mené au sein l'organisation FIPA (Foundation of Intelligent Physical Agents). FIPA-ACL est une extension du langage KQML.

Il est fondé sur vingt et un actes communicatifs, exprimés par des performatifs, qui peuvent être groupés selon leurs fonctionnalités de la façon suivante:

- passage d'information : Inform, Inform-if, Inform-ref, Confirm, Disconfirm,
- Réquisition d'information : Query-if, Query-ref, Subscribe,

- Négociation : Accept-proposal, Cfp, Propose, Reject-proposal,
- Distribution de tâches (ou exécution d'une action) : Request, Request-when, Request-whenever, Agree, Cancel, Refuse,
- Manipulation des erreurs : Failure, Not-understood.

Un message FIPA-ACL peut contenir une partie ou la totalité des éléments décrits dans le tableau ci dessous Les éléments nécessaires pour la transmission d'un message changent selon la situation. Si un agent ne reconnaît pas ou ne peut pas traiter un ou plusieurs éléments, alors il peut répondre avec le message Not-understood.

Voici un exemple de message FIPA-ACL qui est envoyé par l'agent A à l'agent B:

(inform: sender A
: receiver B
:reply-with devis12
:language Prolog
:ontology Ordinateur
:content prix(HP,1500 EUR))
:conversation-id conv01
:reply-by 10 min)

Elément	Signification
Performatif	Le type de l'acte commutatif
Sender	L'émetteur de Message
Reciever	Le destinataire du Message
Reply-To	Le participant de l'acte de communication
Content	Le contenu du message (l'information transportée par le performatif)
Language	Le langage dans lequel le contenu est représenté.
Encoding	Le mode d'encodage du contenu de message
Ontology	Le nom de l'ontologie utilisé pour donner un sens aux termes utilisés.
Protocole	Le nom du protocole d'interaction.
Conversation-ID	L'identifiant de la conversation.
Reply-with	Un identifiant du message, en vue d'une référence ultérieure
In-Reply-To	Il référence le message auquel l'agent est entrain de répondre.
Reply-By	Un délai pour répondre au message.

Tableau-2.1 Les différents messages de FIPA ACL

L'agent *A* informe son interlocuteur que le prix d'un ordinateur *HP* est de 1500 euro. Le contenu du message est exprimé avec le langage *Prolog*. L'ontologie utilisée est celle des ordinateurs. Ce message fait partie d'une conversation ayant comme identifiant *conv01*. L'agent *B* est contraint par une durée de 10 minutes pour donner suite à ce message.

Les langages de communication, sont pleinement utilisés pour la spécification des protocoles d'interaction. Les protocoles d'interaction sont utilisés par les agents pour régir leurs interactions. Nous donnons dans la section suivante quelques définitions des protocoles d'interaction.

3.5.3 Protocole d'interaction :

Les protocoles d'interaction sont des descriptions de patterns standards d'interaction entre deux ou plusieurs agents. « They constrain the possible sequences of messages that can be sent amongst a set of agents to form a conversation of a particular type» [GHB00]

La majorité des travaux existants, notamment les travaux de FIPA, décrit en effet les séquences de messages échangés et leurs contenus (performative, émetteur, receveur…).

A partir des définitions de Greaves et al.[GHB00], Gaia [ZJW03], Huget et de Cossentino [Cos03a] il sort quatre notions fondamentales caractérisent un protocole d'interaction :

- un protocole d'interaction est un pattern d'interaction. Ce qui explique le besoin de le représenter d'une manière générique, indépendamment du contexte d'application.
- chaque protocole d'interaction a un but.
- un protocole d'interaction fait intervenir deux ou plusieurs agents. Chacun de ces agents joue un *rôle* qui permet de l'identifier au cours de l'interaction.
- le protocole d'interaction définit les règles qui permettent de régir une interaction. Ces règles définissent l'ordonnancement des messages, ainsi que les actions auxquelles le protocole fait appel.

3.5.4 Exemples de protocoles d'interaction

Nous décrivons dans cette section quelques protocoles d'interaction

3.5.4.1 Le protocole Contract Net (réseau contractuel)

Le Contract Net proposé par Smith en 1981 [Smi81], est le protocole d'interaction le plus utilisé dans les SMA. Il repose sur un mécanisme d'allocation de tâches régi par le protocole d'appel d'offres qui est utilisé dans les organisations humaines. Le modèle est basé sur une communication par envoi de messages.

Dans ce protocole, les agents peuvent prendre deux rôles : manager (participant) ou contractant (initiateur).

Le manager est responsable de la surveillance et l'exécution d'une tâche et du traitement des résultats de cette exécution. Un contractant fait une proposition au manager pour réaliser la tâche. En cas d'acceptation, il est responsable de l'exécution effective de cette tâche.

Le protocole est composé de trois phases :

- Annonce d'une tâche : le contractant fait une annonce de tâche (ou contrat) aux agents du système.
- Offre d'un service : les agents évaluent leur intérêt en fonction de leurs ressources. Si la tâche est intéressante, ils soumettent une offre.
- Attribution d'une tâche : le contractant choisit un ou plusieurs agents candidats pour exécuter la tâche et informe les agents de ce choix.

3.5.4.2 Les protocoles d'enchères

Les **enchères** (en anglais "auctions") sont des mécanismes d'interaction assez simples; pourtant il y a beaucoup de problèmes à étudier, concernant principalement **le choix du protocole et la stratégie à utiliser**. Une enchère comprend, d'habitude, un initiateur (en anglais "actioneer") qui annonce un objet à vendre, et plusieurs participants (en anglais "bidders") qui sont intéressés à l'acheter. L'initiateur veut vendre l'objet au prix le plus élevé possible et les participants veulent l'acheter au plus petit prix possible. Les participants font des offres, appelées aussi des soumissions (en anglais "bids"). Les offres des participants peuvent se faire une seule fois ou en plusieurs tours, en fonction du protocole d'enchère. Parfois, l'initiateur impose une offre minimale acceptable, appelée **prix de réservation**. A la fin, l'initiateur choisit le gagnant, les règles pour choisir le gagnant étant, de même, spécifiques au protocole.[ch01]

Nous retrouvons dans la littérature plusieurs types d'enchères, mais on se limite dans cette section à la présentation des types d'enchères les plus utilisés dans les SMA. Les protocoles enchères que nous décrivons dans cette section font appel à un commissaire-priseur et plusieurs participants.

Dans **l'enchère anglaise** [FIP02], le commissaire-priseur initie l'enchère en annonçant un premier prix initialement inférieur à la valeur estimée du produit. Après chaque annonce, le commissaire-priseur attend pendant un certain temps pour voir s'il existe des participants acceptant cette offre. Dès qu'un participant se manifeste, le commissaire-priseur annonce une nouvelle proposition de prix égale au dernier prix incrémenté d'une valeur appelée *pas*

d'incrémentation. L'enchère se termine quand aucun des participants ne se manifeste. A ce moment là, le commissaire-priseur compare le prix de la dernière offre acceptée avec la valeur estimée du produit. Si le prix de l'offre est supérieur à la valeur estimée alors l'enchère est accordée au participant qui s'est prononcé pour cette offre. Dans le cas contraire, les participants sont informés de l'annulation de la vente. Le manager peut réexaminer au cours de l'enchère la valeur estimée du produit, pour éviter toute annulation de la vente.

Dans l'enchère hollandaise [FIP02b] (ou allemande), le commissaire-priseur annonce au départ une valeur très élevée par rapport à la valeur estimée du produit. Puis il commence par baisser le prix jusqu'à ce qu'un participant signale son acceptation du prix proposé. Le vendeur possède une estimation de la valeur du produit (*prix de réserve*) au-dessous de laquelle il ne vend pas le produit. Si l'enchère atteint le prix de réserve sans qu'il y ait d'acheteurs, l'enchère se termine.

Dans **l'enchère Vikrey**, les propositions sont privées, aucun participant ne connaît le contenu des autres enchères. Les enchères Vickrey permettent de vendre un article unique, comme les enchères anglaises. La différence est que le soumissionnaire le plus élevé obtient l'article au prix offert par le deuxième plus haut soumissionnaire.

Dans **l'enchère au premier prix**, des enchères sont soumises au vendeur sous un certain format, sans qu'aucun des participants ne sache le contenu des autres enchères. Le gagnant paye le prix qu'il avait soumis.

Dans **la double enchère**, les participants sont des acheteurs et des vendeurs qui négocient sur un même produit. L'enchère peut démarrer par l'envoie d'une offre ou d'une proposition. Les vendeurs envoient des offres qui démarrent avec le plus haut prix puis elles décroissent, et des propositions sont faites par les acheteurs dont les prix évoluent dans le sens inverse des offres. Les offres et les propositions sont publiques, chaque participant est au courant de toutes les offres et les propositions soumises (l'émetteur et le prix proposé). La négociation peut s'achever si un acheteur accepte une offre proposée par un vendeur ou si un vendeur accepte une proposition d'un acheteur.

Il existe d'autres protocoles d'interaction que nous n'avons pas cités dans cette section, tels que les protocoles de vote [Tay95] et le protocole de Bargaining [SGBP03].

3.5.5 Classification des protocoles d'interaction

Dans la littérature, la majorité des classifications concerne les protocoles de négociation, vu que la négociation est la forme d'interaction qui nécessite le plus de communication entre les agents.

3.5.5.1 Les protocoles Compétitifs vs. Coopératifs

P. Maes [GM98] identifie deux classes de protocoles. La première classe regroupe les protocoles distributifs qui se basent sur le principe du partage des gains entre les participants (managers). Chacun des participants cherche à maximiser sa part de profit ; la réussite de l'acheteur signifie la défaite du vendeur, et vice versa. P. Maes les considère comme des protocoles *compétitifs*. Les protocoles de *vente aux enchères* sont considérés comme des protocoles compétitifs.

La deuxième classe regroupe les protocoles *intégratifs*, dont l'objectif de ses participants est de chercher des solutions qui maximisent les profits mutuels des différents participants. ils sont considérés comme des protocoles *coopératifs* en intégrant plusieurs paramètres de négociation.

3.5.5.2 Les protocoles Unidirectionnels vs. Bidirectionnels

Dans le protocole de négociation *unidirectionnel*, on se limite à l'acceptation ou le rejet d'une proposition. On trouve dans cette catégorie les protocoles de *vente aux enchères* et le protocole *Contract Net.*

Le protocole de négociation *bidirectionnel*, dont le scénario est plus complexe, se base sur un échange de messages (offre et contre-offre). Ce type de protocoles offre plus de flexibilité à la négociation soit en changeant la valeur d'un ou de plusieurs attributs négociables, via une contre-proposition, soit en changeant la structure de la proposition, en ajoutant par exemple un nouveaux attribut négociable. Le protocole *Bargaining* [SGBP03] est un exemple de protocole bidirectionnel, qui utilise les performatives *propose* et *counter-propose*.

3.5.Autres classifications

Différentes autres classifications des protocoles d'interaction ont été proposées. Wurman et al. [WWE98] proposent une classification basée sur les critères suivants :

1) Simple ou double : protocole est simple si un agent enchérisseur est soit acheteur soit vendeur dans la même enchère. Dans la double enchère on trouve des agents acheteurs et des agents vendeurs qui participent à la même enchère.

2) Privé ou public : dans le premier cas, les propositions soumises par les participants ne sont connues qu'après la clôture de l'enchère. Dans le deuxième cas, les propositions sont publiques, connues par tous les participants.

3) Conditionné ou inconditionné : un protocole est conditionné si les propositions sont régies par des règles, par exemple, le prix proposé doit être ascendant ou descendant.

Mazouzi [MFSH02] classifie les protocoles d'interaction en quatre principales classes : les protocoles de coordination, les protocoles de coopération, les protocoles de négociation et les protocoles de vente aux enchères.

3.5.6 Implémentation des protocoles d'interaction

Actuellement plusieurs plates-formes pour le développement des SMA sont disponibles, telles que Madkit [Gut01], Zeus [NNLC99], FIPA-OS [PBH00], JACK [Gro04] et JADE [BCTR04], etc. Cependant, ces plates-formes ne proposent pas une solution faciliter l'utilisation des protocoles d'interaction, à l'exception de la plate-forme JADE. A notre connaissance JADE est la seule plate-forme multi-agents qui propose une bibliothèque de protocoles d'interaction.

JADE [BCTR04] (Java Agent Development Framework), est un framework pour le développement logiciel qui vise de développer des systèmes multi-agents et des applications conformes aux standards de FIPA. Il inclue deux principaux produits : une plate-forme agent FIPA compliant et un package pour le développement des agents en Java. JADE est codé en Java et le développeur doit implémenter ces agents en Java pour pouvoir exploiter ce framework.ce der nier offre un ensemble de packages, parmi lesquels le package jade.proto qui est un ensemble de classes qui modélisent un certain nombre de protocoles d'interaction standardisés (*Fipa-Request, Fipa-Query, Fipa-Contract-Net, Fipa-subscribe* et d'autres protocoles définie par *FIPA*). Cette API aide le programmeur à utiliser les protocoles d'interaction. Les messages sont décrits avec le langage *FIPA-ACL*. Le package jade.lang.acl contient des classes qui implémentent les performatifs de FIPA-ACL.

La classe Agent offre un ensemble de méthodes qui assurent la communication inter-agents avec envoi de messages en mode asynchrone. Pour chaque interaction, JADE distingue un rôle initiateur et un rôle participant. Chaque rôle d'interaction est décrit par un automate à états finis. Les actions qui traitent les messages reçus font appel à des méthodes abstraites de la classe du rôle, que le développeur doit surcharger pour pouvoir instancier le rôle. JADE propose des méthodes génériques qui peuvent être utilisées, par défaut, pour traiter les actions du rôle.

Chapitre 3 :

Les Plateformes Multi-Agents

Introduction :

Depuis l'apparition de la notion d'agent mobile, de nombreuses plates-formes ont été développées afin de faciliter la programmation d'applications structurées en termes d'agents mobiles.

Une Normalisation était nécessaire pour rendre ces différents systèmes à base d'agents compatibles entre eux.

Dans ce chapitre on va représenter une étude détaillée des standards et de différentes plateformes car, selon nous, chacune permet de mettre en avant des caractéristiques que nous souhaiterions trouver dans un middleware pouvant s'adapter dans un environnement dynamique.

1. Les Normes Des Agents Mobiles:

La technologie des Agents mobiles fut retardée par le manque des normes (les standards) de ce qui concerne le code, les données, communication et l'interopérabilité.

Au temps présent il existe deux normes principales FIPA et OMG MASIF [IPNET 07]

1.1 OMG MASIF :

Autrefois appelée MAF (Soutenues par les départements de recherches et IBM qui étaient actifs dans le domaine des Agents Mobiles), la norme MASIF (Mobile Agent Interoperability Facility) a été développée par OMG (Object Management Group) en 1998.

Cette norme se préoccupe généralement de la migration des agents mobiles entre les systèmes des agents ayant le même profil basé sur la norme CORBA Interface Definition Language (DIL).

En réalité MASIF se constitue d'ensemble de définitions et interfaces pour le transfert des agents et leur gestion.

MASIF standardise la manière de gérer le système agent, code des agents, leur identification, la et l'adressage local et quelque d'autres notions et concepts de base.

1.1.1 Les standards :

Sont les suivants [IPNET 07]:

a. La gestion d'agent :

En tant qu'administrateur du système, elle peut gérer différents types de systèmes agents. Donc un agent peut être créé, lancé, suspendu ou terminé.

b. Le transfert d'agent :

L'application basée sur l'agent peut migrer d'une manière libre entre les différents systèmes d'agent. Les classes d'un agent peuvent être échangées entre les plateformes par les méthodes de transport d'agent.

c. Les noms des agents et des systèmes d'agents :

MATIS est le standard utilisé pour la sémantique et la syntaxe des noms des agents et des systèmes d'agents qui permet aux agents de s'identifier entre eux.

d. La syntaxe des types et des adresses des systèmes d'agents :

Le transfert d'agent est impossible si le type du système destination n'est pas compatible avec l'agent, afin d'éviter ce type de problèmes La syntaxe de l'emplacement est standardisé par MATIS.

MASIF ne pose aucun standard ou contrainte sur la communication des agents mobiles car ce type de problème est traité par CORBA et même pour les problèmes de sécurité elle compte sur CORBA et ses principes.

1.2 FIPA :

De son coté la communauté d'origine FIPA (Foundation for Intelligent Physical Agents) a basé sur le coté Intelligence Artificiel et plus précisément l'interopérabilité des agents et les problèmes concernant la communication des agents et le langage ACL (Agent communication language) [TOOL 05].

1.2.1 Brève histoire de FIPA : [JADE 07]

1996: FIPA a lancé un appel à la communauté afin de connaître les différents domaines d'application qui va plus tard former l'ensemble des spécifications de FIPA'97.

A partir de 12 réponses reçues, 4 ont été sélectionnées : assistance personnelles, diffusion audio-visuelles, gestion de réseau, assistance personnelle mobile.

1997 : FIPA a décidé d'utiliser ACROL de France Télécom comme un langage de spécification qui a devenu plu tard FIPA ACL ou ACL.

1998 : Les spécifications de base ont été améliorées pour s'adapter à la mobilité des agents et les nouvelles interactions (agent-humain) en plus de ça des travailles supplémentaires ont été effectués sur la sécurité des agents.

1999 : au début de cette année la spécification FIPA'98 a été lancée en retard, plusieurs changements et améliorations ont été effectués sur la FIPA'97.

Entre 20003 et 2004: FIPA se concentre maintenant sur les réseaux ad hoc (communication, sécurité, spécification et modèles....). Fin 2004 FIPA a été déconnecté dû au manque d'assistance et de soutien.

2005-2009 : FIPA a été réintégrée par IEEE (et devient le 11eme comité de standardisation de IEEE Computer society) comme un standard appelé FIPA-IEEE. La plupart des groupes de travaille se concentre sur l'interaction humain-agent, et les agents nomades dans un réseau P2P.

Jusqu'à présent FIPA a pu réaliser le suivant :

- ✓ Un ensemble de spécifications qui supportent la communication des agents.

- ✓ FIPA ACL comme un langage largement utilisé en communication des agents, et des protocoles d'échange de messages jusqu'au protocole des transactions complexes.
- ✓ Des outils open source ou commercial tel que FIPAOS et JADE qui est considéré comme le leader de FIPA-compilant (compilateur FIPA)
- ✓ Une extension de UML appelée AUML (Agent Unified Modeling Language)

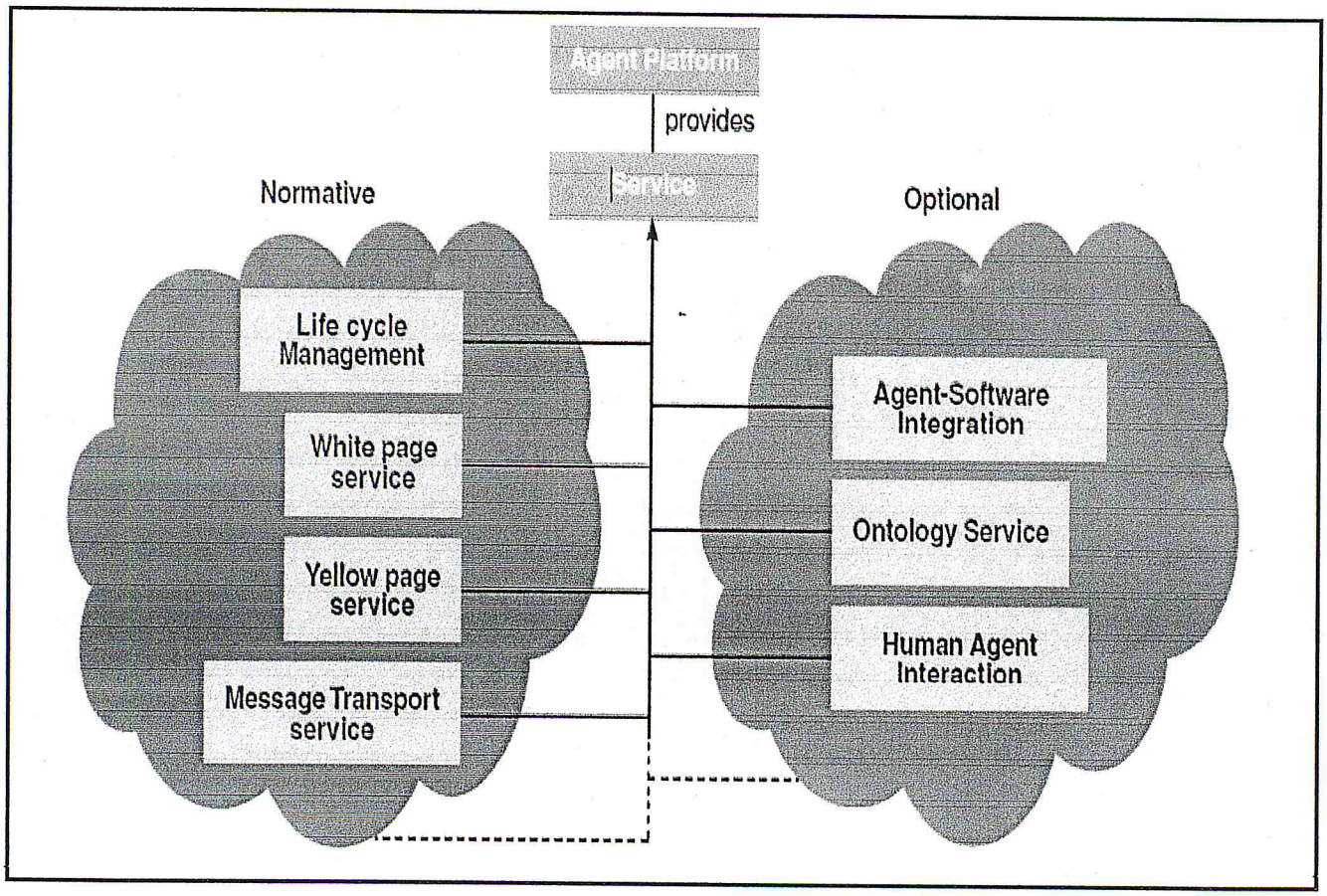


Fig.-3.1Les services fournis par FIPA

1.2.2 Les standards :

1.2.2.1 Les agents selon la FIPA

- ➢ Ils sont définis par un identifiant (Agent IDentifier AID) unique au sein de la plate-forme et d'un état d'exécution comprenant le code et les données qu'ils transportent et il a au moins un propriétaire.
- ➢ Les agents communiquent par un Agent Communication Language(ACL)
- ➢ Un Agent peut enregistrer avec des adresses de transport (Agence) où il peut connecter.

1.2.2.2 Les Directory Facilitator (DF) Selon la FIPA

- Des composantes optionnelles d'une plateforme.
- Les DF offrent des services des pages jaunes (Yellow Pages Services).
- Les Agents peuvent enregistrer leurs services sur les pages jaunes ou demander de la DF de trouver certains services offerts par des autres agents.
- Plusieurs DF peuvent exister dans la même plateforme.

1.2.2.3 Agent Management System (AMS) :

- Une composante principale d'une Plateforme Multi-Agents.
- L'AMS est responsable sur les accès et l'utilisation d'une Plateforme Multi-Agents.
- L'AMS garde une liste d'AIDS des agents enregistrés dans la Plateforme Multi-Agents.
- Les agents doivent enregistrer avec un AMS pour avoir un AID valide.

1.2.2.4 Message Transport Service (MTS) ou Agent Communication Language ACL

- La méthode de communication par défaut des agents qui résident distincts.

1.2.3 L'interopérabilité entre plateformes FIPA :

1.2.3.1 Communication entre Agents :

Réaliser par l'utilisation du MTS (ACL) et les protocoles standards de transport de message de la FIPA (cas Plateforme à Plateforme) et des technologies non-standard (Interne à la plateforme).

1.2.3.2 Environnement d'exécution d'un agent sur chaque plateforme:

On peut utiliser des langages, API, propriétés de transport et des architectures d'agents différents mais les mêmes services de base et de transports le même ACL(MTS).

2. Les Plateformes Agent Mobiles :

2.1 La Plateforme JADE :

JADE (Java Agent DEvelopment Framework) est un intergiciel construit sur Java par CSELT de l'université de Parma afin de permettre une programmation multi-agents simplifiée en prenant en compte la norme FIPA [Web 06 jade.it]

Elle inclut deux principaux produits : une plate-forme agent FIPA compilant et un package pour le développement des agents en Java.

2.1.1 Brève histoire de JADE : [JADE 07]

JADE fut lancée par Telecom ITALIA (CLEST) en 1998 qui a intégré les premières spécifications de la FIPA dans cette Plateforme.

JADE est open source depuis Février 2000 sous licence LGPL (Library Gnu Public Licence) cette licence ne pose pas des restrictions sur l'application qui utilise (JADE)

En Mai 2003 Telecom ITALIA LAB et Motorola Inc. Ont créé le JADE governing board l'organisation des contributeurs pour le développement et la promotion du JADE.

JADE était exclusivement utilisée par la communauté de FIPA quand Telecom ITALIA l'a publié les premiers jours, mais elle est rapidement devenue un outil de support pour plusieurs projets de transfert technologique ou en exploitation commerciale.

2.1.2 Caractéristiques :

- Support à l'envoi de messages, transparent et multi protocoles (grâce au package de Java Jade.proto).
- La communication des agents est basée sur l'envoi de messages FIPA 2000 Transfert Protocol est le langage par default de communication.
- Protocole http et encodage des ACL en XML
- JADE est basée sur le protocole IIOP pour la diffusion inter-plateformes conforme aux normes de spécifications de (FIPA)
- Framework Orienté Objet implémentant en Java les spécifications FIPA.
- JADE est codée en Java et le développeur doit implémenter ces agents en Java pour pouvoir exploiter ce Framework.
- JADE encapsule les spécifications de la FIPA :
- La plateforme fournit : AMS, DF et MTS.
- Transport et traitement des messages.
- Système de gestion des événements dans le noyau de la plateforme
- Permet l'observation de la plateforme, de messages, du transport des messages des agents.

2.1.3 L'architecture de JADE :

Une Plateforme jade est composée de « conteneurs » d'agents qui peuvent être distribués dans le réseau.

Les Conteneurs (Containers) :

Les conteneurs sont des agences qui offrent :

1) Un environnement complet d'exécution pour un agent.
2) Exécution parallèle de plusieurs agents.
3) Contrôle de cycle de vie des agents
4) Assure la communication entre les agents.

Un seul conteneur héberge l'AMS le DF c'est le conteneur principal son nom par default est « *main container* », ce dernier est le premier conteneur qui sera créer au lancement de la plateforme et tous les autres conteneurs doivent le joindre pour par enregistrement avec lui.

Le conteneur principal peut être dupliqué via des services de duplication.

La figure suivante montre les éléments principaux d'une plateforme JADE :

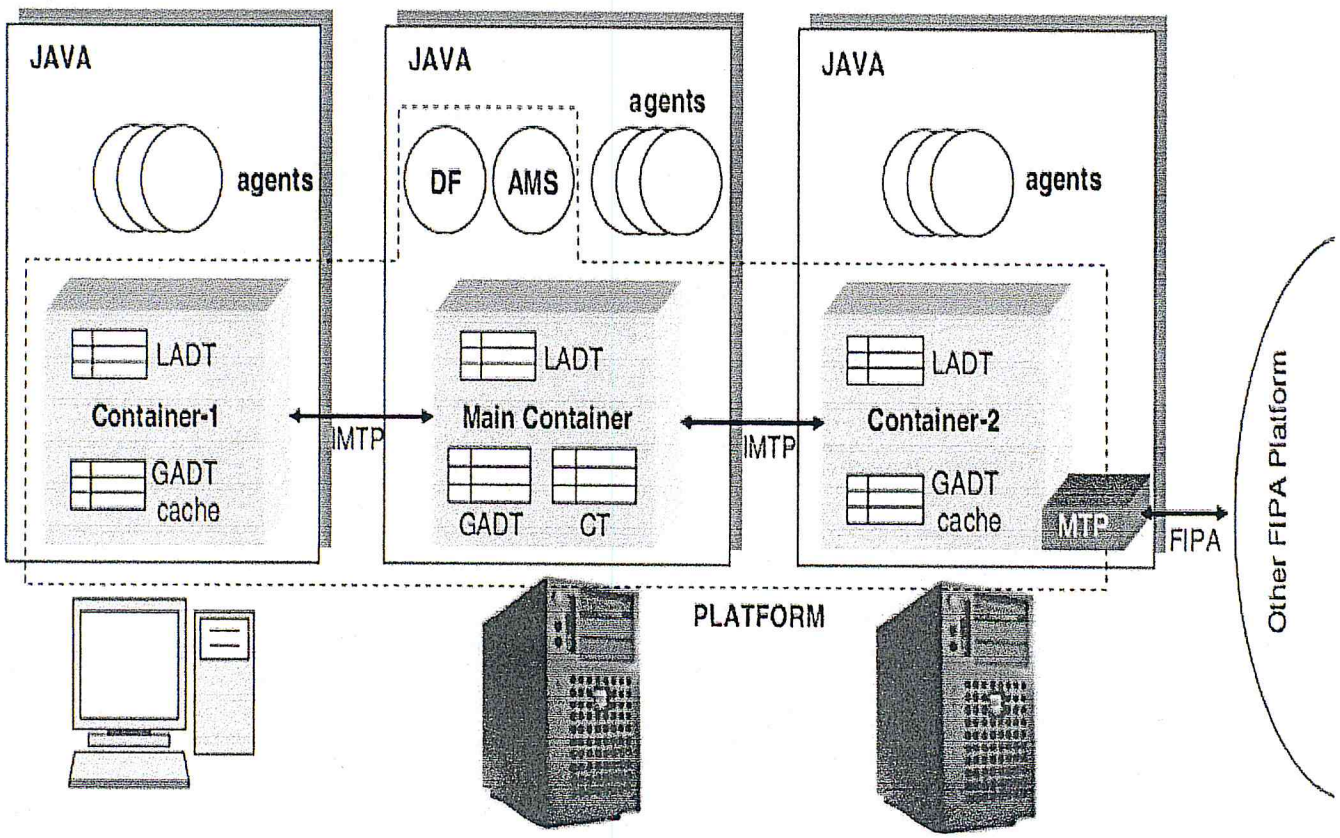


Fig3.2 Architecture d'une plateforme JADE [JADE 07]

2.1.4 Interaction dans JADE:

La classe Agent offre un ensemble de méthodes qui assurent la communication inter-agents avec envoie de messages en mode asynchrone. Pour chaque interaction, JADE distingue un rôle initiateur et un rôle participant. Chaque rôle d'interaction est décrit par un automate à états finis. Les actions qui traitent les messages reçus font appel à des méthodes abstraites de la classe du rôle, que le développeur doit surcharger pour pouvoir instancier le rôle. JADE propose des méthodes génériques qui peuvent être utilisées, par défaut, pour traiter les actions du rôle.

Cette modélisation ne facilite pas la tâche du développeur puisqu'il doit connaître le fonctionnement du protocole et réécrire toutes les méthodes du rôle. L'intervention nécessaire du Concepteur lors de l'instanciation des protocoles d'interaction ne permet pas aux agents de pouvoir changer dynamiquement de rôles.

JADE présente des solutions intéressantes pour résoudre certains problèmes liés à la représentation et l'implémentation des protocoles d'interaction.

2.2 Madkit : [Web07, GDM 09]

Madkit ou Multi-Agent Development Kit [Web07] est une plateforme multi-agent modulaire développé par LIRMM (Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier de l'Université Montpellier II) pour l'utilisation académique. Madkit est écrit en Java et basé sur le modèle organisationnel Alaadin ou Le modèle **AGR** (Agent/Groupe/Rôle).

Il se base sur trois principes architecturaux : un micronoyau, une Agentification des services et une interface componentielle.

2.2.1 Le micronoyau Agent [DevG 09]

Le micronoyau agent de MADKIT est un environnement d'exécution d'agents compact (moins de 40 Ko de bytecode). Le terme "micronoyau" est intentionnellement utilisé en référence à la philosophie des systèmes d'exploitation à micronoyau [Rashid et al. 89]. R. Rashid les définit comme un "*jeu minimal d'outils permettant un déploiement aisé de services de système d'exploitation*".

Nous remplacerons simplement dans cette définition le dernier terme par agents.

Le noyau ne gère que les tâches suivantes:

- **Contrôle des groupes et rôles locaux**: Le micronoyau a la responsabilité de maintenir une information correcte sur les membres des groupes et sur les différents rôles tenus. Il transmet

également les demandes d'admission ou de renseignements aux agents gestionnaires des groupes idoines.

- **Gestion du cycle de vie**: Le micronoyau lance les agents, peut les suspendre ou les arrêter, et leur assigne des identifiants uniques.
- **Passage de message local**: Les messages envoyés d'un agent à l'autre sont remis par le noyau si le receveur et l'émetteur sont locaux. Si ce n'est pas le cas, le message sera éventuellement remis par le biais d'un agent système spécialisé.

2.2.2 Agentification des services

2.2.2.1 Agents, groupes et rôles dans la plateforme MADKIT

Les agents sont définis en héritant d'une classe abstraite qui fournit des mécanismes d'identifications, l'envoi et réception de messages, et une API liée au système de groupe et rôle. Ces méthodes permettent la création de groupe, l'admission, et l'émission de requêtes pour identifier les rôles présents dans un groupe, les agents en charge d'un rôle, ainsi que la demande, la délégation ou le retrait d'un rôle.

Quelques groupes particuliers sont définis: un groupe local rassemble tous les agents s'exécutant sur le micronoyau local. L'admission à ce groupe est automatique, et l'identification d'agents particuliers sur la plate-forme passe par l'implémentation standard de groupe et rôle. Le second groupe particulier est le groupe système, qui rassemble les agents ayant la possibilité d'agir sur le noyau, et donc un pouvoir sur les autres agents de la plate-forme. L'accès à ce groupe est donc sévèrement réglementé et restreint à certains agents lancés au démarrage ou validés par une signature.

L'interaction entre agents systèmes et le noyau passe elle-même une interaction agent standard. Le tout premier agent créé à l'amorçage du noyau est en fait un agent

" *Wrapper* " qui sera le seul à avoir accès à toutes les possibilités et références internes du noyau. Il fonde le groupe local et le groupe système et y assure le rôle de gestionnaire de groupe. Ensuite, les membres du groupe système pourront demander certaines actions privilégiées en dialoguant avec lui, d'où une sécurité accrue.

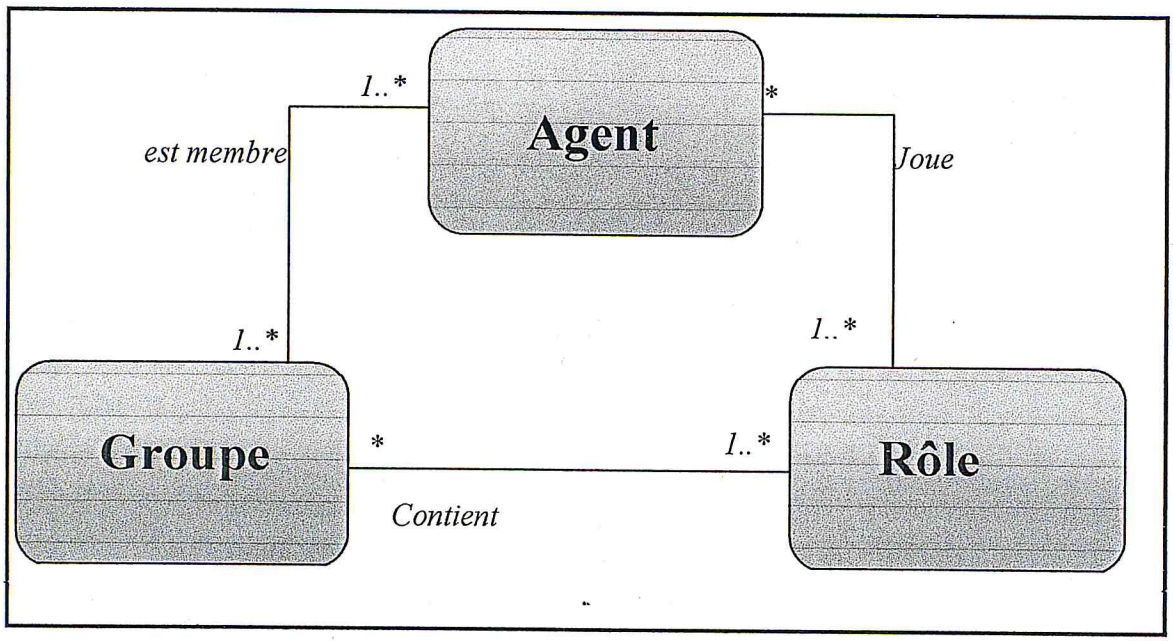


Fig-3.3 Le modèle Agent Groupe Rôle [JADE 07]

2.2.2.2 Services agents

Contrairement à d'autres plateformes, MADKIT utilise des agents standard pour gérer l'envoi de message en distribué, la migration, la sécurité et d'autres aspects similaires. Cela rend la plate-forme particulièrement adaptable, vu que les services agents peuvent être remplacés à loisir. Par exemple, il est possible d'implémenter un mécanisme de gestion de la distribution complètement différent de celui fourni sans voir à changer quoi que ce soit dans les autres agents de la plate-forme. Ces services peuvent également être adaptés au moment de l'exécution en déléguant des rôles d'agent à agent.

Ce principe de délégation de rôle a l'autre effet intéressant de permettre une adaptation à la charge: un agent peut tenir plusieurs rôles au début de la vie d'un groupe, et au fur et à mesure que le groupe grandit, lancer de nouveaux agents et leur confier certains de ses rôles.

2.3 La Plateformes Magique [Web 08] [TuT 09]

MAGIQUE est une API (Application Programming Interface) Java qui permet de développer facilement des agents.

Selon les cas, les agents peuvent tourner sur une même machine ou sur des machines physiquement distribuées sur un réseau. MAGIQUE est un environnement de développement pour SMA mais n'est pas un SMA lui-même. MAGIQUE permet de masquer les aspects techniques de communication entre agents, le Multithreading et la gestion des flux. MAGIQUE banalise la communication entre agents en permettant un appel "a la cantonade". Avec MAGIQUE, inutile de savoir qui possède une compétence, il suffit de savoir qu'elle existe. Cela permet une grande réutilisabilité des agents indépendamment du contexte.

MAGIQUE permet apriori de ne faire communiquer que des agents issus de MAGIQUE.

MAGIQUE signifie "Multi-Agent hiérarchique » [Web 08] .MAGIQUE est à la base un modèle d'organisation d'agents qui propose une organisation hiérarchique.

Dans MAGIQUE, un agent est une entité possédant un certain nombre de compétences. Ces compétences permettent à un agent de tenir un rôle dans une application multi-agents. Les compétences d'un agent peuvent évoluer dynamiquement (par échanges entre agents) au cours de l'existence de celui-ci, ce qui implique que les rôles qu'il peut jouer (et donc son statut) peuvent également évoluer au sein du SMA.

Dans MAGIQUE les agents sont dénis au sein d'une plate-forme d'accueil, la philosophie de la chose est d'avoir une plate-forme par machine (et donc tous les agents dans la même JVM), même si cela n'est pas une contrainte imposée. Schématiquement,

Une application MAGIQUE peut fonctionner selon deux modes d'exécution différents:

1. **Centralisée**

   Dans ce cas, tous les agents lancés tournent dans la même plate-forme.

2. **Distribuée**

   Dans ce cas, des agents tournent sur des machines différentes. Il y a alors une plateforme sur chaque machine et les agents distants communiquent entre eux via leurs plateformes respectives (grâce au mécanisme RMI).

Pour toute personne connaissant le langage JAVA, l'apprentissage de MAGIQUE est très simple et l'utilisation de l'API permet de mettre en œuvre très rapidement et facilement des applications SMA distribuées.

L'environnement graphique du MAGIQUE permet la création des agents et de la structure organisationnelle du SMA qui les regroupent ainsi que le déploiement et l'exécution de ce SMA sur un réseau de machines hétérogènes.

2.4 La Plateforme Jack

JACK [Gro04] est un environnement de développement orienté agent conçu comme une extension de langage de programmation Java. Les agents dans JACK sont considérés comme des composants logiciels autonomes qui ont des buts explicites à accomplir. Pour décrire comment il faut procéder pour atteindre ces buts, les agents sont programmés avec un ensemble de plans (intensions).

Chaque plan décrit comment atteindre le but sous différentes circonstances. Mis en action, l'agent poursuit ces buts (désires), en adoptant des plans appropriés (intentions) en se basant sur les données (croyances) de l'état de l'environnement. Pour supporter la programmation des agents BDI (Belief Desire Intention), JACK propose un langage se basant sur cinq principaux concepts: l'agent, les compétences, les relations de base de données, les événements et les plans.
Dans JACK, le comportement d'un agent est décrit par des plans dont l'exécution est conditionnée par l'occurrence d'événements. Chaque plan peut traiter un certain type d'événements décrit dans la clause Handle du plan. Actuellement, il n'existe pas d'API dans JACK pour le développement et l'utilisation des protocoles d'interaction.
Il n'existe pas une implémentation proprement dite des protocoles d'interaction dans JACK. Les protocoles d'interaction sont implémentés sous forme de plans. L'activité du protocole est décrite d'une manière procédurale dans la méthode body() du plan. Cette méthode décrit les actions exécutées par l'agent suite à la réception des messages.
Les messages échangés entre les agents, et plus précisément le nom des performatifs sont implémentés comme une extension de MessageEvent. Dans JACK, les langages de communication tels que FIPA-ACL ou KQML ne sont pas utilisé. Ceci pose certaines difficultés pour exprimer la sémantique des messages entre les agents et les informations qu'ils comportent.
Pour utiliser le langage FIPA-ACL, Pasquier propose d'ajouter une classe Java qui implémente la couche syntaxique de FIPA-ACL. D'utilisation simple, cette classe (nommée FIPA Message) permet de créer des messages FIPA, d'indiquer de quel acte de langage il s'agit (Inform, Request, Cancel, Agree...) et de renseigner les différents champs du message à l'aide des méthodes prévues à cet effet (setEmeteur, setRecepteur, setLangage, setOntologie, setContenu...).

Les actions qui traitent les messages envoyés et reçus sont implémentées comme extension de plan et ils définissent le nom du ou des performatifs dans le *Handle*. Les données utilisées au cours de l'interaction, tel que le "*Time-Out*", sont considérées comme des croyances. Ils font partie des données de la base de données de l'agent.

JACK n'a pas ajouté de nouveaux concepts relatifs à l'interaction, tels que le rôle et le protocole d'interaction. De ce fait, l'implémentation de l'aspect interactif des agents peut être considérée comme had-doc. Cela engendre des problèmes de maintenance des agents, en voulant par exemple changer de protocole d'interaction, ce qui complique la tâche du développeur.

Une autre limite de cette approche, est la difficulté de réutiliser le code d'implémentation d'un protocole d'interaction dans différentes applications.

3 Les Framework Agents mobiles existantes :

3.1 Telescript :

A été développé par General Magic Inc. Avec son propre langage Orienté Objet pour agents mobiles il est considéré comme premier SMA destiné au marché (Gray & Rus, 2000). Son langage Telescript contient des classes qui permettent un héritage multiple et il est un des SMA les plus sécurisé et les plus efficaces (Tardo & Valent 1996). Mais à cause du succès du Java, Telescript est devenu dépassé, mais son architecture et son concept donna des idées et des fondations a plusieurs autre systèmes.

3.1.1 Architecture de Telescript :

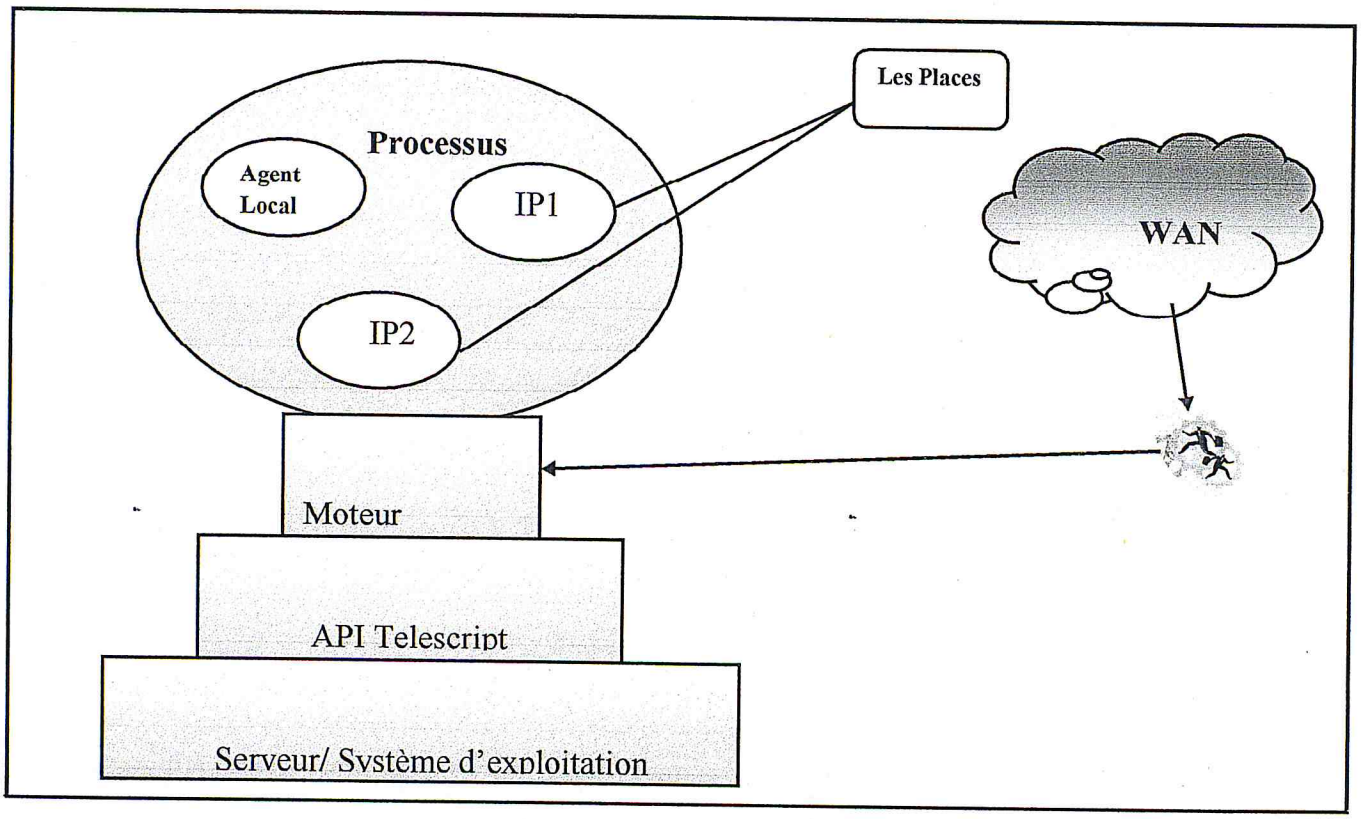


Fig-3.4 Architecture Partielle de Telescript

- Dans une architecture Telescript les agents s'exécutent dans des « Places ». Les places sont divisées selon les services qu'ils fournissent ex : Une place peut être un accès à une base de données. Un agent effectue son travaille on migrant d'une place vers une autre (Gray & Kotz 2000).

Un agent accède a une place selon un mécanisme de sécurité, un agent doit avoir un certificat qui lui permet d'identifier son rôle dans la plateforme et combien de temps il va occuper les ressources, si un agent ne les respecte pas il va être détruit.(Baumer, Breugst 2000).
Un agent mobile accède à un serveur à partir d'un WAN puis le Moteur du Telescript (Comme montré dans la figure précédente). Ce dernier lance l'agent, il est embarqué dans le serveur et son système opérationnel par le API du Telescript. Chaque serveur contient une machine virtuelle

(comme JAVA) ou un agent peut s'exécuter (L'agent est compilé en bytecode afin d'être exécuté).Le serveur contient un « Processus ».

Un processus est un agent qui hérite du class Processus. Chaque place a sa propre adresse IP, donc un agent peut l'identifier dans un WAN par cette adresse puis il l'utilise pour joindre une place (migre).

Un Agent Local est un agent statique qui peut fournir un service ex : l'agent statique peut être un agent seller (fournisseur) si l'agent visiteur est un buyer (client).

3.2 AgentTcl et D'Agents :

Ce système fut développé par l'université de Dartmouth (Gray 1996) afin de replacer des autres SMA, spécialement leur sécurité et le langage supporté. Il utilise un seul langage Tcl et l'extension « Safe Tcl ». Il supporte des agents des autres langages.

D'Agents a remplacé Agent Tcl comme une version avancée, car elle a inclus plusieurs langages comme le langage Java, Tcl et Scheme. Il était le premier produit commercialisé (pour l'utilisation dans les applications).

3.2.1 L'architecture Générale :

L'architecture D'Agents est une extension du l'architecture Agent TCL.

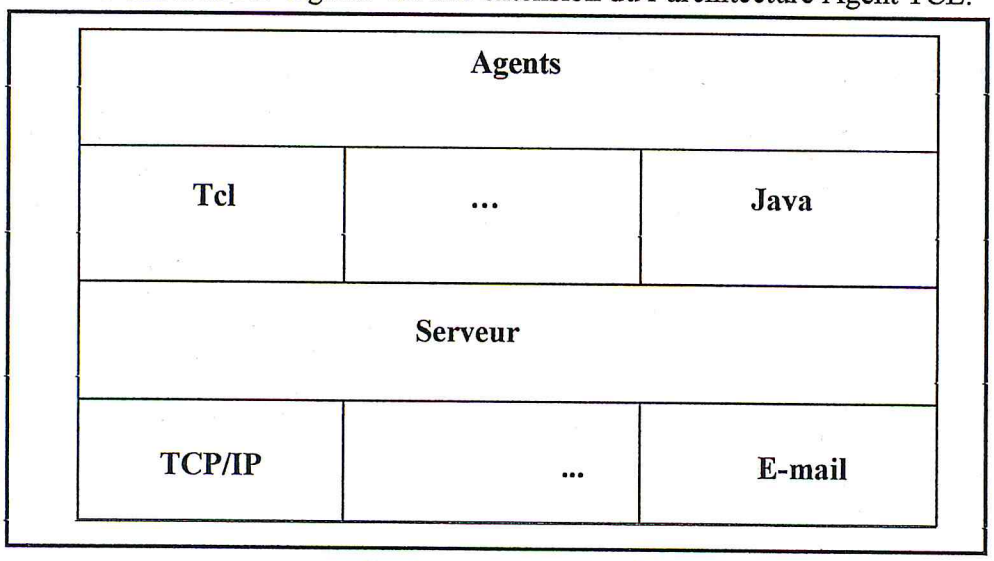


Fig-3.5. Architecture D'Agents

La figure précédente représente l'architecture D'Agents. Elle montre qu'un agent peut être implémenté en plusieurs langages (Java, Tcl, Scheme). Chaque langage doit fournir quatre composantes :

1. Composantes de capture d'état d'agent.
2. Composantes de sécurité d'agent.
3. Composantes d'interprétation.
4. Un serveur API.

- Quand un agent arrive sur un site (un serveur D'Agent), il va être envoyé vers son interpréteur correspondant ex : un agent Java va être envoyé à un interpréteur Java. (Gray, Kotz & Rus 2001). L'interpréteur forme une interface avec le serveur. En ce qui concerne la sécurité, elle est maintenue par les interpréteurs car tous les accès sont basés sur des permissions/restrictions fournis par le manager des ressources.

3.3 Aglets:

Aglets WorkBench (AWB) est un autre système d'agent mobile basé sur Java. Fut Développez par IBM il contient un environnement de développement des agents et leur exécution.

Aglets Workbench contient au tout début :

- Un environnement visuel de construction d'applications réseaux (**Tazza**, maintenant disparu) utilisant des agents mobiles écrits en Java.
- **Tahiti**, le serveur d'Aglets inclut dans la distribution, toujours distribué dans la version Open Source.
- Les API des Aglets.

AWB deviendra plus tard ASDK (Aglets Software Development Kit).

3.3.1 Architecture Aglets

Aglets permet la création des objets Java appelés Aglets (Agile Applets) (Lange & Oshima 1997). Ces Aglets contiennent leur code et leurs données. L'aglet s'exécute dans son environnement qui s'appelé « Contexte ».

Ce « Contexte » est un objet stationnaire qui s'exécute sur le serveur. Le **Contexte** est une de com.ibm.aglet.AgletContext.

Chaque Aglet possède un identifiant unique sur l'ensemble du réseau, défini par com.ibm.aglet.AgletID.

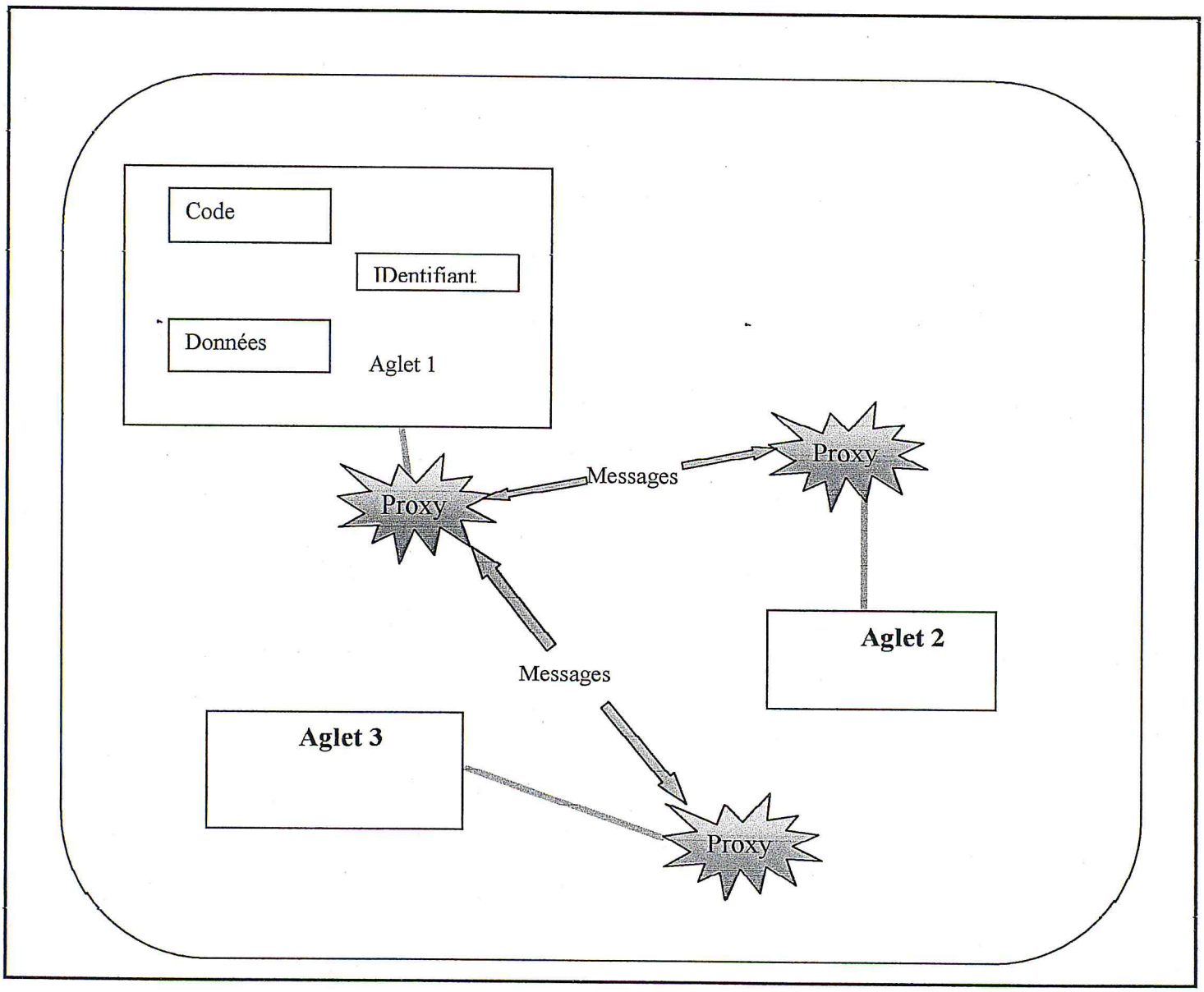


Fig-3.6 L'architecture d'Aglets dans un Serveur

Les tableaux suivants représentent un résumé de comparaison entre les différentes plateformes connues.

Tableau.2. Comparaison entre les différentes Framework- Protection de La Plateforme.

	Certification des Agents	Historique de Chemin	Tolérances Aux Fautes	Protection des Ressources	Interopérabilité Sécurisé	Techniques Money-Based
Telescript	Oui	Non	Partiel	Oui	Partiel	Non
D'agent	Oui	Oui	Non	Oui	Oui	TBI
Ara	Oui	Non	Oui	Partiel	Oui	Oui
Tacoma	Oui	Non	Non	Oui	Non	Oui
Ajanta	Oui	Non	Non	Oui	Non	Non
Concordia	Oui	Non	Non	Partiel	Non	Non
Grasshopper	Oui	Non	Non	Oui	Non	Non
Aglets	Oui	Oui	Non	oui	Non	Non

	Encapsulation des résultats	Génération des Clés	l'*obfuscation* du *code*	Fonctions Cryptographiques	Protection D'itinéraire	Techniques Time-Based
Telescript	Partiel	Non	Non	Non	Non	Non
D'agent	Non	Non	Non	TBI	Partiel	TBI
Ara	Partiel	Non	Non	Oui	Oui	Non
Tacoma	Partiel	Non	Non	Non	Partiel	Non
Ajanta	Oui	Non	Non	Oui	Oui	Non
Concordia	Oui	Non	Non	Oui	Oui	Oui

Grasshopper	Non	Non	Non	Non	Non	Non
Aglets	Non	Non	Non	Non	Non	Non

Tableau.3. Comparaison entre les différentes Framework- Protection d'agent Mobile

Conclusion :

Dans notre choix de la plate forme d'agents nous avons choisi JADE (Java Agent Development Framework). Ce choix est justifié par le fait que JADE est une plate-forme multi-agents développée en Java, dotée d'une interface graphique et qui peut être répartie sur plusieurs serveurs et le fait d'être open source va faciliter l'implémentation et à notre connaissance seule la plate-forme JADE propose une bibliothèque de protocoles d'interaction, que le développeur peut réutiliser pour le développement de ses agents. Parler aussi des plates-formes agents mobiles !!!!!

Partie 2 : Réalisation

Chapitre 4 :
Conception

Introduction

Le paradigme des Agents Mobile est le paradigme le plus novateur parmi ceux qu'a connu l'évolution du code mobile. L'idée de travailler avec des entités Autonomes, proactives et intelligentes a toujours été fascinante pour les chercheurs de plusieurs domaines. La recherche sur les agents explore et réalise éventuellement l'idée des composants logiciels se comportant comme des individus de la société humaine, avec toutes les capacités rationnelles et sociales. Cette technologie des agents vise à offrir un nouvel outil de conception des applications avec un niveau d'abstraction qui s'approche des comportements humains plutôt que de la machine physique.

Malgré tout ça la question «pour quoi utiliser les agents mobiles ?» a du mal à trouver une réponse.

Une raison principale pour cela est que les travaux actuels se focalisent sur l'aspect technique [ROU-02] :

- Les agents mobiles supportent les opérations en mode déconnecté et par conséquent les opérations asynchrones ce qui n'est pas le cas des modèles les plus répandus actuellement à savoir Client-serveur.
- Les agents mobiles permettent la personnalisation (ou la spécialisation) des services présents sur un réseau, les techniques client-serveur proposent souvent un service fixé à priori à travers une interface statique. le service étendu apporte plus de flexibilité.
- Les agents mobiles réduisent la charge du réseau lorsque la taille des données offertes par un service s'accroît considérablement : les interactions locales et la migration des traitements vers les données réduisent cette croissance.
- Les agents mobiles apportent une capacité de croissance aux systèmes car ils facilitent la répartition de charge.

1. Limites de la technologie des agents mobiles :

Comme toute technologie, la technologie des agents mobiles est loin d'être parfaite et souffre encore de quelques problèmes dont on peut citer :

- Les systèmes d'agents mobiles proposés restent des travaux expérimentaux et sont proposées à l'évaluation libre sur le web vu que chaque constructeur propose ses propres approches.
- La normalisation nécessaire et ses tentatives qui n'aboutissent pas encore.
- L'interopérabilité non assurée entre les différents systèmes d'agents mobiles qui existent, même si de grands efforts sont faits dans ce sens.

1.1Les Problèmes de sécurité :

En effet comme tout système distribué, les agents mobiles sont sujets aux menaces du réseau comme : la corruption de données, le masquerading, le déni de service (DoS denial of service), les écoutes, la répudiation ...etc

Ces problèmes ont des solutions comme : chiffrement, autorisation, authentification, non-répudiation, etc. mais qui sont adoptées aux systèmes repartis classiques (ex : Client-serveur).

En ce qui concerne les agents mobiles, il faut non seulement sécuriser les hôtes visités mais aussi les agents contre les falsifications.

Les problèmes de sécurité des agents mobiles se divisent en deux catégories : protection des sites d'accueil contre les agents malveillants et la protection des agents contre les sites malveillants ou même contre des agents malveillants. [MEN 04]

1.1.1 Protection des sites d'accueil

Dans les systèmes à agents, les hôtes reçoivent des agents et leur offrent des environnements d'exécution, par conséquent ils sont exposés au danger d'être bloqués (bourrage de tampons, boucle sans fin...etc.) ou d'être carrément endommagés (chevaux de Troie). Voici quelques solutions pour la protection des hôtes :

- ✓ La signature du code d'un agent par son créateur ou par une autorité de signature garantit son authenticité, son origine et son intégrité. Apres ces vérifications, le site d'accueil peut limiter l'accès aux ressources locales ce qui permet de réduire considérablement les effets malveillants sur ceci.

- ✓ Le bac à sable (SandBoxing) ou l'isolation est une autre technique qui consiste à isoler l'exécution d'un agent suspect de manière à rendre son exécution sans influence sur le reste de la plateforme. Ceci est fait en attribuant un espace virtuel à l'agent.
- ✓ Les interpréteurs sûrs constituent aussi une solution pour contrer le code malveillant. Java apporte de bonnes solutions pour la protection des hôtes grâce au modèle de sécurité de java, en effet, la machine virtuelle de Java utilise une vérification statique du code (syntaxe, typage) et un gestionnaire de sécurité afin de vérifier les instructions du ByteCode.
- ✓ L'historique de l'itinéraire est une solution qui consiste à garder l'historique du chemin composé des plateformes déjà visitées par l'agent. L'hôte d'accueil peut éviter l'exécution d'agents malveillants en cas de détection de plateformes malveillantes sur son historique.

1.1.2 Protection des agents

Les problèmes de sécurité des agents non-mobiles peuvent être, au moins théoriquement, résolus par les techniques de sécurité et protocoles actuellement utilisés. Cependant le problème majeur qui reste à résoudre est sans doute lié à la mobilité des agents dont l'utilisation des clés PKI est susceptible d'être une partie importante de la solution.

La sécurité des agents reste le challenge des systèmes mobiles. Les attaques sur les agents portent sur l'intégrité (modification non autorisée des données) et la confidentialité. Il n'existe pas de solutions universelles pour protéger un agent mobile lors de son déplacement d'un hôte à l'autre mais des solutions partielles dont la plupart visent beaucoup plus à détecter les attaques (machines malveillantes ou agents malveillants) que de les prévenir. Voici quelques solutions pour la protection des agents :

- ✓ Détection de falsifications d'agent : utiliser des fonctions d'évaluation d'état pour détecter si un agent à été falsifié durant son voyage.
- ✓ Protéger les agents contre des sites malveillants en utilisant du **matériel** prouvé bienveillants (certifié) [WBS 99].
- ✓ Pour protéger la confidentialité des parties de code et des données, il existe des solutions partielles qui chiffrent le code et les données. Ainsi le **cryptage des données** par un système de cryptage à clé publique (PKI) permet de les protéger du fait que seul le détenteur de la clé privée correspondant à la clé publique utilisée lors du cryptage peut y accéder (le créateur de l'agent est souvent le seul à avoir cette clé privée). Le **brouillage de code** [MAN 06] ou obscurcissement du code (technique qui rend le code source difficile à comprendre) protège l'agent contre les analyses des intrus.

✓ L'utilisation des nœuds de confiance est une autre solution pour protéger la confidentialité des données. Les données sensibles -préalablement cryptées- ne seront jamais décryptées pour être exploitées que dans l'un de ces nœuds de confiance.

2. Proposition

Dans la synthèse du chapitre trois, on a vu que les plates-formes d'agents mobiles qui existent sur le marché ne traitent pas le problème de protection des agents.

Par conséquent, notre objectif est de *concevoir* un **une plate-forme** ou **Framework sécurisé** pour les **agents mobiles**. Pour ce faire, nous nous sommes intéressés à l'architecture MP [MEN 04]. Cette architecture, basée sur un modèle d'interaction particulier, le modèle Seller-Buyer (SB, acheteur-vendeur), apporte une solution au le problème de la sécurité des agents mobiles. Nous nous proposons donc de réaliser une plate-forme d'agents mobiles sécurisée. Comme le développement d'une plateforme complète prend beaucoup de temps, il est intéressant d'opter pour une plate-forme FIPA tel que **JADE** car c'est un environnement distribué pour les agents mobiles fournissant les opérations de base (création et gestion des agents, communication, migration). De plus JADE, à l'instar des autres plates-formes FIPA, n'offre pas de solution pour la sécurité des agents mobiles. Il s'agit donc de modifier JADE en lui appliquant l'architecture MP. La plate-forme ainsi conçue sera désignée par JADE. Dans ce qui suit, nous allons présenter brièvement l'architecture MP et le modèle SB.

2.1 Un prototype d'architecture à base du modèle seller-buyer [Men 04] :

Si on pense différemment, on voit que le problème de sécurité du paradigme des agents mobiles doit être résolu, non pas en cherchant des solutions qui seront toujours provisoires (partielles), mais en **supprimant les deux points faibles** [MEN04:

- ✓ Sécurité des hôtes visités contre les agents malveillants ;
- ✓ Sécurité des agents mobiles contre les hôtes malveillants.

L'idée est d'interdire aux agents de migrer *directement* vers les hôtes et d'interdire aux hôtes d'accueillir des agents, tout en assurant les communications entre ces deux facteurs du système. Cela peut être réalisé en envisageant un site intermédiaire qui accueillera les agents mobiles provenant du site client et les mettra en relation avec le(s) site(s) destination(s). Cette nouvelle manière d'interaction constitue le modèle SB (Seller-Buyer) [MEN 04] (voir figure 4.1).

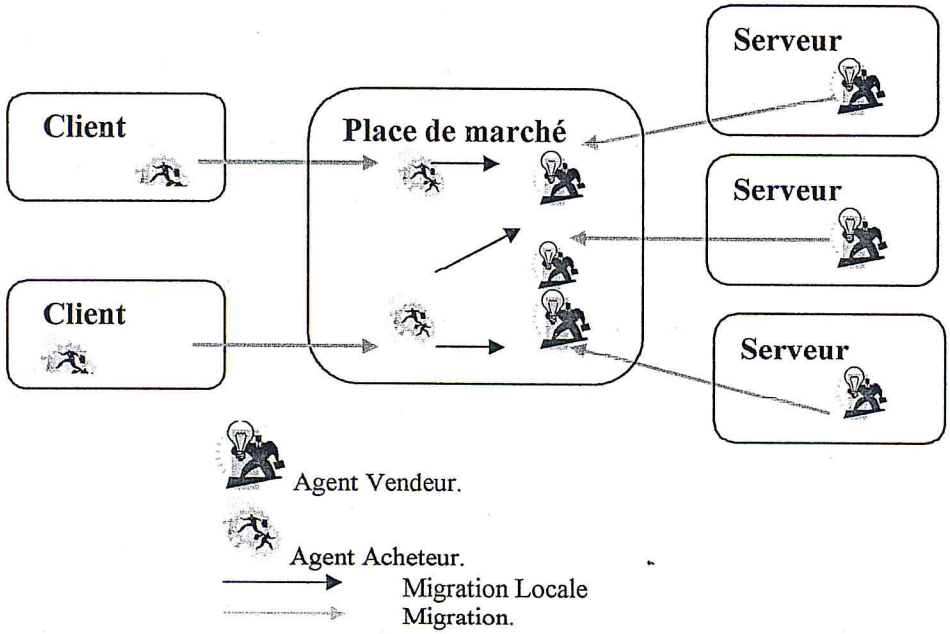


Fig-4.1. Modèle SB: Interaction des agents mobiles à travers une place de marché

L'implémentation de ce modèle donne fruit a L'architecture MP qu'on va résumer ci-dessous

2.2 L'architecture MP :

2.2.1 Les « Agents Providers » (AP):

Ce sont des sites qui offrent aux utilisateurs un environnement accessible (par internet par exemple) afin de créer leurs agents sécurisés les représentant. Ces sites sont responsables de recevoir les agents mobiles à la fin du processus, gérer les résultats et finalement détruire les agents qui ont fini leur mission.

2.2.1.1 Structure D'un AP :

Les principaux composants d'un AP sont les suivants :

- Un module serveur pour les clients/vendeur, auquel les utilisateurs se connectent pour télécharger l'applet nécessaire afin de remplir un formulaire de la requête/service.
- Un module de création des agents mobiles représentants des Client/Vendeur dans le système.

- Un module de gestion des agents qui permet d'inscrire les agents mobiles, leurs tâches, suivre leur migration.
- Une liste des services du système (l'ensemble des plateformes) pour vérifier si le service demandé existe ou pas.
- Un environnement d'exécution des agents mobiles.

2.2.2 Place de marché (Market Places):

Les Market Places (ou les MPs) sont des réseaux Intranet i.e. des réseaux locaux de plusieurs machines utilisant les techniques de communication d'Internet (IP, serveurs HTTP), ils sont dotés d'un firewall pour la sécurité du réseau et un catalogue interne (une base de connaissance).

Une Place de Marché se compose de :

- **Un MP Manager** : c'est un agent d'accueil des agents dans la MP qui est responsable de gestion de la place de marché et de mise à jour de l'annuaire du service MPDS (MP directory service) qui est la base de connaissance des ressources (services)
- **Le MP Security Service :** service qui joue le rôle d'un Firewall de la MP.
- **Les boutiques (e-shops)** : sont des hôtes qui hébergent un agent facilitateur et qui accueillent les agents vendeur d'un service et les agents clients correspondants. Les boutiques sont des lieux où les agents peuvent négocier afin de vendre ou d'acheter un service.

Les services hébergés sont classés selon des critères en classes. Une place de marché ne peut héberger qu'une seule classe de service. Une boutique n'héberge qu'un seul service qui appartient à la classe de service de sa MP. Ces services seront véhiculés par les agents vendeur de service. Une boutique contient les agents vendeur du même service.

2.2.3 Le Service MPNS : (MP Name Service)

C'est le service de localisation des MPs, il maintient les informations sur les MPs et leurs e-shops. Il travaille de la même manière que le DNS d'internet. Il reçoit une requête pour localiser les MP qui hébergent une certaine classe de service. Comme les MPs sont identifiées par des adresses unique (@ IP, ou FQDN –Fully Qualified Domain Name) le service MPNS répond à l'agent par une liste de noms pleinement qualifiés (FQDN) des MPs correspondantes. La base globale de ce service stocke les informations suivantes :

- Les IDs des places de marché dans ce cas les FQDN des MP.

➢ Un Catalogue Global qui contient une liste de services, la spécialisation et la charge de chaque MP. Il MAJ par les agents MPMs.

2.2.4 La sécurité de la plateforme :

Nous allons utiliser une approche basée sur la cryptographie qui sera couplé avec les techniques suivantes :

- Un Réseau fermé.
- La prévention et la détection de falsification des agents.

2.2.4.1 Le réseau fermé :

Notre système est un réseau fermé, aucune autre entité extérieure n'est permise d'y accéder. Et nous allons créer un agent représentant chaque site qui veut utiliser notre système. Notre réseau fermé se décompose en :

- ➢ Les sites AP.
- ➢ Les sites MPs
- ➢ Les serveurs MPNS.

2.2.4.2 La prévention de falsification d'agent:

Pour prévenir la falsification des agents nous allons utiliser un élément essentiel dans notre système. Il s'agit du site **TSA** (Trust and Security Authority) ou **autorité de sécurité et de confiance**. Cette autorité joue le même rôle que les **CA** (Certification Authority) ou autorités de certification. Elle a comme objectif de certifier les éléments du système et les agents.

Principalement, les clés sont utilisées pour crypter la communication entre un émetteur et un récepteur. Un agent doit être certifié pour qu'il puisse migrer dans le réseau.

Un agent peut être signé / crypté partiellement ou entièrement.

2.2.4.3 Protection Des MPs :

Chaque place MP contient un pare-feu qui peut filtrer les agents incidents. Ce pare-feu s'appelle MPSS (MP Security Service). Ce pare-feu contrôle les certificats présentés par les agents et il va vérifier avec la TSA.

Avant la migration d'un agent, la MP reçoit les informations de sécurité de l'agent (Clés publiques) pour pouvoir décrypter ce dernier.

La figure suivante est un résumé de la sécurité dans l'architecture MP.

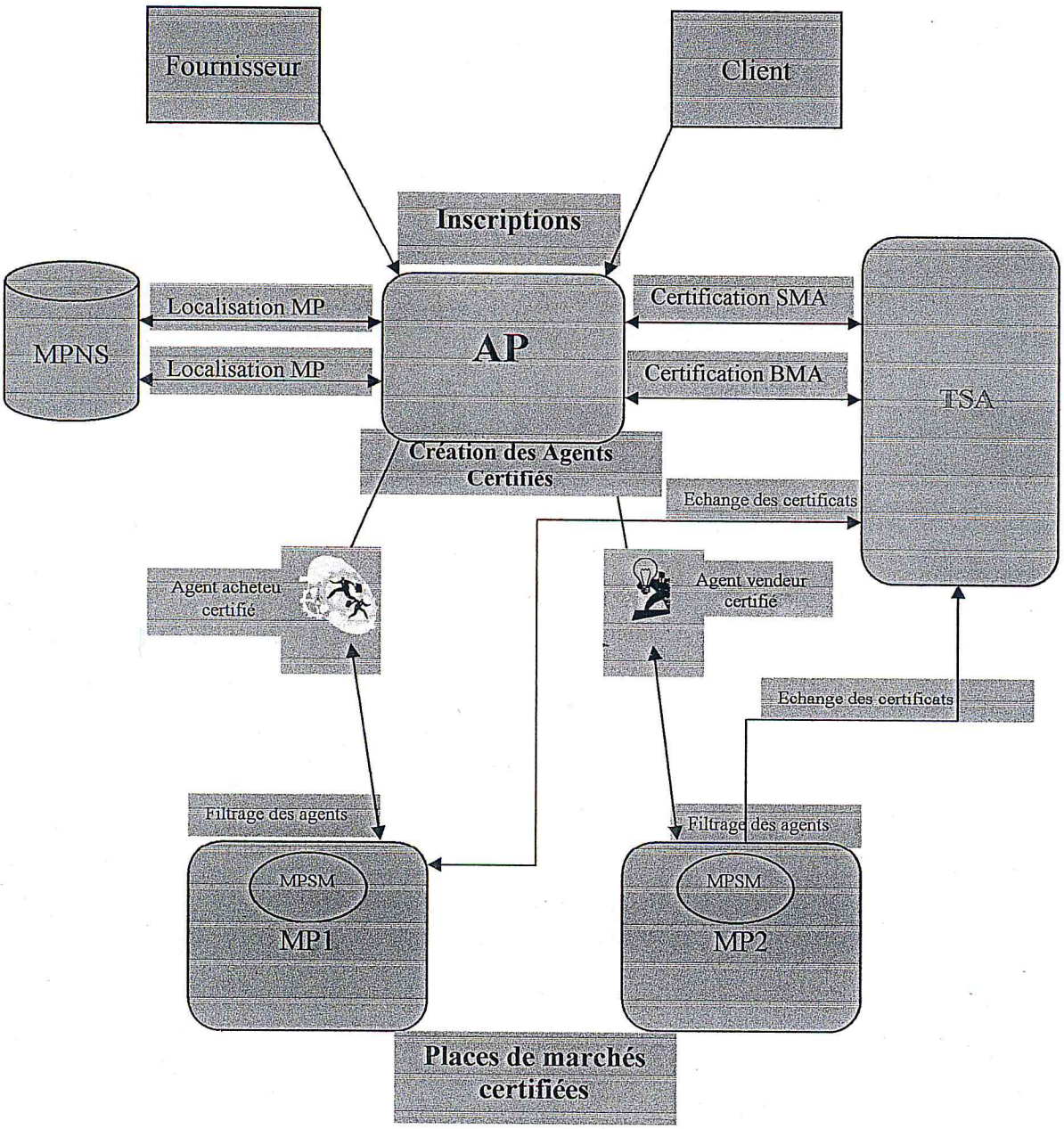


Fig-4.2 Modèle de la sécurité de la plateforme avec l'infrastructure PKI [MEN 04]

3. Conception de JADE-MP

Chaque E-Shop est représenté par un container. L'ensemble des e-shops de la même MP sont connectés au Main-container de la MP. Sur la machine principale de chaque MP une instance de la plateforme JADE est exécutée, même dans un E-shop il peut avoir une instance jade.

Donc chaque MP est représentée par une plateforme JADE (donc un Main-container) auquel sont connectés les containers des différents e-shops.

3.1 Conception des composants JADE-MP

Nous allons essayer de présenter les différents composants de cette architecture.

3.1.1 L'Agent Provider (AP):

Le site AP est un site sur lequel s'exécute une plateforme JADE dans laquelle sont créés les agents requis par les utilisateurs. Cette instance gère aussi le service MPNS qui se situe généralement sur une machine distante.

3.1.2 Les Place de marché (MP):

Une MP peut être une seule machine ou tout un réseau connecté (LAN), mais il y'a une seule machine principale c'est à dire un Main-Container. Les autres machines exécutent des containers (les E-shops) qui sont reliés au Main-Container.

3.1.3 L'E-Shop :

Chaque E-Shop est représenté par un container simple. Ce site héberge un ***service agent*** ou son agent facilitateur qui accueille les agents vendeur de services et les agents clients. Pour cela, cet agent facilitateur doit avoir le comportement ou le rôle de l'agent DF.
A ce niveau on peut –en plus- avoir d'autres types d'agents facilitateurs qui sont dits ***agents courtiers***. Ces derniers ont le rôle de faciliter la rencontre des agents vendeur et clients mais en favorisants quelques agents vendeur sur les autres.

La figure 4.3 montre l'architecture générale de JADE-MP.

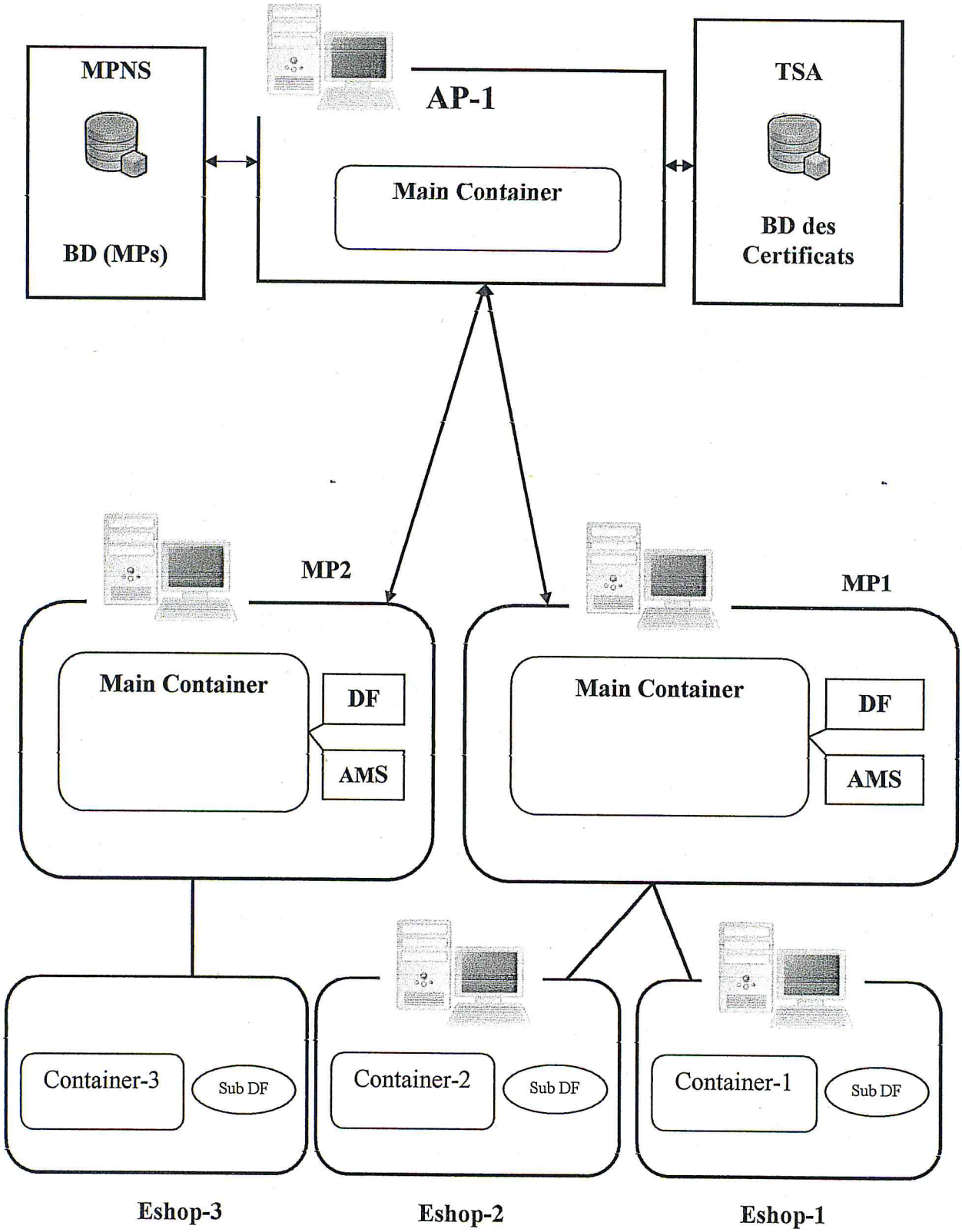


Fig4.3 Schémas général de l'architecture JADE-MP

3.2 Conception des agents JADE-MP

3.2.1 Les agents du système

Notre système comporte deux principaux types d'agents.

- Les agents statiques : sont pratiquement les agents qui veillent sur le bon fonctionnement du système. Ainsi on parle des agents de gestion des différents sites du système, des agents de sécurité du système et des agents des services du système.
- Les agents mobiles : sont les agents délégués des utilisateurs et des vendeurs. Ils se déplacent dans le système –migrent d'un site a l'autre via le réseau- pour accomplir leurs tâches.

Ces deux types héritent tous de la classe Agent du package ***jade.core.Agent***, mais chaque type diffère de l'autre par la manière d'implémentation des méthodes de base des agents, la logique suivie par les agents et par leurs comportements.

Les tâches que chaque agent JADE doit exécuter sönt appelés ***Behaviours*** (comportements), une fois un Behaviour créé, il est automatiquement planifié pour être exécuté.

3.2.2 Les agents statiques :

Dans cette catégorie on trouve principalement des agents facilitateurs et les agents de services.

Au niveau de l'AP on trouve les agents suivants:

3.2.2.1 AF_AP (Agent Facilitator AP):

Gère le site Agent Provider (AP), créé au lancement de l'AP. Il a pour rôle de:

- Signer les agents mobiles avant la migration.
- **Préparé les agents mobiles avant migrer vers une MP correspondante.**
- Décrypter les résultats récoltés par les agents mobiles et les afficher à l'utilisateur.
- Détruire les agents mobiles à la fin de leur cycle.

3.2.2.2 MPNSM (MP Naming Service Manager):

L'MPNSM est lancé au lancement de l'AP. Il implémente un ***cyclicBehaviour*** (un comportement répétitif issu de la classe ***CyclicBehaviour*** du package ***jade.core.behaviours***). Il a pour rôle de fournir les itinéraires MP adéquats aux agents mobiles.

Au niveau de la Place de Marché (MP) on trouve les agents suivants :

3.2.2.3 MPM (MP Manager) :

Assure la gestion et MAJ de la Base de données au niveau de la MP. Il S'exécute dans la place de marché –créé automatiquement au lancement de la MP- il a pour rôle de:

- Accueillir les agents mobiles.
- Décrypter les données des agents mobiles visiteurs de la MP.
- Mettre à jours la base de données de la MP.
- Crypter les résultats recueillis par les agents mobiles.

3.2.2.4 MPSM (MP Security Manager) :

Il assure la sécurité de la Place de Marché, il a le comportement d'un ***Pare-feu*** (Fire-Wall) d'un réseau Internet, automatiquement créé aussi au lancement de la MP. Il a pour rôle de:

- Recevoir les certificats des visiteurs et ceux de la place par le biais du TSA.
- Vérifier l'identité des agents désirants migrer vers son MP.

3.2.2.5 MPDSM (MP Directory Service Manager) :

Il gère le service MPDS au lieu du MPM pour alléger la tâche de ce dernier. Le MPDSM est lancé automatiquement au lancement de la MP. Tout comme le MPNSM, l'MPDSM implémente aussi un ***cyclicBehaviour*** qui est un comportement répétitif issu de la classe ***CyclicBehaviour*** du package ***jade.core.behaviours***. Il a pour rôle de:

- Fournir les listes des e-shops adéquates aux agents mobiles.
- Mettre à jours la base de connaissance du service MPDS ou annuaire MPDS.

Au niveau de l'E-Shop on trouve les agents suivants :

3.2.2.6 AF_E-Shop (Agent Facilitator E-Shop):

Cet agent est un peu particulier, vu qu'il n'hérite pas de la classe Agent du package ***jade.core.Agent*** comme tous les autres agents. Mais il hérite de la classe df du package ***jade.domain.df*** et ceci dans le but de créer des sous-DF, comme on l'a déjà évoqué précédemment, il n'est pas possible d'avoir plus d'un agent DF par plateforme, mais par cette pratique on peut avoir autant de sous DF que nécessaire.

En fait on a essayé de profiter au maximum des fonctionnalités déjà offertes par JADE et ne pas refaire ce qui existe déjà par hériter la classe Df et la modifier selon nos besoins.

L'AF_E-Shop peut donc offrir les mêmes fonctionnalités du DF, seulement il se contente de l'information concertante les agents hébergés par le container (l'e-Shop) ou il réside. Il a pour rôle de:

- Enregistrer les agents mobiles et leurs services dans la base locale.
- Fournir la liste des agents vendeur de service adéquats.
- Mettre à jours la base de données au niveau de l'e-shop.
- Gérer les transactions entre les agents acheteurs et vendeurs.

3.2.3 Les agents mobiles

Les agents mobiles ont la particularité d'avoir un modèle d'autonomie de décision assez avancé par rapport aux autres agents, par conséquent, chaque agent mobile implémente un automate d'état finis. Les automates à état finis dans les agents JADE sont des instances de la classe ***FSMBehaviour*** (Finite State Machine Behaviour) du package ***jade.core.behaviour.FSMBehaviour***. Comme son nom l'indique, un **FSMBehaviour** est un comportement, et comme cette classe hérite de la classe **CompositeBehaviour** alors il est possible de créer un **FSMBehaviour** par composition d'autres behaviours (appelés sous behaviours).

Les sous behaviours qui composent l'automate d'état finis de l'agent mobile sont écrits pour traiter une situation et passer ensuite au behaviour suivant en fonction du résultat du traitement fait.

4 Les classes d'agents :

Donc il existe deux types d'agent dans le système :

- Agents statiques
- Agents mobiles

4.2 Les agents facilitateurs (statiques):

(FA : facilitator agent) sont des agents statiques leur rôle est d'enregistrer les informations et faciliter le travail pour l'autre type d'agents (les agents mobiles)

On compte trois types d'agents facilitateurs (FA) :

- Facilitateur AP AP_FA
- Facilitateur MP MP_FA
- Facilitateur E-shop Eshop_FA

4.3 Les agents mobiles:

Sont des agents mobiles qui vont représenter un client dans notre système. Il existe deux types d'agent mobile ;

- Les Agents mobiles vendeur de service MA_S
- Les Agent mobiles Acheteurs MA_B

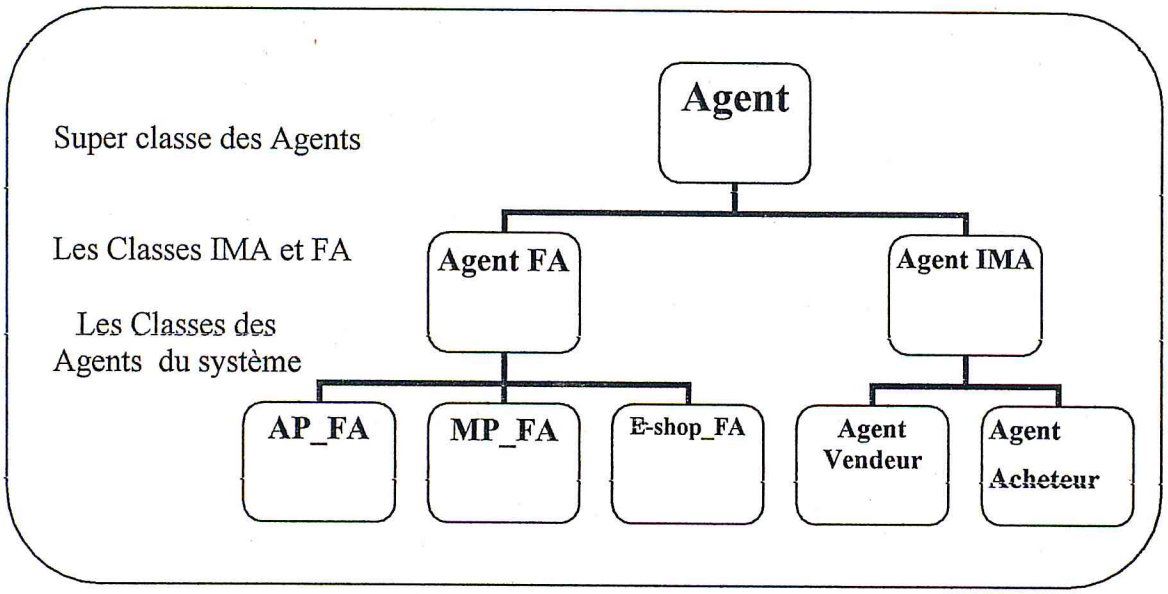
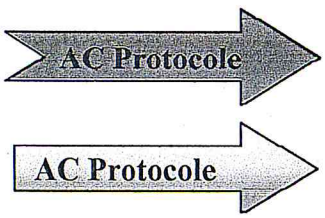


Fig-4.4. Le diagramme d'hiérarchie de classe des agents du Framework

5 Les diagrammes de classes des Agents du système :

Dans cette partie, nous allons présenter chaque classe d'agent avec un diagramme de classe en AUML (Annexe 2). Ceci pour pouvoir montrer les messages entrants et sortants de chaque agent qui est un des avantages d'AUML.



Messages entrants

Messages sortants

5.2 Les agents facilitateurs :

5.2.1 L'agent facilitateur d'AP

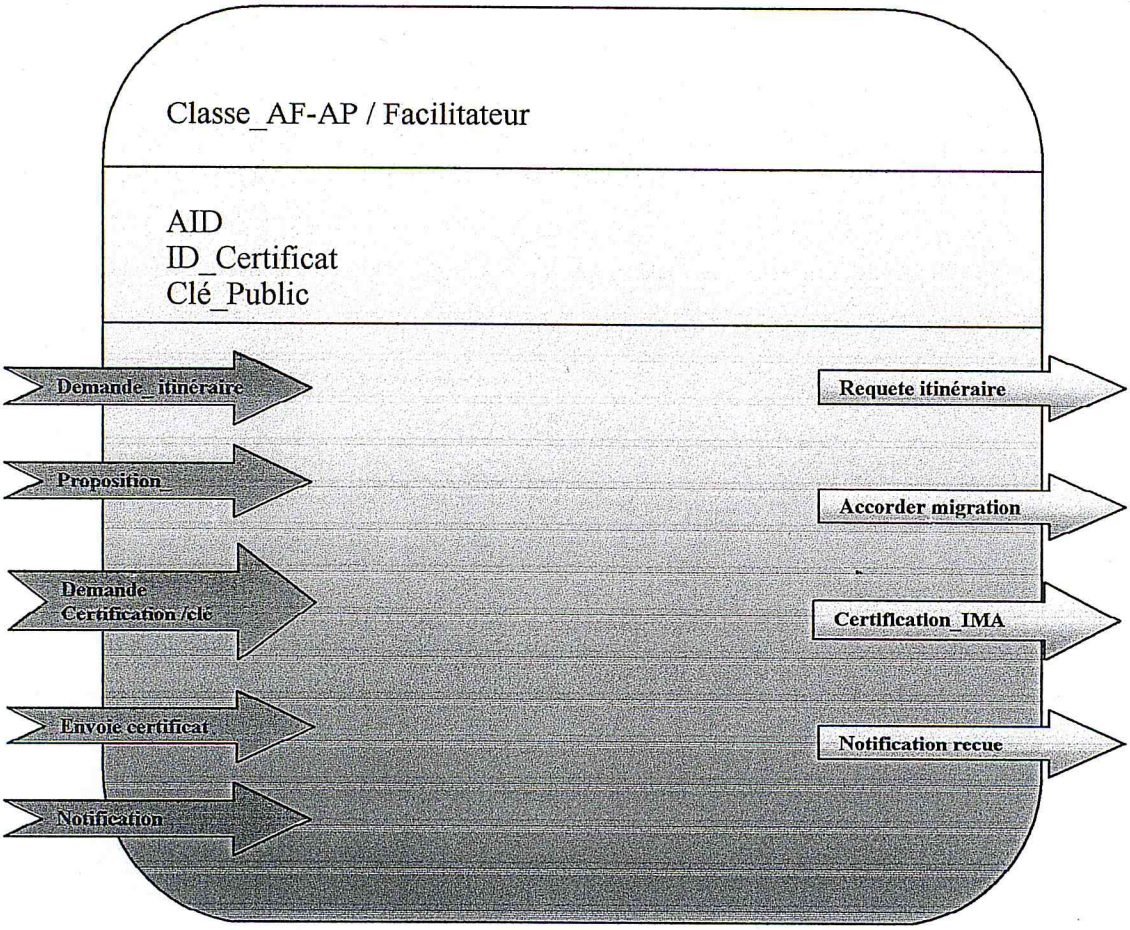


Fig-4.5.Diagramme AUML de l'agent AF_AP

5.2.2L'agent facilitateur d' MP

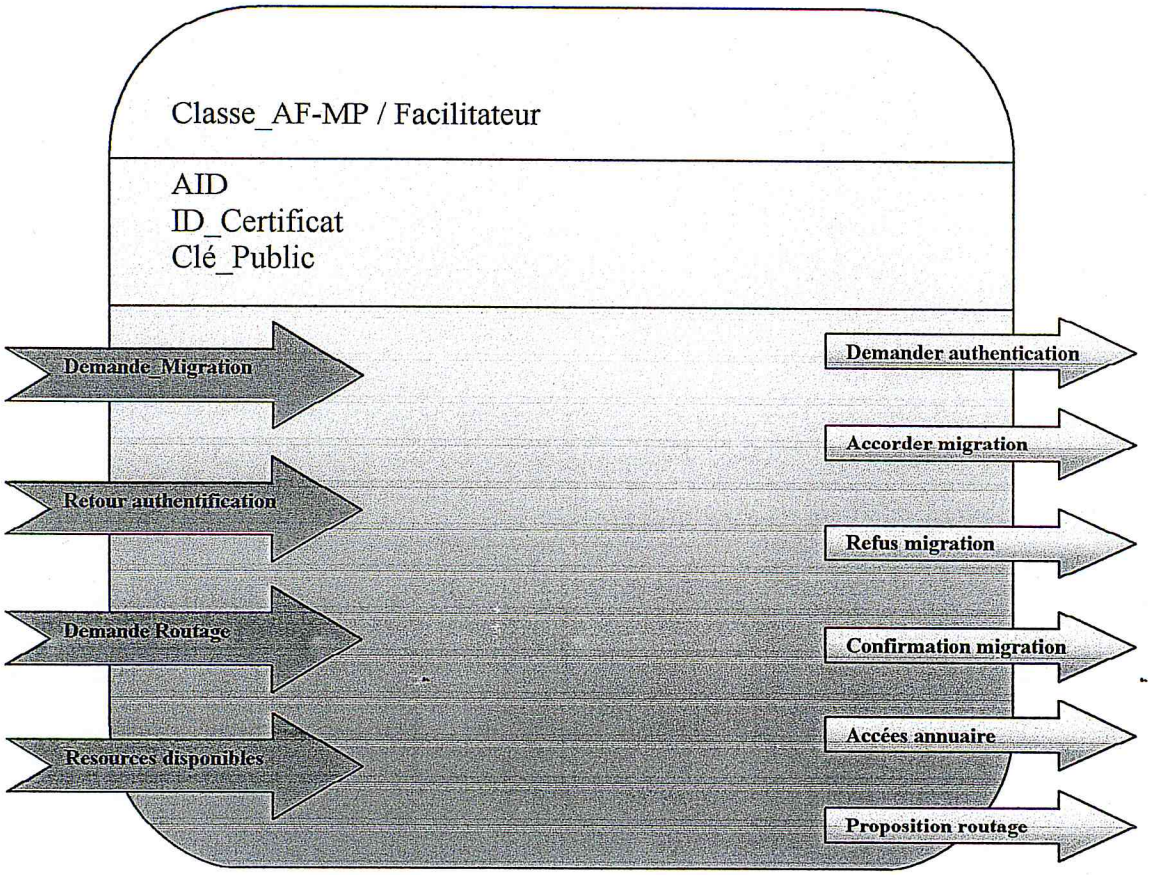


Fig-4.6 Diagramme AUML de l'agent AF_MP

5.2.3L'agent facilitateur d'E-shop

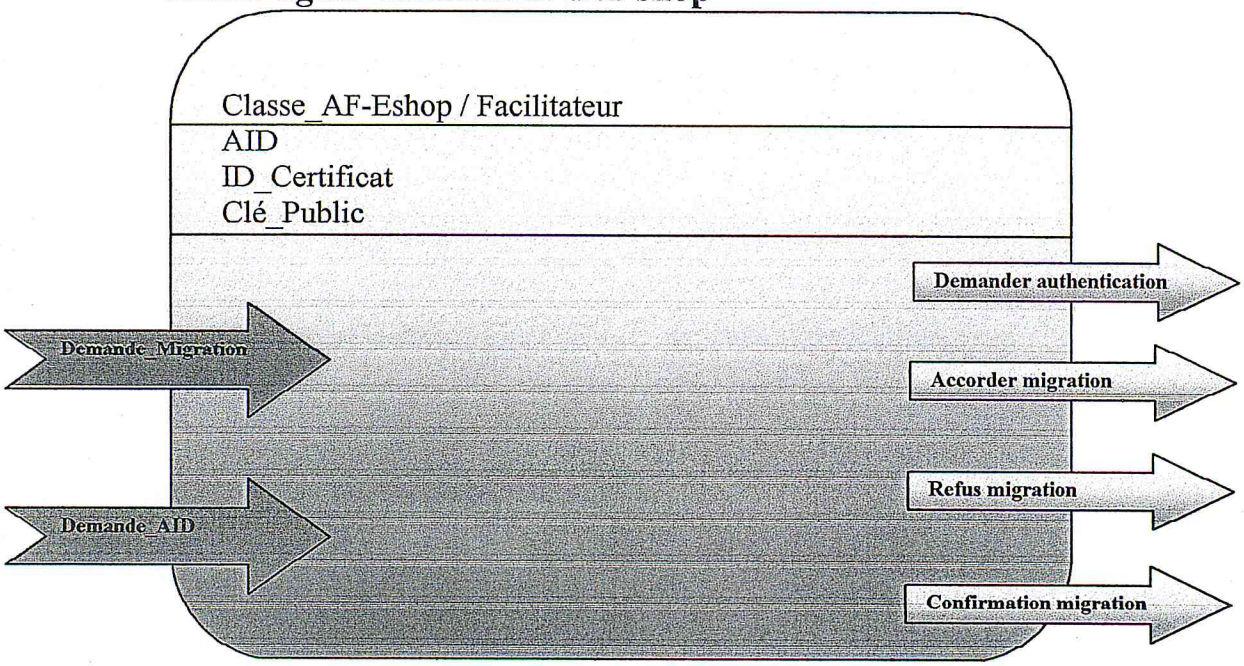


Fig-4.7. Diagramme AUML de l'agent AF_Eshop

Descriptions des agents facilitateur :

- AID _FA: l'identifiant de l'agent facilitateur
- ID_Certificat : l'identifiant du certificat d'agent IFA.
- Clé_public : La clé public du l'agent IFA délivré par le TSA.

5.3 Les agents mobiles xMAs :

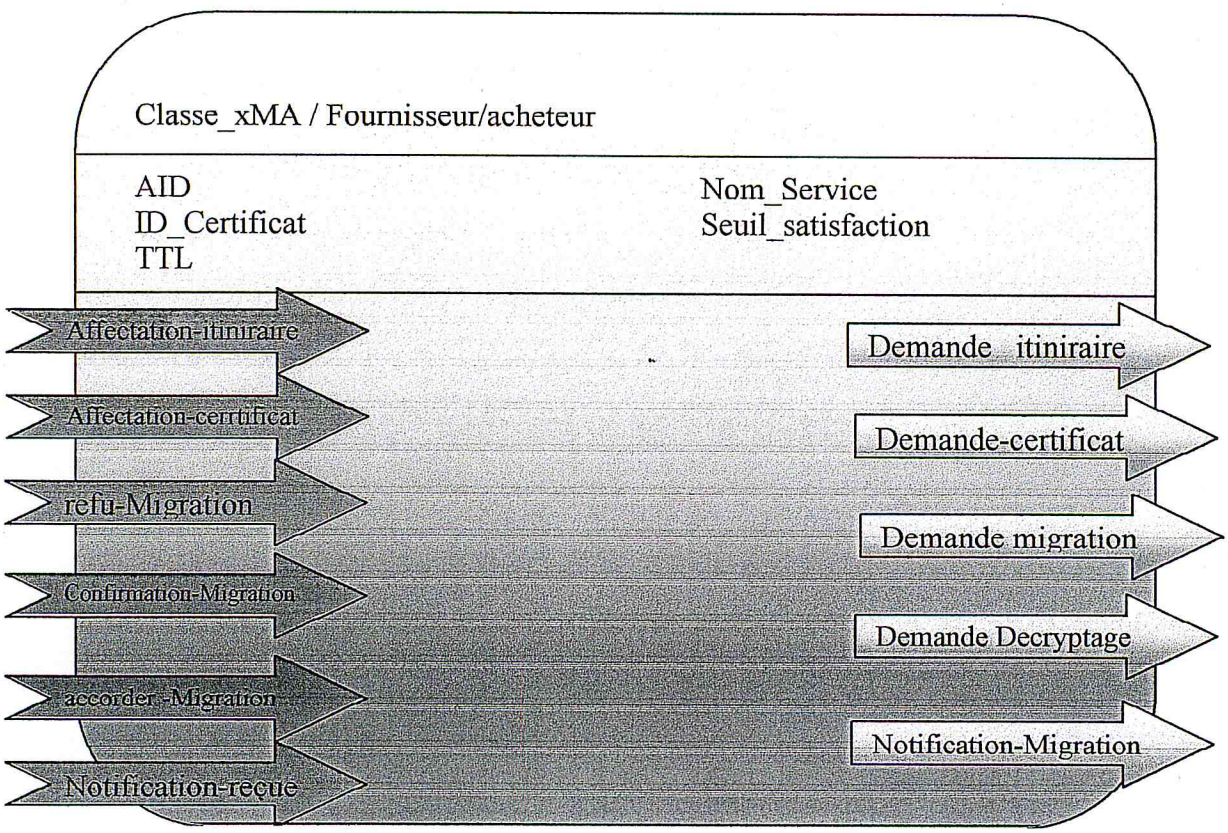


Fig-4.8. Diagramme AUML de l'agent xMA

Descriptions des agents xMA :

- AID _FA: l'identifiant de l'agent facilitateur
- ID_Certificat : L'identifiant du certificat d'agent xMA.
- TTl : Time to live la durée de vie d'agent mobile.
- Seuil_satisfaction : Le seuil de satisfaction d'agent vendeur ou client dans le système.
- Nom_service : Nom du service d'un agent fournisseur.

6 Communication entre les agents du système:

6.2 Au niveau d'AP :

L'interaction AP_FA et MP_FA avec un xMA (l'agent mobile) a pour but la préparation de ce dernier pour la migration vers une MP correspondante à sa Classe de service.

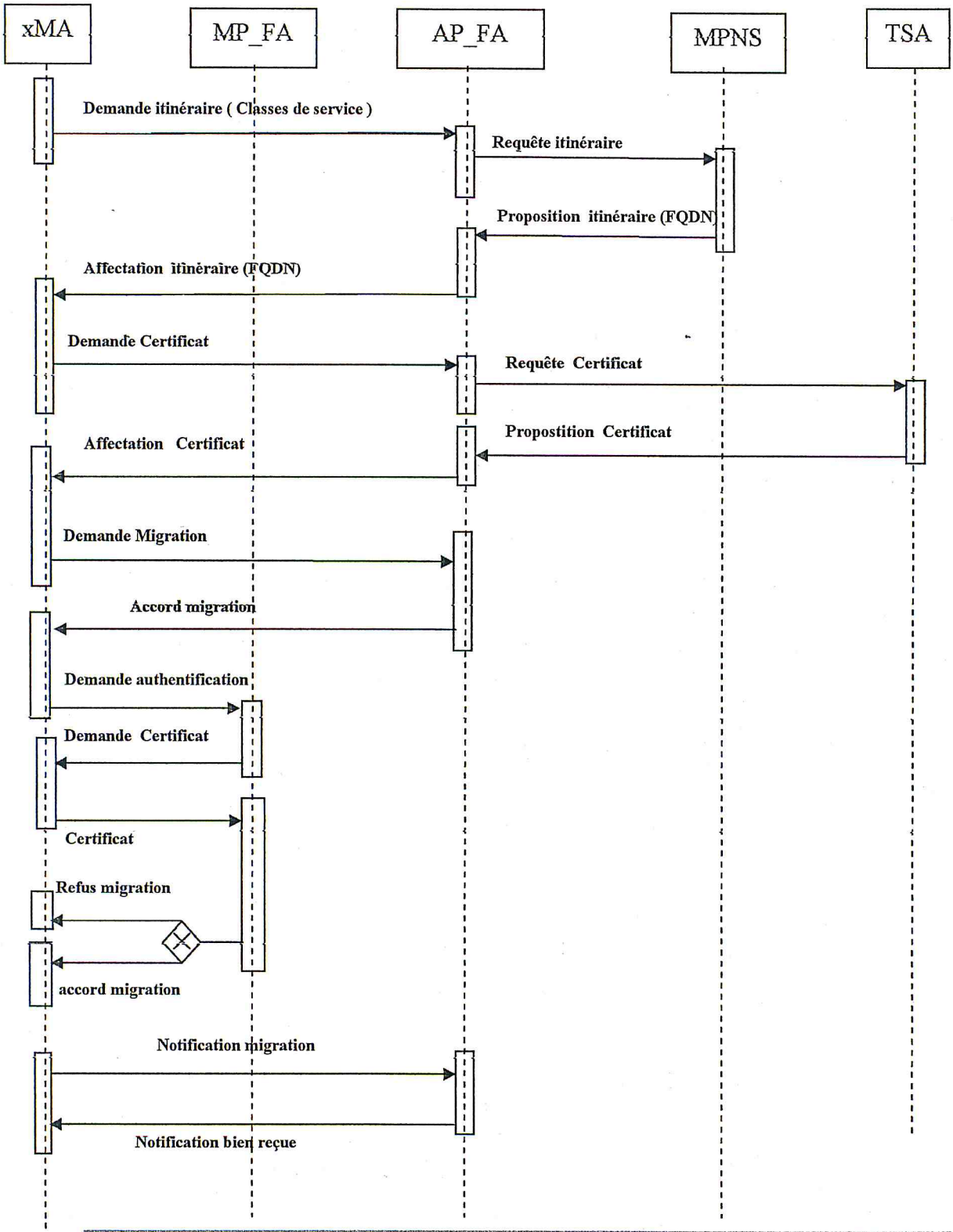


Fig-4.9 Interactions de l'agent mobile avec l'agent AP_FA et l'agent MP_FA

6.3 Au Niveau d'MP :

L'interaction MP_FA avec un xMA (l'agent mobile) a pour but la préparation de ce dernier pour la migration vers une E-shop correspondante à son service.

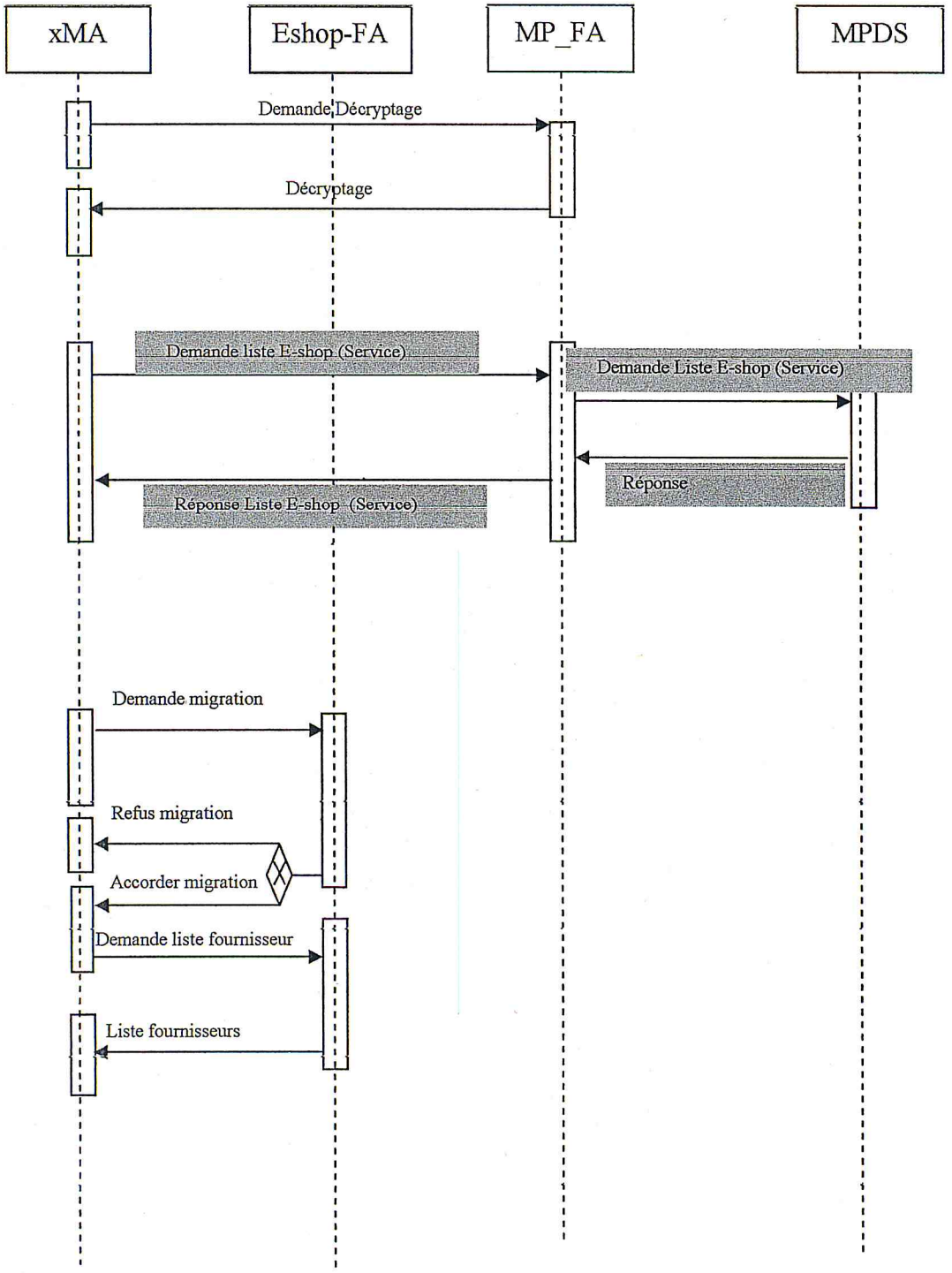


Fig-4.10. Interactions Au niveau d'un MP avec les Agents MP_FA et Eshop_FA

6.4 Au Niveau d'E-shop :

L'interaction entre les agents mobiles BMA-SMA : c'est la négociation entre l'agent vendeur et l'agent acheteur sur les conditions du service.

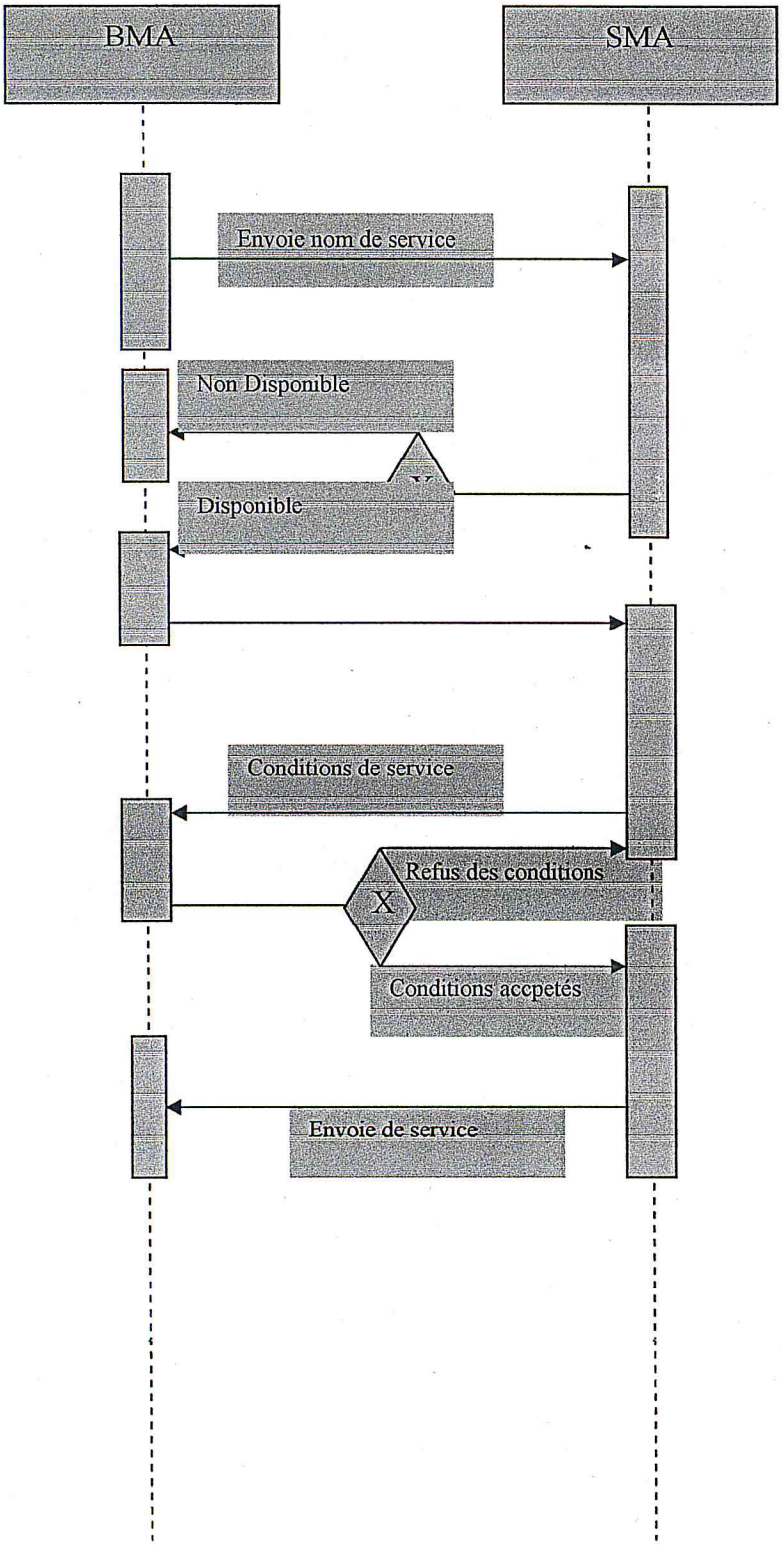


Fig-4.11 Interactions Au niveau d'un E-shop entre BMA et SMA

7. Conception des bases de données du système

Pour faciliter la localisation des services et des ressources nécessaires aux agents mobiles et pour des raisons de traçabilité du système entre autres les agents mobiles, on a construit une base de données qui stocke toutes les informations nécessaires sur notre système. Pour faciliter la gestion et la maintenance de cette base de données, on a pensé la distribuer sur l'ensemble des composants du système.

7.1 La base de données au niveau de l'AP

C'est la base comprenant les informations importantes concernant l'AP. Dans cette base on trouve les informations sur les utilisateurs, les agents et les composants ou entités du système. On trouve aussi à ce niveau la base de connaissance du service MPNS qui fournit aux agents l'itinéraire des MPs à visiter, ainsi que la base de données du service TSA.

Identification des classes :

Classe	Description
Service_Class	elle contient les informations sur toutes les classes ou catégories de service hébergées dans le système.
Service	elle contient les informations sur tous les services hébergés dans le système.
User ou utilisateur	elle contient les informations sur les utilisateurs inscrits dans le système.
Agent_info	elle contient les informations sur les agents créés par l'AP ou y résidants, elle se décompose en deux sous classe « Static_Agent _info » et «Mobile_Agent_info ».
AP	elle contient les informations sur les hôtes AP du système.
MP	elle contient les informations sur les places de marchés du système.
EShop	elle contient les informations sur les Eshops du système.

Tab4.1 Descriptions des classes au niveau d'AP

7.1.1 La base de connaissance MPNS_DB :

C'est la base du service MPNS, elle contient les informations suivantes :

- ID de classe de service;
- Liste des ID des MPs qui hébergent cette classe.

Identification des classes :

Classe « MP » : elle contient les informations sur les places de marchés du système.

Classe « Service_Class » : elle contient les informations sur toutes les classes ou catégories de service hébergées dans le système.

7.1.2 La base de données TSA_DB :

C'est la base qui se charge de garder les informations importantes du TSA.

Identification des classes :

Classe « Certificat » : Base des certificats octroyés ainsi que les modèles de certificat.

7.2 La base de données au niveau de la MP

A ce niveau on va devoir stocker les informations concernant les agents –statiques et mobiles- qui sont dans la place de marché. On trouve aussi la base de connaissance du service MPDS qui est le service permettant la localisation des e-Shops dans une MP.

Identification des classes :

Classe « Static_Agent _Info» : elle contient les informations sur les agents statiques de la place de marché.

Classe « Mobile_Agent_Info » : elle contient les informations sur les agents mobiles résidants actuellement dans la place de marché.

7.2.1 La base de connaissance MPDS_DB

C'est la base du service MPDS, ou annuaire MP. Elle contient les informations suivantes :

- Les ID des e_shops de la MP;
- Les informations sur les services et ressources disponibles sur tous les e-shops de la MP.

Identification des classes :

Classe « E-Shop » : elle contient les informations sur les e-shops de la MP.

Classe « Service » elle contient les informations sur tous les services hébergés dans la MP.

7.3 La base de données au niveau de l'E-Shop

Cette base contient toutes les informations sur tous les services et ressources disponibles sur l'e-shop.

Identification des classes :

Classe	Description
Statique_Agent_Info	contient les informations sur les agents statiques de l'e-shop.
Buyer_Agent_Info	contient les informations sur les agents acheteurs dans cet e-shop.
Seller_Agent_Info	contient les informations sur les agents vendeurs dans cet e-shop.
Request	contient les informations sur les demandes des agents acheteurs.
Offer	contient les informations sur les offres de service fournies par les agents vendeurs.

Table 2 TAB-4.2Descriptions des classes au niveau d'AP

Nous présentons ci-dessous les diagrammes de classe des différentes bases de données.

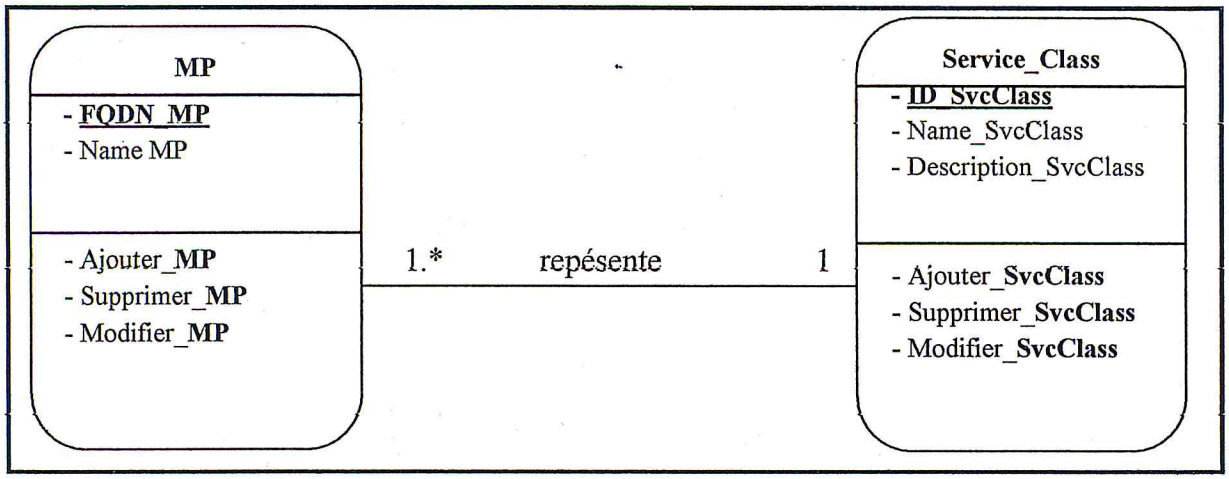


Fig-4.12.Diagramme de classe de MPNS_DB

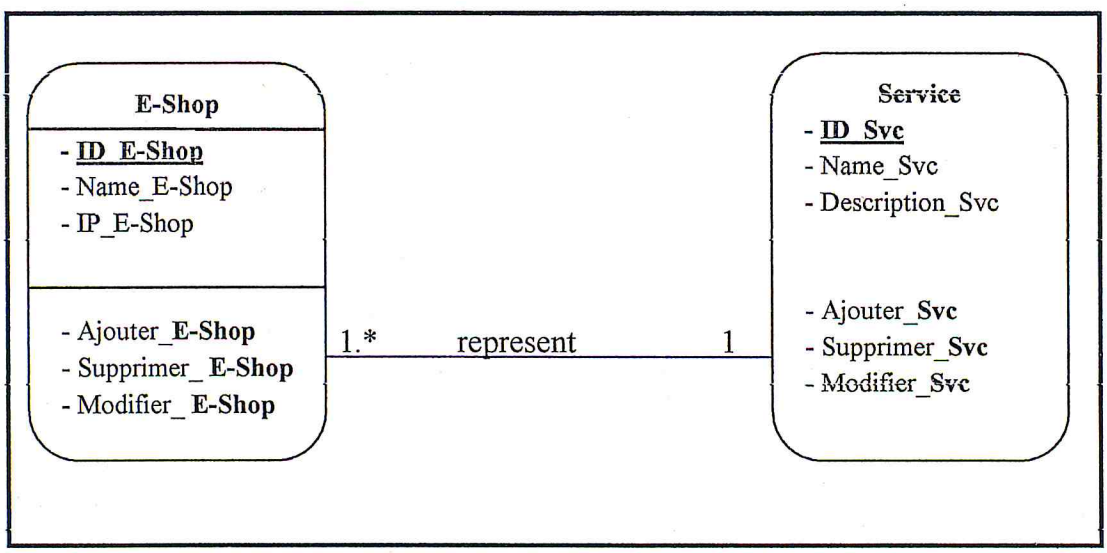


Fig-4.13.Diagramme de classe de MPDS_DB

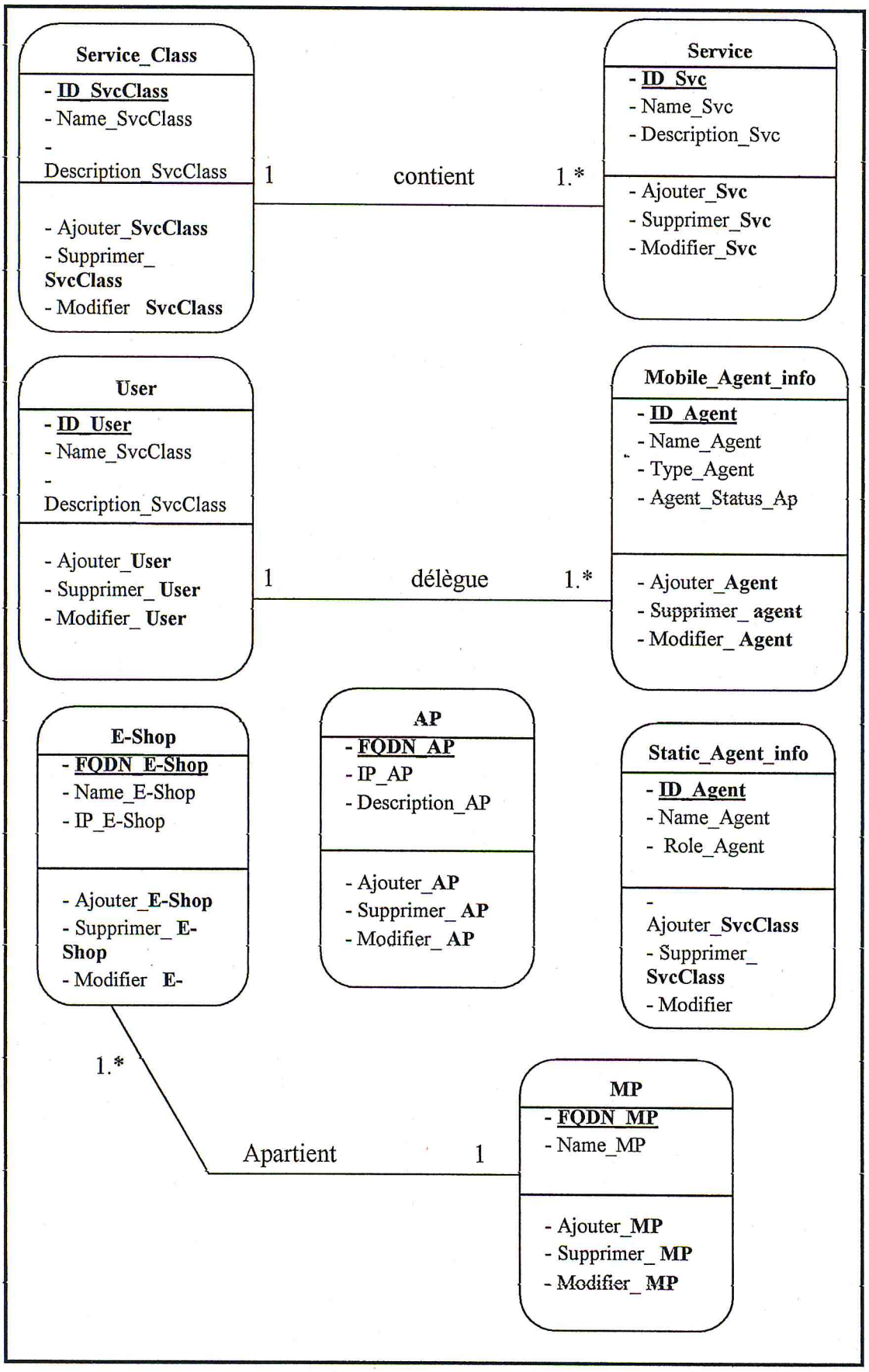


Fig-4.14.Diagramme de classe de la base de données au niveau de l'AP

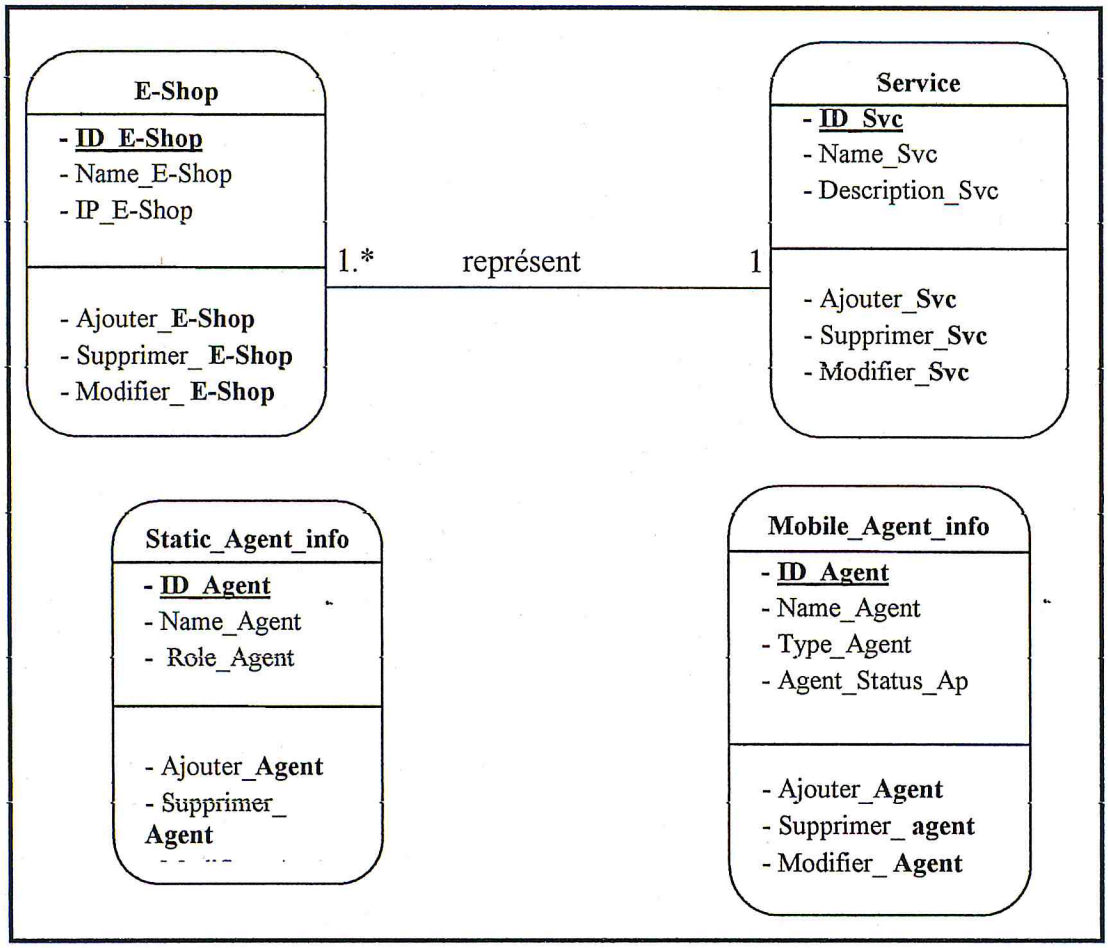


Fig-4.15.Diagramme de classe de la base de données au niveau de l'MP

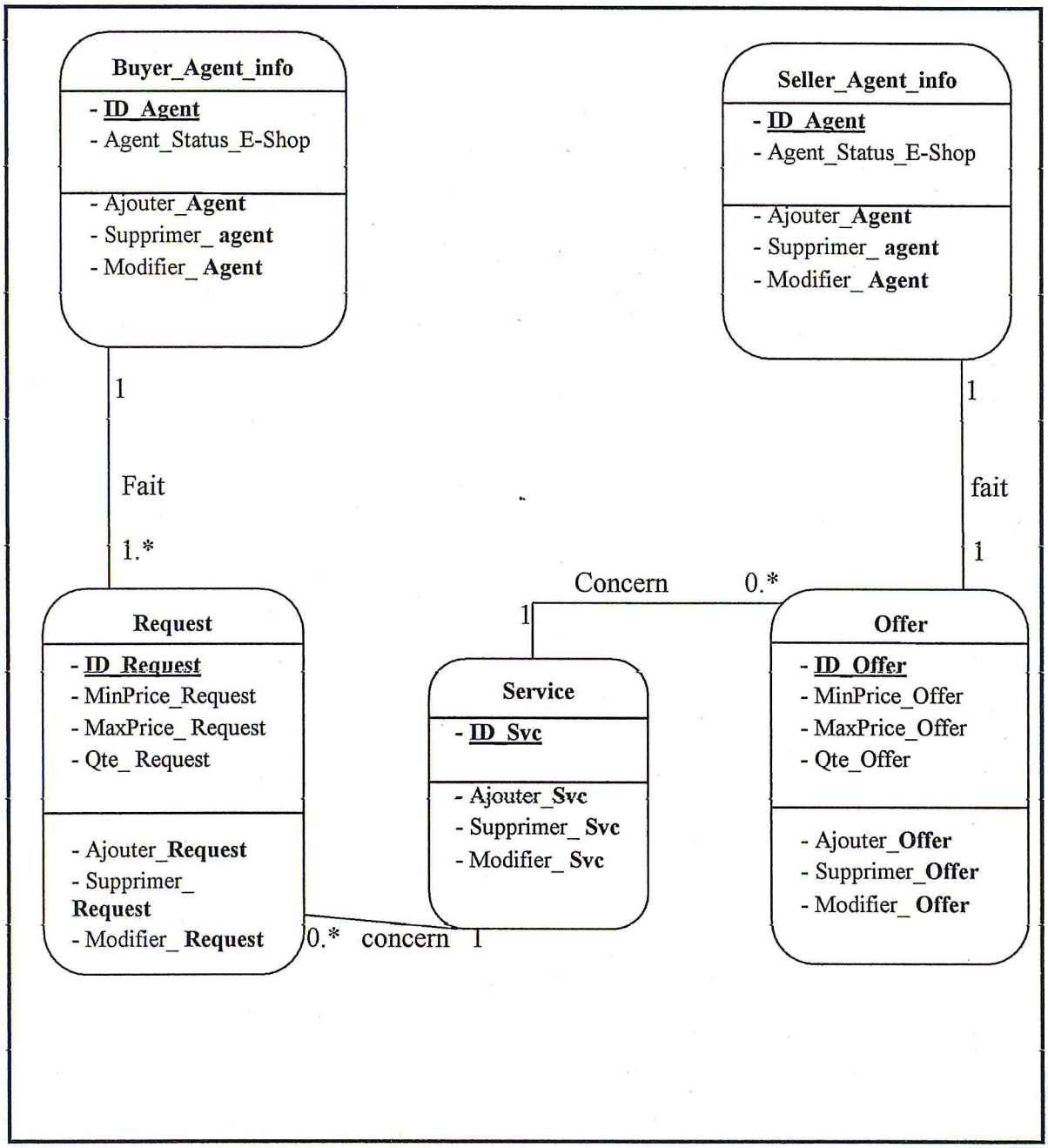


Fig-4.16.Diagramme de classe de la base de données au niveau de l'E-Shop

Conclusion :

Dans ce chapitre nous avons modélisé notre système et décrit sa conception, le but était d'introduire la sécurité (que la plupart –pour ne pas dire toutes- des plateformes d'agents mobiles n'offrent pas parmi la liste de services que propose chacune) à une plateforme d'agent mobile, à travers l'application de l'architecture MP à cette dernière. Jade a été la plateforme élue pour être modifié pour des raisons déjà invoqués dans les chapitres précédents. Nous avons décrit dans ce chapitre la conception de JADE-MP sur laquelle se base notre ***Framework sécurisé pour les agents mobile***, qui est le but principal de cette étude.
Après avoir conçu notre système, nous pensons que le Framework qu'on vient de concevoir offre un certain niveau de sécurité en plus des services de base offerts par la majorité des plateformes des agents mobiles. Et avec ce Framework nous pensons avoir éliminé les faiblesses des agents mobiles qui retardent leur adoption, en plus de pouvoir mieux répondre aux objectifs des applications actuels.
Le prochain chapitre portera sur l'aspect technique de notre Framework. Nous donnerons les détailles sur comment ce système, avec tous ces composants, a été implémenté.

Chapitre 5 :

Implémentation

Introduction :

Un projet informatique requiert, dans sa phase de développement, la mise en œuvre de différentes technologies. Cette partie présente les choix effectués pour la réalisation de notre projet en termes de langages et d'outils nécessaires à l'implémentation et décrit les différentes classes développées pour mettre en ouvre les composants de notre plate-forme.

Certain choix ont été pris bien avant le début de la réalisation de la plate-forme et ceci dans des buts précis comme le langage de programmation utilisé est le JAVA vue son entière compatibilité avec la plate-forme JADE - qui elle aussi est choisie au préalable - pour l'écriture des agents.

1. Technologies utilisées

Nous présentons ici brièvement les technologies et outils utilisés durant la phase de réalisation.

1.1 Langage de programmation

Nous avons implémenté notre plateforme avec le langage de programmation JAVA vu que ce dernier est un langage simple, intuitif, orienté objet, performant et dynamique. C'est aussi un langage multiplateforme disposant d'une JVM (Java Virtual Machine) lui permettant de s'exécuter dans des environnements hétérogènes en permettant une Independence envers les réseaux et les systèmes d'exploitation.

1.2 Serveur de bases de données

MySQL est un système de gestion des bases de données (SGBD) rapide et flexible. Pour éviter les surprises des premières utilisations de nouveau SGBD nous avons choisi MySQL, à la place de HSQLDB, car il est simple, efficace et satisfait nos besoins.

HSQLDB est un SGBD que les concepteurs de Jade recommandent d'utiliser dans les systèmes qui ont besoin d'une persistance de données manipulées par les agents.

HSQLDB est écrit entièrement en JAVA, il est disponible sous la norme BSD.

Les deux points forts de HSQLDB sont la légèreté et la rapidité.il est aussi apprécié par sa taille réduite, ce qui lui permet d'être exécuter entièrement en mémoire.

1.3 Serveur Web :

Apache Tomcat est un conteneur de servlet J2EE. Issu du projet Jakarta, Tomcat est désormais un projet principal de la fondation Apache. Tomcat implémente les spécifications des servlets et des JSP de Sun Microsystems. Il inclut des outils pour la configuration et la gestion, mais peut également être configuré en éditant des fichiers de configuration XML. Comme Tomcat inclut un serveur HTTP interne, il est aussi considéré comme un serveur HTTP.

1.4 Plateforme d'agents :

Dans le choix de plateforme on a eu le choix entre différentes et plusieurs plateformes d'agent mobile mais on a choisis JADE et on a déjà argumenté pourquoi JADE. On a justifié par le fait que JADE est open source et est facile à modifier car elle est développée en Java. JADE est FIPA compatible et dotée par un API puissant.

1.4.1 L'environnement d'exécution des agents :

Nos Agents s'exécutent sur des environnements spécifiés à JADE les « Containers » (conteneurs). Les Containers ont une architecture distribuée et hiérarchique au travers d'un Main-container servant de container de départ auquel viendront se rattacher l'ensemble des autres containers répartis.

Il existe aussi une possibilité d'associer plusieurs plates-formes en reliant leur Main-container mais en limitant leurs fonctionnalités.

En effet, les régions définies par les Main-container autorisent uniquement l'échange de messages mais pas la migration des agents. Tous ces éléments sont gérés par des agents qui représentent l'entité de base de la plate-forme.

1.4.2 Architecture logiciel de la plate-forme JADE

JADE reprend donc l'architecture de l'Agent Management Reference Model proposé par FIPA. Les différents modules présentés dans la figure suivante sont présentés sous forme de services. Les services de base proposés sont le Directory Facilitator (DF) et l'Agent Management System (AMS). Il est possible de lui demander de tenir en plus le service de Message Transport Service (MTS) pour communiquer entre plusieurs plates-formes. Mais ce service sera chargé à la demande pour ne conserver par défaut que les fonctionnalités utiles à tout type d'utilisation.

L'agent est l'acteur fondamental de la plate-forme, un Agent Identifier (AID) identifie un agent de manière unique. Le DF est un composant qui fait office d'annuaire. C'est un service de « pages jaunes » qui permet de mettre en relation les agents avec leurs compétences. Un agent peut enregistrer ses compétences dans le DF ou interroger le DF pour connaître les compétences proposées par les autres agents. L'AMS est un autre composant important car il contrôle l'accès et l'utilisation de la plate-forme et maintient un répertoire contenant les adresses de transport des

agents de la plate forme. Ce service est plus un service de type « pages blanches » qui effectue la correspondance entre l'agent et l'AID.

Chaque agent doit s'enregistrer à un AMS pour avoir un AID. Il n'y a qu'un AMS par plate-forme. Le MTS est une méthode par défaut de communication entre agents de différentes plates-formes. Cela permet l'interconnexion entre systèmes hétérogènes ou tout au moins de système ne communicant pas de la même façon. L'Agent Platform (AP) constitue l'infrastructure physique sur laquelle se déploient les agents. Il contient le DF, l'AMS et le MTS. Lorsqu'on parle de AP, on inclut souvent le matériel électronique, l'OS, le software et les composants cités ci-dessus avec les agents. Enfin, l'Agent Identifier (AID) est un identifiant précis d'un agent. On lui donne plusieurs paramètres tels que l'adresse de transport, l'adresse de service de résolution de nom, … Un exemple est : name@HAP (Home Agent Platform)

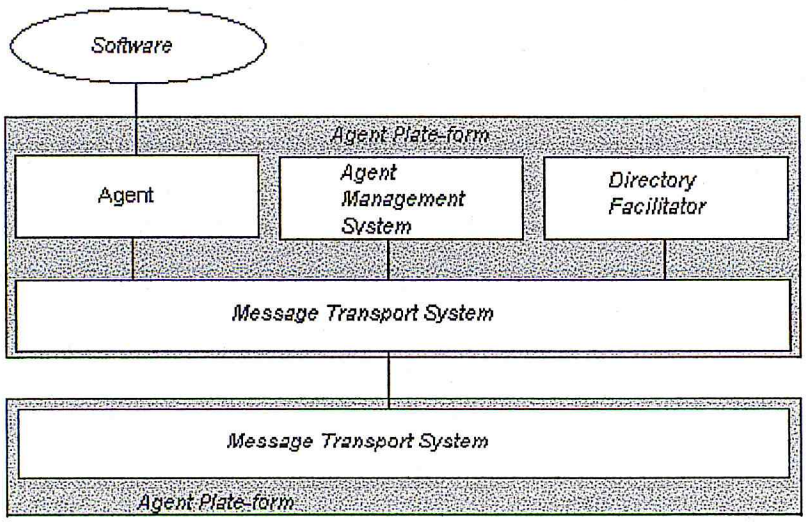


FIG- 5.2 : ARCHITECTURE LOGICIEL DE LA PLATE-FORME JADE

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

- Enregistrer un agent dans les pages jaunes (register).
- Supprimer un agent des pages jaunes (deregister).

- Modifier le nom d'un service fourni par un agent (modify).
- Rechercher un service (search).

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

1.4.3 Le protocole Contract-Net de JADE

Le Protocole Contract-Net (CNP) a été défini par Smith. Il est un mécanisme de négociation par appel d'offre (ou Contract) entre deux types d'agents : l'agent gestionnaire et les agents contractants. L'agent gestionnaire, souhaitant sous-traiter une tâche qu'il doit accomplir, est l'initiateur du contrat. Chaque agent contractant est un agent auquel on propose ce contrat. Le fonctionnement du CNP à été décrit dans. Ce protocole très souvent utilisé, a été normalisé par l'organisation FIPA et implémenté dans la plate-forme JADE.

2. La Mobilité des agents JADE:

Dans sa version de base, jade fournit un service de mobilité qui permet à un agent de migrer d'un container vers un autre container dans la même plateforme cette forme de migration est appelée *migration intra-plateforme*

On peut faire migrer un agent grâce à la méthode : DoMove(Destination Location)

Tel que : Location est une interface (au sens objet du terme) implémenté dans ce cas la par la classe ContainerID (une instance de cette classe représente un Container).

Dans cette version de base, il n'est possible pour un agent que de migrer d'un container à un autre sur la même plateforme (et forcement sur le même réseau). Pour faire une migration entre deux plateformes (par exemple entre plateforme1 et plateforme2 dans le schéma précédent) il faut utiliser l'add-on IPMS.

La migration entre les plateformes avec l'add-on IPMS se fait toujours par la méthode DoMove() précédente, mais cette fois ci le paramètre de destination devient une instance de classe PlateformID

Donc JADE offre trois types de mobilité :

Type	Nombre de plateforme	Type de réseau
Type1	1	Sur une seule machine
Type2	1	Réseau local
Type3	plusieurs	Réseau étendu

Tableau-5.2 types de mobilité dans jade.

3. Réalisation du Framework :

Notre system se compose d'un ensemble de plateformes JADE distribuées sur plusieurs machines. Nous allons expliquer comment une plateforme JADE peut être lancée pour pouvoir ensuite parler de l'implémentation des différents composants du système.

Mise à jour de la variable java CLASSPATH

La première chose qui doit être faite pour pouvoir utiliser JADE est la mise à jour de la variable d'environnement java CLASSPATH, afin que les bonnes librairies soient trouvées et utilisées lors de la compilation ou de l'exécution des programmes JADE.

La mise à jour de la variable CLASSPATH peut être faite visuellement sous Windows ou par la commande suivante :

On suppose que JADE a été décompressé dans le « c:/ »

```
set CLASSPATH=%CLASSPATH%;.;c:\jade\lib\jade.jar;
        c:\jade\lib\jadeTools.jar; c:\jade\lib\Base64.jar;
                c:\jade\lib\iiop.jar
```

Lancement d'une plateforme JADE

JADE offre deux façons possibles pour lancer une plateforme. On distingue :

Façon 1 : par l'invocation de la méthode « createMainAgentContainer() » définit comme suit

« **AgentContainer jade.core.Runtime.createAgentContainer(Profile arg0)**». Cette methode renvoie un conteneur d'agent et prend comme paramètre un profile sur lequel on doit spécifier les différentes caractéristiques de notre plateforme à être lancer, comme le nom de la plateforme par exemple.

Façon2 : par les commandes Shell. En effet JADE offre une classe jade.Boot qui permet le lancement des différents composants JADE comme les conteneurs (principale ou simple), les agents …etc. la commande suivante permet de lancer une plateforme JADE « java jade.Boot ».

Plusieurs paramètres peuvent être specifier comme

–gui : qui permet de lancer, en même temps qu'au lancement de la plateforme JADE, l'agent de l'interface graphique de la plateforme.

–container : qui permet de lancer un simple conteneur.

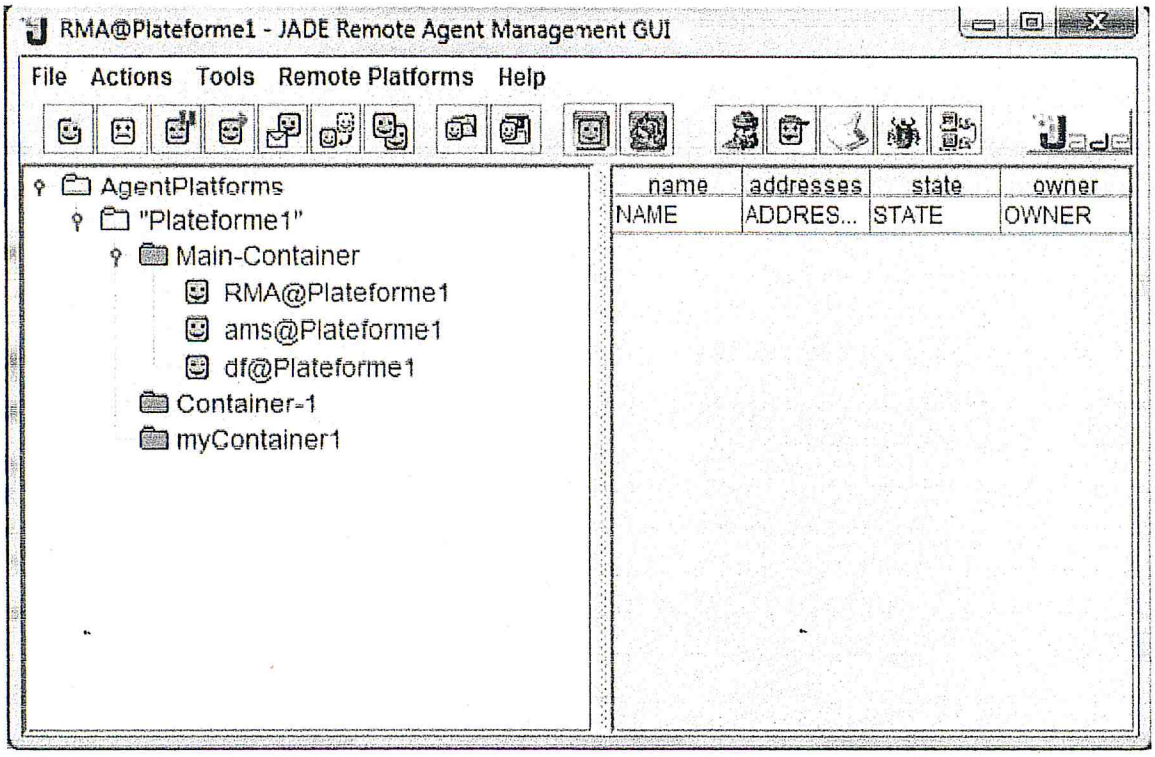
Par exemple si on exécute les commandes suivantes :

java jade.Boot –gui –nom plateforme1

java jade.Boot – container

java jade.Boot –container –name-container myContainer1

un plateforme appelée plateforme1 sera lancée puis un simple conteneur sera lancé et enfin un autre simple conteneur appelé myContainer1 sera lancé. La figure suivante montre les résultats de l'exécution de ces commandes.



NB : si la variable java CLASSPATH n'a pas été mise à jour au début, on doit le faire a chaque exécution d'une commande JADE, par exemple java jade.Boot –gui devient java –classpath «JADE-Classes» jade.Boot –gui.

Lancement des agents

Comme le lancement d'une plateforme, le lancement des agents aussi peut être fait par :

Façon1 : les programme java et dans ce cas nous utiliserons la méthode « **AgentController jade.wrapper.ContainerController.createNewAgent(String arg0,String arg1, Object[] arg2)** »

Qui prend comme paramètres :

arg0 : le nom de l'agent.

arg1 : la classe de l'agent.

arg2 : la liste des arguments de l'agent.

Façon2 : les commandes Shell. Par exemple on peut compiler et lancer un agent comme suit :

Javac –classpath «JADE-classes» HelloWorldAgent.java

Bien sûrs la variable java CLASSPATH doit pointer sur les librairies JADE pour la compilation.

Java –classpath «JADE-classes»;. jade.Boot Smith :HelloWorldAgent

Cette commande lance une plateforme JADE, et lance dedans un agent qui a pour nom local « Smith » et a pour classe HelloWorldagent.java

La variable java CLASSPATH **doit aussi pointer sur la classe HelloWorldAgent** pour pouvoir lancer l'agent.

3.1 L'Agent Provider (AP):

Le module serveur pour les clients du site AP, auquel les utilisateurs se connectent, via internet par exemple, offre une IHM qui permet aux utilisateurs de saisir leurs requêtes.

Le site AP est le premier site à lancer. Avant de le lancer ce site a besoin de :

1. Un serveur web lancé : il faut lancer le serveur web s'il est désactivé.
2. Un SGBD MYSQL s'il est installé : il faut l'installer s'il n'est pas.
3. Vérifier si Il y'a une instance de JADE est déjà exécutée : il faut fermer toute instance exécutée.

   Dans ce qui suit la partie du code qui lance un **AP** avec sa **BD Global** et le Service **MPNS.**

```
…
if (webServerCheck()) {
            //initComponents();
            try {
  Runtime.getRuntime().exec("cmd /c start java =classpath " + classpathValue
+ " jade.Boot -gui " + mobilityservice +" MPNS_F:MPck.MPNS_F");
Runtime.getRuntime().exec("cmd /c start "+ jadeURL
+"\\MySQL\\restorAP_DB.bat");
                }
  catch (IOException ex)
                        {
                        }

…
```

Fig-5.2 Code de lancement d'un AP

3.2 Localisation des MPs :

Le Service MPNS est responsable de la localisation des MPs. Un agent voulant récupérer les adresses d'un ensemble de MP hébergeant à une classe de service envoie un message de type REQUEST à l'agent facilitateur de l'AP AP_FA du format suivant

```
(REQUEST
  :sender( agent-identifier
                  :name NomAgentChercheur@NomDeLaMachine:1099/JADE
                  :addresses (sequence http://NomDeLaMachine:7778/acc )
             )
  :receiver(set ( agent-identifier :name MPNS@NomDeLaMachine:1099/JADE
  :content  "classeDeServiceRecherché"
  :encoding  ENCODING
  :language  SL0
  :protocol  fipa-query
 )
```

L'agent AP_FA fait une recherche sur la base de données du MPNS après il envoie le message suivant

```
(INFORM
:sender( agent-identifier
           :name MPNS@NomDeLaMachine:1099/JADE
           :addresses (sequence http://NomDeLaMachine:7778/acc )
         )
:receiver        (set    (    agent-identifier     :nameNomAgentChercheur
@NomDeLaMachine:1099/JADE ) )
:content  "adresseIPdesMPs"
:encodingENCODING
:language  SL0
:protocolfipa-propose
 )
```

Le code suivant est une partie du code de l'agent service MPNS qui fait la recherche sur la BD MPNS :

```
...
try {
                    // TODO code application logic here
con = DriverManager.getConnection("jdbc:mysql://localhost/ap_db", "root",
"root");
                } catch (SQLException ex) {

                    System.out.println("failed 2 open connection...!");
                }
              try {
           Statement st = con.createStatement();
            ResultSet res = st.executeQuery("SELECT * FROM mps  ");//

            while (res.next()) {
                        String IP2 = res.getString("MP_@IP");
                        String clsvc = res.getString(3);


                     if (clsvc.equalsIgnoreCase(Cl_Sr))
                     {
                           IP=IP2;
                           }
                    }

                } catch (SQLException ex) {

              System.out.println(ex);
                     System.out.println("MySQL ErroR");
       }
```

Fig-5.3 La recherche sur la BD MPNS sur une classe de

3.3 Les Place de marché (MP):

Une MP est composée de :

1. Un firewall MPSM pur filtrage d'agent, c'est un agent service exécuté sur la machine principale du MP.
2. Un Agent MPM (MP manager) gestionnaire du MP.
3. Le service MPDS (Market Place Domain Service).

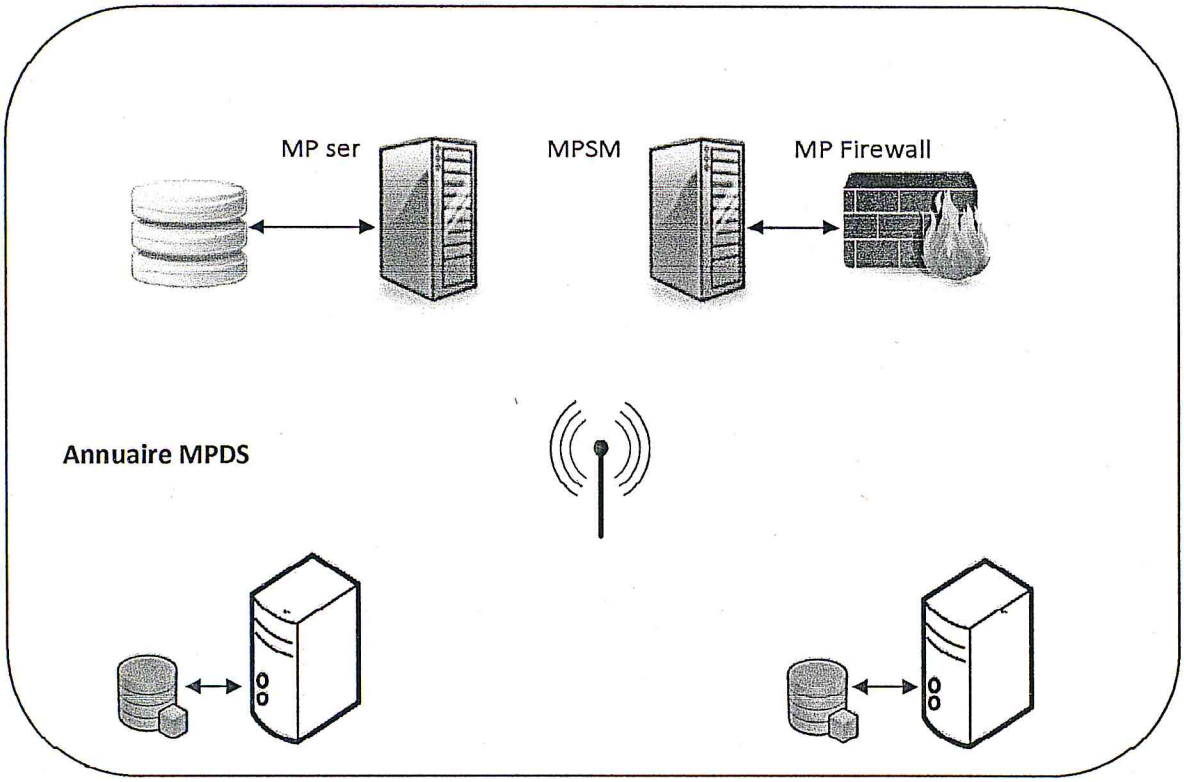


Fig-5.4 Schéma d'une vue matériel d'une MP

Comme une MP est une plateforme JADE aussi, elle est lancée comme suit :

```
if (jTextField_IP_MP.getText().equals("localhost")) {
            try {
                Runtime.getRuntime().exec("cmd /c start java -classpath "
+ classpathValue +
                        " jade.Boot -gui "+ mobilityservice +"
MPM:TestPck.Agent_MPM;MP_CleanUp:desktopapplication1.cleanUpMP;MP_Logger:d
esktopapplication1.MPLoggerAgent("+ jTextField_SvcClass_MP1.getText()
+","+ jTextField_AP_IP_MP1.getText() +")");
                Runtime.getRuntime().exec("cmd /c start " + jadeURL +
"\\MySQL\\restorMP_DB.bat");
            } catch (IOException ex) {

Logger.getLogger(DesktopApplication1View.class.getName()).log(Level.SEVERE
, null, ex);
            }
        }
```

3.4 L'E-Shop :

Nous avons déjà dit que l'e-Shop est un simple conteneur donc il se lance normalement avec :

- la méthode createAgentContainer(), comme un e-Shop peut être distant i.e. pas sur le même hôte que le conteneur principale (Main-container). Nous devons specifier dans son profile l'hôte où le Main-container est lancé, pour qu'il s'inscrive auprès de ce dernier.

- ou avec la commande java jade.Boot –container -host «hôte-Main-Container»

Voici le code de l'application qui lance l'e-Shop

```
try {
            // TODO add your handling code here:
      String eShopName = jTextField_EShop_Name.getText();
      String eShopIP = jTextField_EShop_IP.getText();
      String eShopService = jTextField_EShop_ServiceName.getText();
      Runtime.getRuntime().exec("cmd /c start java -classpath " +
                  classpathValue  +
                  " jade.Boot -container -container-name " +
                  eShopName + " -host " +
                  eShopIP + " AF_" + eShopName + AF_AgentClass +
                  ";R_"+ eShopName + R_AgentClass + "("+
                  eShopService + "," + eShopIP +")");
      Runtime.getRuntime().exec("cmd /c start " + jadeURL +
\\MySQL\\restorEShop_DB.bat");
        } catch (IOException ex) {

Logger.getLogger(DesktopApplication1View.class.getName()).log(Level.SEVER
E, null, ex);
        }
```

3.5 Les agents du système

L'implémentation des agents était la tâche la plus difficile, car les agents par leur nature fonctionnent d'une manière indépendante et travaillent en parallèle avec les autres agents. Et comme JADE n'alloue qu'un thread par agent, alors nous avons dû choisir intelligemment la manière de passer d'un Behaviour à un autre en exploitant au maximum la puissance des différents types de Behaviours fournis par JADE. Car chaque Behaviour ne peut pas être interrompu pour exécuter un autre Behaviour.

Donc tous les agents sont des classes java qui héritent de la classe Agent du package ***jade.core.Agent***

Comme on a dit précédemment les actions des agents ou comportements sont implémentés à travers des ***Behaviours***. JADE offre une liste de Behaviours prédéfinis, on trouve -entre autres- les ***OneShotBehaviour***, ***CyclicBehaviour*** et ***FSMBehaviour***. Ces derniers seront beaucoup rencontrés dans l'implémentation de nos agents. D'où l nécessité de les introduire.

OneShotBehabviour : Un one-shot Behaviour est une instance de la classe **jade.core.behaviours.OneShotBehaviour**. Il a la particularité d'exécuter sa tâche une et une seule fois puis il se termine. La classe OneShotBehaviour implémente la méthode **done()** et elle retourne toujours **true**.

CyclicBehaviour : Un cyclic Behaviour est une instance de la classe **jade.core.behaviours.CyclicBehaviour**. Comme son nom l'indique un cyclic Behaviour exécute sa tâche d'une manière répétitive. La classe CyclicBehaviour implémente la méthode **done()** qui retourne toujours **false**.

FSMBehaviour : FSMBehaviour est une sorte de Behaviour qui implémente un automate à états finis dont chaque état correspond à l'exécution d'un sous-Behaviour. L'introduction de l'automate se fait de la manière suivante :

- L'ajout d'un nouveau état se fait par la méthode registerState (Behaviour state, String name) ;

- L'ajout de l'état initial (il n'existe qu'un seule état initial) se fait par la méthode registerFirstState (Behaviour state, String name);
- L'ajout d'un état final (il est possible d'en avoir plusieurs) se fait par le méthode registerLastState (Behaviour state, String name).

a. State : le Behaviour qui représente l'état;
b. name :le nom de l'état.

- L'ajout d'une nouvelle transition se fait par la méthode registerTransition(String s1, String s2, int event);
- L'ajout d'une transition par défaut (la seule transition entre deux états ou bien la transition à prendre si aucune autre n'est prise) se fait par la méthode registerDefaultTransition(String s1, String s2, String[] toBeReset)

a. s1 : l'état source;
b. s2 : l'état destination;
c. event: l'étiquette de la transition;
d. String[] toBeReset : l'ensemble d'état pour lesquels on doit faire un reset() avant de les ré-exécuter parcequ'on peut pas re-éxecuter un Behaviour une autre fois avant de lui faire un reset().

Nous allons voir dans ce qui suit comment sont réalisés ces différents agents selon l'*état actuel* d'avancement de la réalisation.

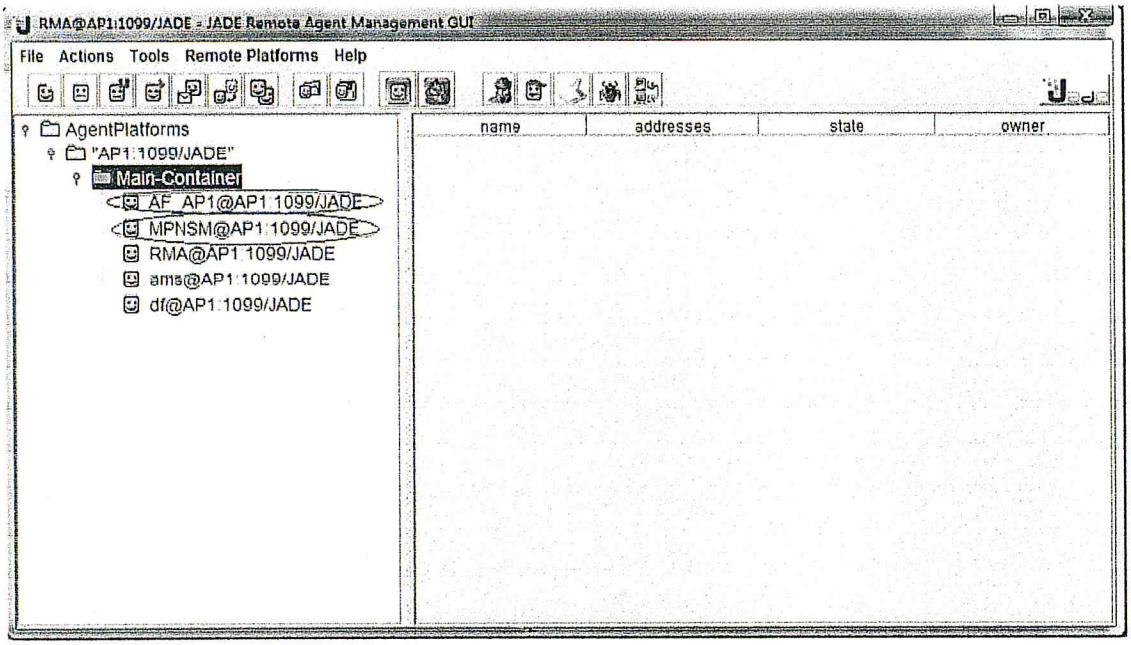


Fig-5.5 Les agents facilitateurs d'un AP

La figure suivante montre les agents MPM, MPSM et MPDSM lancés dans la plateforme JADE de la place de marché MP1.

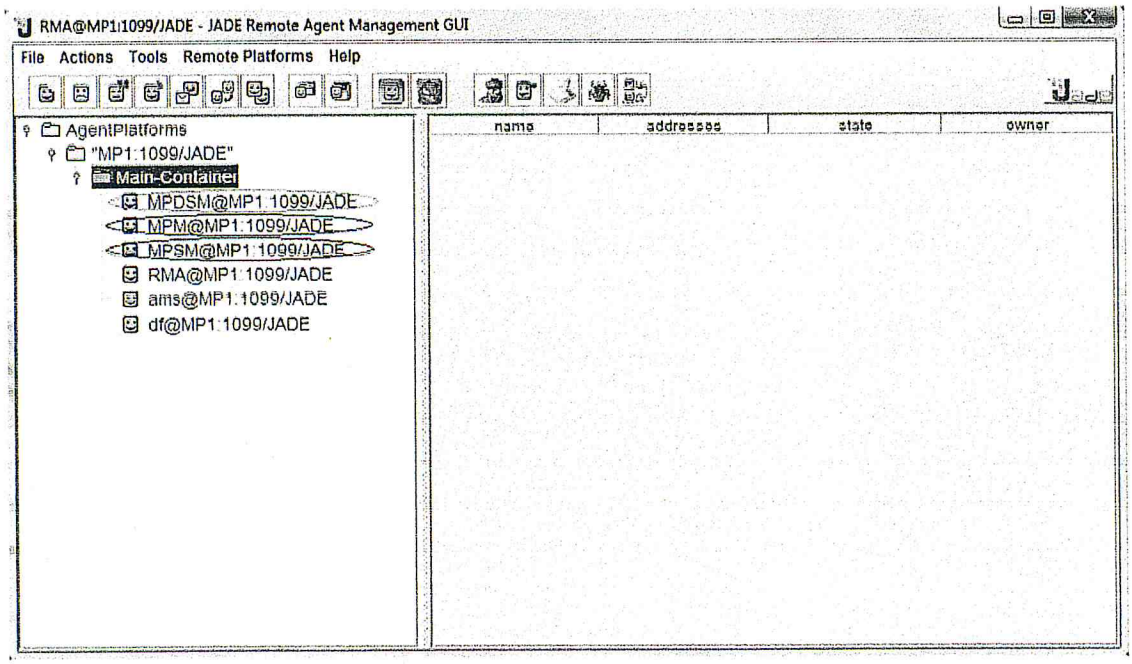


Fig-5.6 Les agents facilitateurs d'un MP

Au niveau de l'E-Shop on trouve les agents suivants :

La figure suivante montre l'agent AF_E-Shop1 lancé dans le container E-shop1.

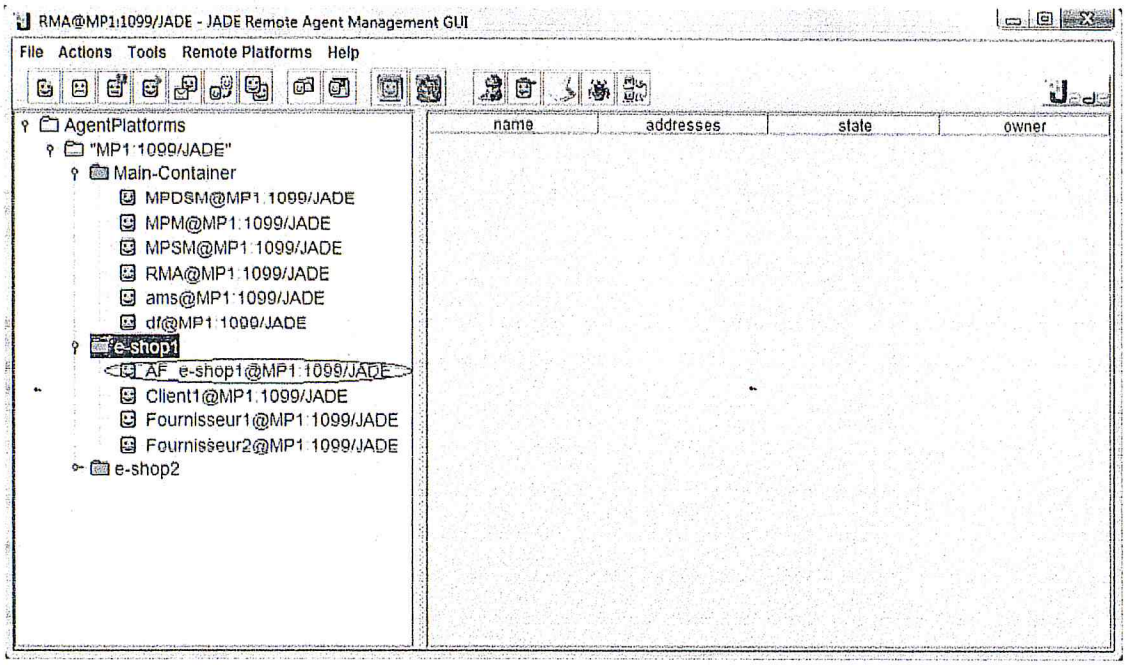


Fig-5.7 Les agents facilitateurs d'un E-shop

Chapitre 6 :
L'évaluation de la sécurité

Introduction :

Dans ce dernier chapitre nous allons faire une évaluation de la sécurité de notre plateforme multi-agent. Cette évaluation doit se faire en fonction de critères bien précis et bien définies.

La sécurité de notre plateforme tente de maintenir les caractéristiques principales suivantes :

La confidentialité

Assurer la confidentialité des données, consiste à faire en sorte que les informations restent secrètes et que seules les personnes autorisées y aient accès.

L'authentification

Assurer l'authentification dans le système, consiste à prouver que l'identité des différentes entités doit pouvoir être vérifiée, et que les informations reçues sont conformes à celles fournies, il faut ensuite assurer que les deux entités communicantes sont bien ce qu'elles affirment être. De plus, le service d'authentification doit montrer que la connexion ne peut être brouillée par une troisième entité essayant de se faire passer pour un des deux correspondants.

L'intégrité

L'intégrité permet d'éviter la corruption, l'altération et la destruction des données dans le réseau de manière non autorisée, elle reste un domaine très large couvrant à la fois les modifications, les moyens de modification mais également la cohérence des données après la modification.

La disponibilité

Consiste à assurer une disponibilité continue des informations, des services et des ressources dans le réseau (les temps de réponse, la tolérance aux fautes, le contrôle de concurrence, le partage équitable de ressources, etc.). D'une autre façon, c'est prévenir contre les perturbations et les interruptions de son fonctionnement et aussi contre toute utilisation abusive des services et des ressources du système.

La non répudiation

Consiste à assurer que lorsqu'une information est transmise entre deux entités, l'émetteur ne doit pas pouvoir nier avoir envoyé l'information, et le destinataire l'avoir reçue. Chaque partie possède ainsi la preuve de l'existence de la transaction.

Le contrôle d'accès

L'accès à certaines ressources (informations ou services) doit être restreint à des entités autorisées.

Lorsqu'une entité souhaite effectuer une opération sur un objet, le système transforme l'opération en une requête qu'il fait passer au moniteur de référence. Ce dernier est responsable du contrôle d'accès aux ressources. Si l'entité est autorisée à accéder à l'objet et à réaliser le type d'opération demandé alors l'accès à l'objet est accordé et l'opération peut se dérouler sans aucune ambiguïté. Dans le cas contraire, il retournera un refus catégorique à l'utilisateur. Les buts du contrôle d'accès rejoignent ceux de la disponibilité.

Le secret du flux

L'intérêt est d'empêcher tout utilisateur non autorisé d'avoir la possibilité d'analyser les flux des données à travers le réseau. Tout accès illégal, même en lecture, à un flux de données permet à l'utilisateur de déduire des informations utiles et qui peuvent, ultérieurement, servir ses intentions malveillantes. La taille des messages échangés, leurs sources et leurs destinations, ainsi que la fréquence des communications entre les utilisateurs sont des exemples de données à préserver pour prévenir le secret des flux dans le réseau et le rendre plus sûr.

Le confinement

Le principe de confinement est complémentaire à la confidentialité, s'inscrivant dans le même but du bon usage des informations.

Le confinement garantit qu'un sujet n'arrive pas à divulguer volontairement le contenu des objets auxquels il a accès à quelqu'un qui n'a pas le droit d'y accéder.

Dans ce que suit on va présenter quelques attaques et comment on a contré ces attaques lors de développement de notre Framework.

1. **Présentation de quelques Attaques possible sur notre Plateforme :**

Généralement on distingue quatre types de menaces (attaques) :

Les attaques d'un agent contre une plateforme, les attaques de la plateforme contre un agent, les attaques d'un agent contre un autre agent et les attaques d'autres entités contre un agent ou la plateforme.

1.1 Masquerading :

Ou l'usurpation d'identité où un agent malveillant peut lui faire passer pour un autre agent dans la plateforme (Avec la plateforme ou avec un autre agent). Il peut avoir comme but d'obtenir des ressources auxquelles l'attaquant n'anormalement pas accès ou de faire endosser la responsabilité de certaines actions à un autre agent.

Une Plateforme peut elle aussi se fait passer pour une autre plateforme dans le but de tromper l'agent par rapport à sa destination normale et de lui faire quitter son domaine de sécurité. De cette façon la plateforme peut obtenir une information sensible contenue dans l'agent.

Dans notre cas l'authentification à base de certificat rende impossible ce type d'attaque lorsque l'authentification est requise par la politique de sécurité.
Un agent est crypté et seule la plateforme réelle avec la clé correspondante peut lui décrypter.

1.2 Rejeu :

L'attaquant qui a réussi à intercepter des messages les ré-émet dans le but d'obtenir des informations ou de perturber la cible de l'attaque. Nous considérons qu'il s'agit d'une attaque sur L'intégrité des messages. Les accès non autorisés sont limités par l'utilisation des proxys, et puisque le code est singé l'accès aux données est impossible pour des agents et des malveillants.

1.3 Attaque DOS

Dénégation d'un service se produit lorsqu'un agent consomme Beaucoup de ressources. (comme du temps de traitement, de la mémoire ou processeur) Ces attaques peuvent être intentionnelles ou non (erreur de programmation).

1.4 DDOS :

Denis de Service distribué, ici on lance une très grande attaque à partir de plusieurs points différents au même moment, ce qui provoquera un crash de la cible. Les plus grandes attaques DDOS ont été lancées à partir de réseaux zombies. Les participants à cette attaque ne se rendent pas compte qu'ils ont fait partie d'une attaque à grande échelle.

Une plateforme peut aussi invoquer ce type d'attaque. Donc elle refuse d'effectuer les services demandés par l'agent. Ainsi, elle peut ignorer les demandes de service, introduire des délais inacceptables pour des taches critiques, ne pas exécuter le code de l'agent ou bien le terminer sans avertissement. D'autres agents qui attendent la réponse de cet agent seront en interblocage *(deadlock)*.

Après chaque attaque, le Hacker tente d'effacer ses traces. Généralement les traces sont gardées dans notre Base de données, indiquant ainsi toutes les connexions et les usages faits de ces dernières.

1.5 Le vol d'information

Le vol d'information est un danger très important, car la plate forme a accès au code a l'état et aux variables de l'agent mobile, il est donc dangereux de faire stocker des information sensibles dans l'agent. Quand un agent arrive sur une plateforme il faut l'authentifier puis le soumettre à la politique de sécurité le concernant. Cette pratique permet d'empêcher l'accès

d'utilisateurs et de processus non autorisés a des ressources protégées. En cas d'accès non autorisé il y aura un vol d'information.

1.6 La duplication des agents et de données :

Par une plateforme malveillante et impossible car on ne trouve pas deux Agents avec le même AID.

1.7 L'altération :

Un agent se déplace sur des plateformes plusieurs dans des niveaux de sécurité différents. En conséquence il est possible qu'une plateforme modifie le code, l'état ou une variable et cela constitue une altération de l'agent. Dans le cas de transactions financières un agent court des risques chaque fois qu'il se déplace sur une plateforme et il peut devenir impossible de trouver la plateforme responsable si elle n'est pas détectée tout de suite. Et on distingue eux cas :

- Single hop problem : où l'agent fait un saut vers une plateforme depuis sa plateforme mère puis il revient a cette dernière.
- Multi-hop problem : où l'agent visite plusieurs plateformes étrangères.

Bien sur les risques sont plus importants et plus difficiles à gérer dans le dernier cas.

1.8 Les Dégâts :

Des dégâts peuvent survenir lorsqu'un agent détruit ou modifie des ressources ou des services en les reconfigurant ou en les effaçant de la mémoire ou du disque.

Tous les autres agents sur la plateforme qui utiliseront ce service ou cette ressource en subirait les effets.

1.9 L'harcèlement :

C'est lorsqu'un agent mobile ennuie les usagers par des attaque répétées : un exemple est l'affichage d'un message ou des pubs non demandées a l'écran.

1.10 L'ingénierie sociale :

C'est où un agent mobile manipule des utilisateurs ou des hôtes en utilisant la désinformation ou la coercition. Par exemple un agent mobile demande des mots de passe sous une fausse autorité (ex. celle de l'administrateur du système) .

1.11 Les bombe logique :

Une bombe logique est une attaque qui son déclanchement est basé sur un événement externe une date par exemple ou à cause de l'identité du propriétaire d'un agent (attaqué par la plateforme).

2. Les Contremesures utilisées:

La sécurité de la plateforme se base sur deux aspects :

1. Sécurité des Agents.
2. Sécurité des hôtes.

La protection des agents contre une plateforme malveillante fait l'appel à plusieurs techniques comme :

- ✓ L'enregistrement d'itinéraires ;
- ✓ Les traces cryptographiques ;
- ✓ Le calcul de fonctions cryptographiques ;
- ✓ Les boites noires limitées dans le temps ;
- ✓ La génération de clés à partir de l'environnement ;
- ✓ La Protection de l'agent par tolérance aux fautes ;
- ✓ L'encapsulation des résultats partiels ;
- ✓ Le Marquage du code.

Dans notre plateforme on a utilisé quelques techniques pour la protection des agents et des hôtes.

Le problème de la sécurité des hôtes est résolu par l'utilisation de la JVM (Java Virtual Machine) qui utilise des techniques de sécurité du code mobile et du contenu exécutable.

L'utilisation de CA (Certification Authority) joue le rôle d'autorité de confiance de type PKI (Public Key Interface) et les MPSS (MP Security Service) au niveau de la MP qui filtre les agents selon leurs AIDs (rejeté les agents qui sont pas certifiés).

Par contre la sécurité des agents mobiles (Contre les plateformes et les agents) est différentes est plus difficile, l'utilisation des proxys réduits les attaques contre les agents par des plateformes malveillantes et *protège un client totalement i.e. un client est inconnue pour un autre client.*

L'utilisation des réseaux fermés où on doit déléguer un agent pour chaque utilisateur du système, c'est la seul façon qu'une entité peut y accéder ce qui va assurer de ne pas avoir des agents malveillants.

On a aussi proposé l'idée de mettre à jour un journal d'audit avec l'identité de toutes les plateformes déjà visitées par un agent et les actions de l'agent sur ces plateformes. Ce journal est protégé cryptographiquement pour éviter une tentative de modification. Ainsi après la découverte d'une action frauduleuse un agent fraudeur pourra être identifié.

Un agent a aussi des limitations qui vont aider à protéger un hôte. Ces limitations sont choisies en fonction de tâche de l'agent. Il existe deux types de limitations :

- Limitation en temps(TTL) : où un agent mobile n'a le droit de rester dans la couche d'exécution qu'un certain temps. Une fois ce temps est écoulé il est détruit ou revient au site d'origine (son Agent Provider).
- Limitation en destination : le nombre de destination de l'agent mobile est limité i.e. les migrations ne pourront se faire que vers une liste d'hôtes définie dés la création (grâce à notre service MPNS).

L'encapsulation des résultats consiste à détecter les altérations de l'agent en encapsulant les résultats des calculs ou d'un service de l'agent à chaque plateforme visité. La validité de ces résultats se fait sur notre plateforme précisément au site AP (l'Agent Provider). L'encapsulation a plusieurs sous objectifs comme la confidentialité (assurée par la cryptographie). L'intégrité (Assurées par les signatures numériques) et la non-répudiation.

2.1 Détection de la falsification des agents [MEN 04]:

Une des solutions qui peuvent être envisagées pour protéger l'itinéraire d'un agent est de le crypter. Malheureusement, cette solution n'est pas bonne à utiliser de faite qu'on doit procéder à un cryptage et un décryptage au niveau de chaque hôte, ce qui alourdit d'une manière remarquable les performances du système.

Une autre solution plus efficace est de laisser l'itinéraire en claire, en le cryptant par partie de la manière suivante :

A chaque fois que l'agent arrive à un hôte, celui là crypte l'itinéraire par sa propre clé privée et concatène le résultat de cryptage avec l'itinéraire en claire de la manière suivante :
Ainsi, l'itinéraire **I0** va se transformé en l'itinéraire **I1** quand l'agent arrive à l'hôte **E1** de la manière
suivante :
I0→ I1= I0+E11 ou:
- I0 : l'itinéraire d'origine non crypté.
- E11 : le résultat de cryptage de I0 par l'hôte H1.
- L'opération « + » désigne la concaténation.

Et à l'arrivé à l'hôte H2 l'itinéraire se transforme de la manière suivante :
I1 → I2 = I1+E12 = I0+E11+E12.

Donc au retour à l'ASP l'itinéraire sera une concaténation de l'itinéraire d'origine (non crypté) avec l'ensemble des résultats de cryptage de l'itinéraire par les clés privés des différents hôtes visitées par l'agent mobile.

L'ASP va décrypter les itinéraires en utilisant les clés publiques (obtenues auprès de TSA) des hôtes visités.

La falsification de l'itinéraire par un hôte malveillant sera détectée. En plus l'identité de cet hôte sera connue.

Le tableau suivant va montrer les déférents types des attaques (Threat Models) sur la plateforme et l'agent et les contremesures utilisées.

	Contremesures	
Type d'attaque	Agent Malveillant	Plateforme Malveillante
Masquerading	AIDs et les certificats	Cryptages des agents, Certifications des places
DoS	Proxy/ Contrôle du log	
Les Accès non autorisés	Les proxys	Cryptages des agents Les signatures numériques
Harcèlement	Firewalling	Signatures Numérique
Les Attaques composés	Contrôle de niveau d'accès	Les Proxys
Bombe logique	Vérification du code	Signatures numérique
Répudiation	Trace d'exécution	
Altération	Les Proxys	Cryptages /signatures numériques
Duplication de données et les agents.	Les proxys	AIDs et les cryptages des agents
Eavesdropping (l'espionnage)	Cryptages des agents Les proxys	Cryptages des agents et Les proxys
Destructions	Limitation de Gamme/durée de vie	Authentification
Hit & Run	Contrôle du log	

Table 1Tableau-6.1. Différents types d'attaque et leurs contremesures.

Conclusion :

Dans ce chapitre on a pu faire une évaluation de la sécurité de notre plateforme, on a commencé par citer quelques attaques connues sur la plateforme et les agents et comment on a sécurisé ces deux. On a aussi résumé tous dans un tableau.

Cependant, comme nous nous sommes heurtées à des contraintes de temps et de matériels (l'absence d'un simulateur d'attaques compatible sur Windows), nous avons dû opter pour l'évaluation faite.

Conclusion générale

A travers ce projet de fin d'études, des domaines variés ont été traités:
Intelligence Artificielle, programmation distribuée, programmation orientée agent, etc.

Nous nous sommes dans un premier temps consacrées à l'étude de l'évolution du modèle client-serveur et pourquoi il est certainement le modèle le plus utilisé pour la construction d'applications réparties et comment le modèle des agents mobiles a émergé dans ce domaine. Nous avons vu aussi que la technologie « agent mobile » offre des avantages pour la réalisation des tâches complexes et répétitives dans les systèmes dynamiques. Néanmoins, l'utilisation de cette technologie dans des applications réelles est encore loin des attentes des chercheurs et reste, alors limitée à cause des problèmes de sécurité, notamment ce qui concerne la protection de l'agent vis-à-vis de l'hôte et la protection de l'hôte contre les agents. Nous avons aussi présenté quelque Framework et plateformes de développement d'agent mobile existantes et on a vu que la plupart souffrent de manque de sécurité. Donc l'objectif de notre travail fut de concevoir un Framework sécurisé pour simplifier le développement et l'exécution des systèmes multi-agents, et puisque le développement de toute une plateforme prend beaucoup de temps et que nous somme limités par ce dernier nous nous sommes servis d'unes de ces plates-formes comme source la plateforme JADE où nous avons changé son architecture par l'architecture du modèle SB (Seller Buyer 1) qui est plus sécurisé et plus résistante au attaques sur l'agent mobile et la plateforme. Les résultats de nos efforts ont abouti à la première version d'un Environnement de Développement et d'exécution de systèmes Multi-Agents plus sécurisé JADE-MP Cet environnement. Il permet la création d'applications réparties plus sécurisé basé sur des systèmes multi-agents d'une performance meilleure qu'un simple SMA. La suite de notre travaille consiste à améliorer la manière d'hébergement du service. A la fin nous espérons collaborer avec des groupes et qu'ils trouvent un intérêt appréciable en ce modeste travail.

Références bibliographiques

[AMA 03] Stuart, J.Russell, and Peter Norvig. *Artificial intelligence A modern Approach second edition.* Pearson Educational International, 2003.

[ABB, 05] : ABBOUTE Hassane, Plates-formes de développement des systèmes multi-agents : AgentBuilder & ZEUS, Mise à jour : Février 2005. Disponible: http://www.limsi.fr/~jps/enseignement/examsma/2005/1.plateformes_1/agentbuilder_zeus.html

[ALM, 04] : ALMAMOU Desage, SMA: le langage KQML, 2004. Disponible : http://www.limsi.fr/Individu/jps/enseignement/examsma/2004/ALMAMOU_DESAGE/KQML_exam_TDM_01.htm#KQML

[BOU, 07] : BOURDON, François. Systèmes Multi-Agents, Mise à jour : 16 octobre 2007. Disponible: http://www.greyc.unicaen.fr/mad/SupportsCours/

[CKP 09] Maria, Chli, and Philippe de Wilde. *"Convergence and Knowledge Processing in Multi-Agent Systems"*.Springer,2009.

[ENB 06] Weidong ,Kou, and Yelena Yesha. *"Enabling Technologies for Wireless e-Buisness"*. Etats unis , Springer,2006

[FERB 95] Jacques, Ferber. *Les Systèmes Multi Agents : vers une intelligence collective.* InterEditions, 1995.

[GEN 09] A.Genco. *Mobile Agents principles of operation and Applications* Italie : WiTeLibrary, 2009.

[HBMAS 09] Virginia, Dignum. *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models.* Harshey, New York : Information Science Reference, 2009.

[HOL 97] Holger Peine and Torsten Stolpmann.The Architecture of the Ara Platform for Mobile Agents Dept. of Computer Science. University of Kaiserslautern, Germany 1997.

[IGI 07] Hong, Lin. *Architectural Design of Multi-Agent Systems : technology and techniques.* Harshy New York : Information Science Reference, 2007.

[JADE 07] Fabio, Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developping Multi-Agent Systems with Jade.* John Wiley & Sons, Ltd, 2007.

[KOLS 04] Christophe, Kolski, Emmanuelle Grislin-Le Strugeon, and Emmanuel Adam. « Conception des systèmes multi-agents : pistes de réflexion en vue de futures coopérations entre ergonomes et informaticiens » *France.* 2004.

[MAN 06] Omer Manssor Paracha. A security framework for mobile agent systems Dissertation.com. Boca Raton, Florida, USA. 2009

[MEN 04] Djamel Eddine Menacer, Un modele D'architecture à base d'agetns mobiles pour les applications reparties, 2004.

[Mich 04] Michael Sonntag, Rudolf Hörmanseder. Mobile agent security based on payment (Institute for Information Processing and Microprocessor

Technology (FIM), 2004.

[RPC 09] SungJin,Choi,and al. *A Reliable Communication Protocol For Multi-Region Mobile Agent Environments* .IEEE Transactions on parallel and Distrubuted Systems , 2009.

[SIM 03] Thierry, Nabeth, and al. *"Enhancing Knowledge Management Systems with Cognitive Agents (Améliorer les Systèmes de Gestion de la Connaissance avec des Agents Cognitifs) ». (2003)*

[SAM 09] Samuel Pierre. Réseaux et systèmes informatiques mobiles, Fondements, architectures et applications.. Presses inter Polytechnique USA , 2003.

[SEC 06] Lu, Ma, and Jeffrey J. P. Tsai. *Security Modeling and Analysis of Mobile Agent Systems.* Londre : Imperial college press, 2006.

[TOOL 05] Peter, Braun, and Wilhelm Rossak. *Mobile Agents Basic Concepts, Mobility Models, and the Tracy Toolkit.* San Diego. Elsevier Inc. (USA) and dpunkt.verlag (Germany), 2005.

Webographie

[MS1] http://msdn.microsoft.com/en-us/library/ms809340.aspx

[limi] http://www.limsi.fr/~jps/enseignement/examsma/2005/1.plateformes_3/index-Ferguen.html

[Mon] http://www.iro.umontreal.ca/~vaucher/Agents/Jade/primer2.html

[Java1] http://www.java2s.com/Tutorial/Java/0490__Security/UsingCertificatesinJava.htm

[Ava] avaxhome.ws

[Jade1] http://jade.tilab.com/

[AUML] http://www.auml.org/

[Djug] http://djug.developpez.com/jade