REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTRE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOHAMED KHIDER BISKRA

FACULTE DES SCIENCES EXACES, DES SCIENCES DE LA NATURE
ET DE LA VIE
DEPARTEMENT D'INFORMATIQUE

MEMOIRE

Présenté pour l'obtentiondu diplôme de
MAGISTER

Spécialité : Informatique
Option : Intelligence artificielle et systèmes d'informations avancés

---

Conception d'une plateforme multi agent pour la collecte de données dans une base de données distribuée

---

Présenté par
Mr. BENDAHMANE Tawfik

JURY

Dr Babahenini Mohamed Chaouki	**M.C.A**	**Président**
Pr. Kazar Okba	**Professeur**	**Rapporteur**
Dr Cherif Fodhil	**M.C.A**	**Examinateur**
Dr Khelil Nacer	**M.C.A**	**Examinateur**

Année universitaire 2014/2015


بسم الله الرحمن الرحيم

Dédicaces

A ma chère mère et mon cher père

A ma chère femme

A ma fille et mon fils – Sandra et Ahmed kenzi

A mes frères et ma sœur

A tous mes amis

Remerciements

En tout premier lieu, je remercie Dieu, le tout puissant, qui m'a éclairé le bon chemin et qui m'a aidé à réaliser mon mémoire malgré les obstacles que j'ai vécu ces dernières années.

Je dois tout le soutien moral et encouragement à mon encadreur, **Mr. le professeur Okba KAZAR** qui je remercie beaucoup pour son patience et gentillesse et qui m'a fait profiter de son grande expérience ainsi que ses précieuses remarques qui ont grandement contribué à améliorer la qualité de ce mémoire.

Je tiens également à exprimer ma reconnaissance aux membres de jury :

Monsieur Babahenini Mohamed Chaouki, Maître de conférences A qui me fait l'honneur de présider mon jury.

Monsieur Cherif Fodhil, Maître de conférences A pour l'honneur qu'il me fait en acceptant d'examiner ce travail.

Monsieur Khelil Nacer, Maître de conférences A pour l'honneur qu'il me fait en acceptant de faire partie du jury.

Je voudrais remercier tous mes collègues de la promotion et les enseignants qui ont marqué leurs empreintes dans notre formation.

Merci infiniment à toute ma famille, mes collègues de travail et toute personne qui m'a aidé de près ou de loin.

Table des matières

Résumé

La collecte des données dans une base de données distribuée a été toujours basée sur le modèle client/serveur, mais dernièrement des études se sont dirigées vers l'utilisation de l'agent mobile qui peut physiquement migrer à travers un réseau informatique dans le but d'effectuer des tâches sur des machines différentes, ayant la capacité de leur fournir un support d'exécution.

Ces agents sont considérés comme composants autonomes, une propriété qui leur permet de s'adapter à des environnements dynamiques à l'échelle d'un réseau large. Ils peuvent également échanger des informations entre eux afin de collaborer au sein de leur groupe.

Nous avons proposé un système multi agents pour la collecte des données dans un environnement distribué en faisant appel agents mobiles. Cette approche est utilisée pour répondre aux besoins de la collecte des données. Celle-ci consiste à collecter les données selon les requêtes de l'utilisateur à partir de plusieurs bases de données homogènes dans de différents sites du réseau. Cette démarche s'appuie sur plusieurs types d'agents mobiles. Chaque type d'agent gère un domaine fonctionnel précis.

Une étude de cas concernant la gestion budgétaire d'un ministère et ses annexes en prenant comme exemple le ministère de la jeunesse et des sports. Les résultats de cette approche nous ont permis d'évoquer les limites liées à l'exigence de l'homogénéité des bases de données, ce qui nous amène à de nouvelles perspectives de recherche et à penser à développer notre système pour qu'il soit capable de faire la tâche de collecte des données à partir d'un ensemble de bases de données hétérogènes.

ABSTRACT:

The data collection in a distributed database was always based on the model customer/server, but, recently studies went to the use of the mobile agent who can physically migrate through a computer network with the aim of making tasks on different machines, having the capacity to supply them a support of execution.

These agents are considered as autonomous components, a property which allows them to adapt itself to dynamic environments on the scale of a wide network. They can also exchange information between them to collaborate within their group.

We proposed a multi-agents-system for the data collection in a distributed environment by appealing mobile agents. This approach is used to meet the needs of the data collection. This one consists in collecting the data according to the requests of the user from several homogeneous databases in various sites of the network. This approach leans on several types of mobile agents. Every type of agent manages a precise functional domain.

A case study concerning the budget management of a ministry and its appendices by taking as example the Ministry of Youth and Sports. The results of this approach allowed us to evoke the limits connected to the requirement of the homogeneity of databases, what brings us to new perspectives of research and to think of developing our system so that it is capable of making the task of data collection from a set of heterogeneous databases.

تلخيص

إن جمع البيانات من قواعد المعطيات الموزعة اعتمد دائما على النموذج زبون/موزع، لكن مؤخرا اتجهت بعض الدراسات إلى استعمال العميل المتنقل، و بالفعل فإن العملاء المتنقلون يمكنهم الانتقال فيزيائيا عبر شبكة الحواسيب من أجل إنجاز المهام الموكلة لها في هذه الحواسيب التي لها القدرة على توفير الموارد الخاصة بها للعميل المتنقل لتنفيذ مهامه،

يعتبر هؤلاء العملاء كمكونات مستقلة و هي الخاصية التي تمكنهم من الاندماج في بيئة متغيرة ضمن شبكة معلوماتية واسعة، و تستطيع أيضا تبادل المعلومات بينها من أجل التعاون في المجموعة الواحدة.

لقد اقترحنا خلال عملنا نظاما متعدد العملاء خاص بجمع المعلومات من عدة قواعد معطيات موزعة باستعمال العميل المتنقل ، هذا النموذج يسمح بالإجابة على احتياجات عملية جمع المعلومات، هذه الأخيرة التي تتمثل في جمع المعطيات وفق المطالب المقدمة من المستعمل من عدة قواعد للمعطيات متجانسة و في أماكن مختلفة من الشبكة المعلوماتية، هذه العملية تعتمد على عدة أنواع من العملاء المتنقلين كل نوع منها يقوم بتسيير نشاط معين.

نتائج هذه الدراسة مكنتنا من معرفة الحدود المرتبطة بكون قواعد المعطيات متجانسة الشيء الذي يدفعنا لأفاق جديدة لعملنا و دراستنا و التفكير في تطوير نظام يسمح بجمع المعلومات من مجموعة من قواعد المعطيات الغير متجانسة.

Introduction générale

L'usage des réseaux informatiques par des différentes entreprises qui sont généralement distribuées dans des sites géographiques différents à pousser les recherches à converger vers les méthodes et outils matériels et logiciels permettant le rapprochement du loin c'est-à-dire réaliser des applications réparties qui aide un utilisateur dans un site géographique X,par exemple, à avoir accès aux données se trouvant dans un autre site géographique Y, bien sûr selon les droits d'accès qui lui sont accordés [01].

Le bon fonctionnement de ces applications réparties nécessite la communication et les échanges entre les entités formant ce type d'application. Le modèle "client/serveur" où les échanges se font par des appels distants à travers le réseau est le modèle le plus utilisé. [01]

Dans ce modèle, seul le client représente une application au sens propre du terme et le rôle du serveur est de répondre aux demandes des clients.

Le serveur construit ses réponses indépendamment du client, ainsi une partie des données envoyées peut être sans aucune utilité, ce qui abouti à élever le trafic dans le réseau.

De plus ce modèle exige une connexion permanente entre le client et le serveur, ce qui provoque des pertes d'information dans le cas d'une panne de connexion. Le concept d'agent mobile, par le fait qu'il ne perd pas son code et son état, apparait donc comme une solution facilitant la mise en œuvre d'applications réparties. Ces agents sont des entités qui se déplacent d'une machine à une autre sur le réseau.

Ainsi, en envoyant les agents là où les tâches se font, les messages échangés deviennent locaux et libèrent d'autant la charge du réseau, et en plus, le serveur répondra aux exigences du client et ne lui envoi que les informations utiles [02].

Parmi les applications les plus importantes utilisant des agents mobiles est la recherche d'informations dans des serveurs distribués. Dans ce type d'applications les agents mobiles se déplacent sur différents sites pour rechercher des informations pour leurs clients. De nombreux travaux ont été élaborés afin d'introduire la technologie d'agents mobiles et les concepts liés à cette dernière pour la recherche d'information dans des environnements dynamiques [02].

Comme la collecte des données dans une base de données distribuée est une application répartie, nous voyons que le concept « agent mobile » peut être l'une des solutions pertinente, à travers la solution d'une approche de système multi agents à base d'agents mobiles.

Notre travail est présenté à travers un mémoire constitué de quatre chapitres

- Le premier chapitre est consacré à la présentation du concept agent, en spécifiant sa définition, ses différentes architectures, ainsi que l'aspect structurel et fonctionnel de chaque type, à la présentation du système multi agents en spécifiant sa définition et ses différentes architectures, et le concept d'agent mobile à travers sa définition, ses propriétés et les différents domaines d'application.

- Dans le deuxième chapitre nous étudions les méthodologies à suivre lors du développement d'un SMA, les types et modèles de communication et l'organisation, la coordination et la coopération entre les agents.

- Le troisième chapitre est consacré à la présentation des différentes plateformes existantes à l'image de Jade et AgentBuilder.

- Le quatrième chapitre consiste à présenter notre contribution qui se résume dans la proposition d'une approche de système multi agents pour la collecte des données dans une base de données distribuée en spécifiant l'architecture générale de notre approche ainsi que l'aspect architectural et fonctionnel des différents agents, d'autre part, nous présentons notre étude de cas qui concerne la gestion budgétaire d'un ministère et ses annexes, dans cette étude, nous commençons par la présentation de certaines notions de la plateforme Jini utilisée pour l'implémentation de notre système multi agents, puis nous présentons le fonctionnement du système.

- Enfin, nous terminons par une conclusion générale qui permet d'avoir un bilan de cette recherche et présenter les perspectives envisagées.

1. Agent et système multi agents

1.1. Introduction

Avant toute tentative d'analyse ou d'étude des méthodes et plates-formes pour la réalisation des systèmes multi agents, il est important de donner une définition de l'agent et du SMA, et de déterminer les différentes caractéristiques que devrait posséder ce type de système. En effet, il fallait s'assurer d'avoir un certain consensus au sein de la communauté multi-agents sur la définition et les caractéristiques d'un agent. Ce qui a donné lieu à plusieurs débats et a été une source de division entre les chercheurs. Heureusement, les diverges se sont estompées et la majorité des intervenants du domaine sont maintenant d'accord sur les caractéristiques globales que doivent posséder les agents [03].

1.2. Définition d'un agent

Le dictionnaire de la langue française "Larousse" définit un agent comme "une personne chargée de gérer, d'administrer pour le compte d'autrui ".
Cette définition met l'accent sur deux aspects fondamentaux :

- Un agent fait des choses
- Un agent agit à la demande de quelqu'un ou de quelque chose.

Actuellement, il n'y a pas une définition exacte de l'agent car on trouve une diversité de définition de l'agent [04, 05,06], et celle de J.Ferber [07] est la plus englobante puisqu'elle détermine tous ce qui entre dans le processus de comportement de l'agent :
"On appelle agent une entité physique ou virtuelle,

a. Qui est capable d'agir dans un **environnement**,
b. Qui peut **communiquer** directement avec d'autres agents,
c. Qui est mue par un ensemble de **tendances** (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser),
d. Qui possède des **ressources** propres,
e. Qui est capable de **percevoir** (mais de manière limitée) son environnement,
f. Qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),
g. Qui possède des **compétences** et offres des **services**,
h. Qui peut éventuellement se **reproduire**,
i. Dont le **comportement** tend à satisfaire ses objectifs en tenant compte des ressources et des compétences dont elle dispose et en fonction de sa perception, de ses représentations et des communication qu'elle reçoit".

De cette définition, on voit que les agents sont non seulement capables de raisonner ou de communiquer entre eux, mais capables aussi d'agir dans leur environnement ce qui affecte cette dernière et lui fait subir des changements, qui à son tour affecte les futures décisions des agents

1.3. Architectures des agents

Dans la définition de J.Ferber, on a vu que les agents agissent, communiquent entre eux, possèdent des ressources, perçoivent leur environnement et possèdent des compétences, mais pour arriver à faire toutes ces capacités, l'agent doit avoir une structure ou architecture permettant de définir comment un agent décide l'action à réaliser. Dans cette section, on va voir les trois architectures les plus utilisées, à savoir :

- Architecture BDI "Belief-Desire-Intention" : dans laquelle la décision de l'agent dépend de la manipulation de structures de données représentants les croyances, les désirs et les intentions de l'agent.
- Architectures réactives : dans laquelle la décision dépend de l'interaction directe entre l'agent et son environnement.
- Architecture hybride : dans laquelle la décision dépend de la combinaison entre le comportement proactif de l'agent, dirigé par les buts, et le comportement réactif.

1.3.1. Architecture BDI

Une architecture BDI est conçue en partant du modèle "Croyance-Désir-Intention", en anglais "Belief-Desire-Intention", de la rationalité d'un agent intelligent. On va présenter la signification de ces trois éléments dans un modèle BDI [08].

1.3.1.1. Aspect structurel

L'agent BDI a une représentation explicite de ses croyances, désirs et intentions. On dénote par B l'ensemble des croyances de l'agent, par D l'ensemble de ses désirs, et par I l'ensemble de ses intentions. Les ensembles B, D et I peuvent être représentés au moyen de divers modèles de représentation de connaissances, par exemple en utilisant la logique des prédicats du premier ordre, une logique d'ordre supérieur, le modèle des règles de production, ou bien comme de simples structures de données[08].

L'agent doit produire des plans, notamment une séquence d'actions qu'il va exécuter pour résoudre le problème. La représentation des actions la plus commune consiste à représenter les effets de ces actions sur l'environnement. Par exemple, si dans un état de l'environnement, l'agent déplace un chariot de la pièce X à la pièce Y, la représentation de cette action sera "déplacer-chariot de X à Y" et l'effet sur l'environnement, contiendra notamment le fait que le chariot n'est plus dans la pièce X et qu'il est dans la pièce Y. Le résultat d'une action sera celui envisagé si

l'environnement est déterministe. Les plans conçus par l'agent sont toujours des structures de connaissances ou structures de données qui sont toujours sauvegardés dans une bibliothèque de plans appelée LibP. Cette composante peut être présente ou non dans l'architecture. Si elle existe, le processus de planification de l'agent est simplifié car il peut retrouver des plans adéquats à une certaine situation en parcourant cette bibliothèque de plans.

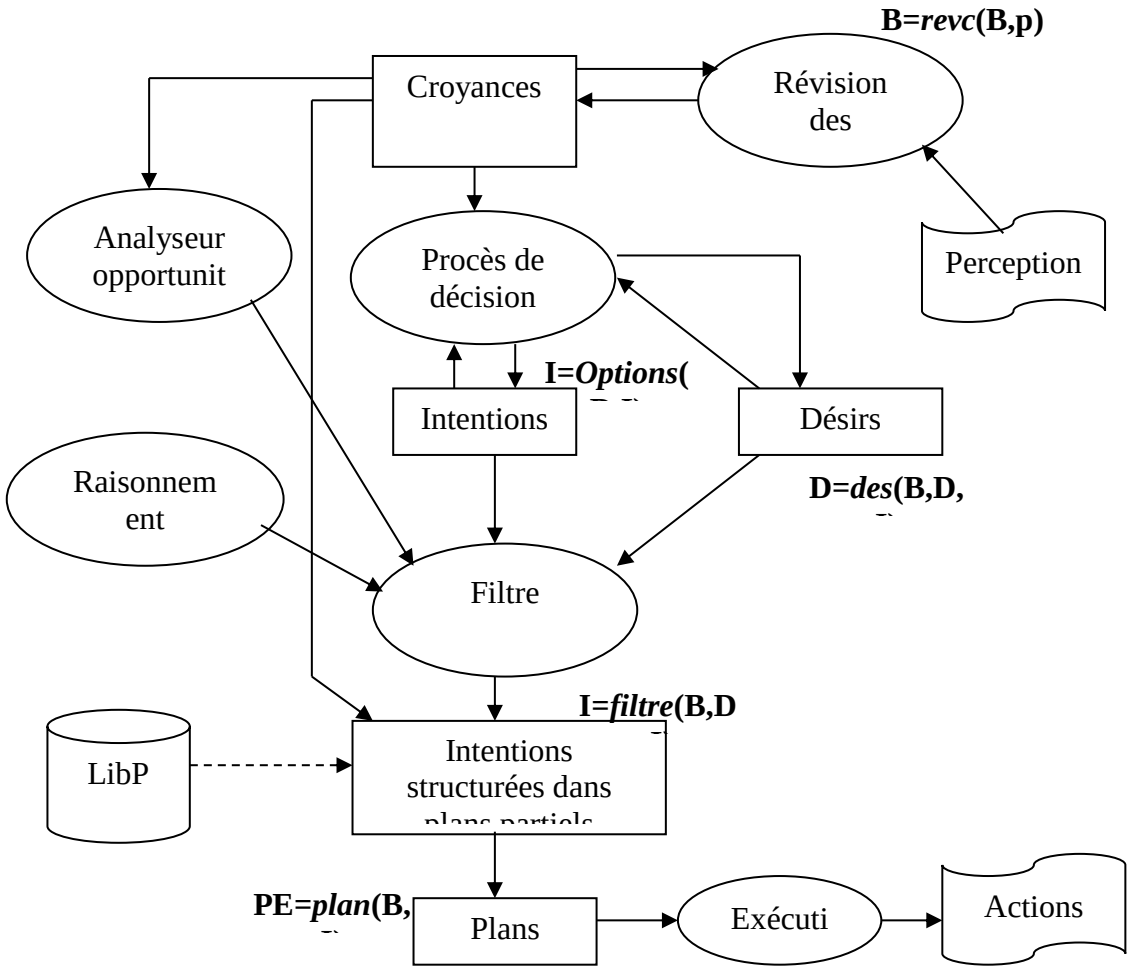


Figure 1. Architecture BDI d'un agent [08]

Dans la figure 01, on remarque que cette architecture est composée d'un ensemble de structures de connaissances ou de données représentées par des formes rectangulaires et un ensemble de composantes de contrôle et d'exécution représentées par des ovales, ainsi que les quatre fonctions nécessaires pour la manipulation des désirs, croyances et intentions de l'agent[08] qui sont:

- **Revc** : est la fonction de révision des croyances de l'agent lorsqu'il reçoit de nouvelles perceptions sur l'environnement, elle est réalisée par la composante Révision des croyances ;

- **Options** : est la fonction qui représente le processus de décision de l'agent prenant en compte ses désirs et ses intentions courantes ; cette fonction est réalisée par la composante Processus de décision ;
- **Des** : est la fonction qui peut changer les désirs d'un agent si ses croyances ou intentions changent, pour maintenir la consistance des désirs de l'agent (on suppose dans notre modèle que l'agent a toujours des désirs consistants) ; cette fonction est également réalisée par la composante Processus de décision ;
- **Filtre** : est la fonction la plus importante car elle décide des intentions à poursuivre ; elle est réalisée par la composante Filtre.

1.3.1.2. Aspect fonctionnel

1.3.1.2.1. Croyance (B = Belief)

Les croyances d'un agent sont les informations que l'agent possède sur l'environnement et sur d'autres agents qui existent dans le même environnement. Les croyances peuvent être incorrectes, incomplètes ou incertaines et, à cause de cela, elles sont différentes des connaissances de l'agent, qui sont des informations toujours vraies. Les croyances peuvent changer au fur et à mesure que l'agent, par sa capacité de perception ou par l'interaction avec d'autres agents, recueille plus d'information.

1.3.1.2.2. Désir (D = Desire)

Les désirs d'un agent représentent les états de l'environnement, et parfois de lui-même, que l'agent aimerait voir réalisés. Un agent peut avoir des désirs contradictoires ; dans ce cas, il doit choisir parmi ses désirs un sous-ensemble qui soit consistant. Ce sous-ensemble consistant de ses désirs est parfois identifié avec les buts de l'agent.

1.3.1.2.3. Intention (I = Intention)

Les intentions d'un agent sont les désirs que l'agent a décidé d'accomplir ou les actions qu'il a décidé de faire pour accomplir ses désirs. Même si tous les désirs d'un agent sont consistants, l'agent peut ne pas être capable d'accomplir tous ses désirs à la fois.

1.3.2. Architecture réactive

Les architectures réactives représentent le fonctionnement de l'agent au moyen de composantes avec une structure de contrôle simple, et sans représentation évoluée des connaissances de l'agent. L'intelligence de l'agent est vue comme étant le résultat des interactions entre ces composantes et l'environnement. Cela veut dire qu'une telle architecture peut résoudre des problèmes complexes, qui normalement demandent un comportement intelligent, sans traiter l'intelligence du point de vue classique de l'intelligence artificielle. On dit que l'intelligence émerge de l'interaction entre des composantes simples, et entre les agents réactifs et l'environnement [09].

L'architecture réactive la plus connue et la plus influente est l'architecture de subsomption. Une architecture de subsomption comporte plusieurs modules, chaque module étant responsable de la réalisation d'une tâche simple. Ces modules correspondent à des comportements spécifiques pour accomplir une tâche particulière, et s'appellent modules de compétence [09].

1.3.2.1. Aspect structurel

Pour la perception de l'environnement, plusieurs modules peuvent assumer l'exécution d'une action différente ; pour choisir l'action la plus opportune, les modules sont organisés en couches hiérarchisées, chaque couche ayant une priorité différente. Les couches supérieures correspondent à des tâches plus abstraites qui sont détaillées à l'aide des tâches plus concrètes et plus simples, les couches supérieures ayant une priorité plus petite que les couches inférieures. Les couches inférieures correspondent aux tâches simples et elles ont une priorité plus grande[09]

Par exemple, si on utilise cette architecture pour construire un robot qui doit faire l'exploration de la planète Mars, on peut avoir l’architecture suivante :

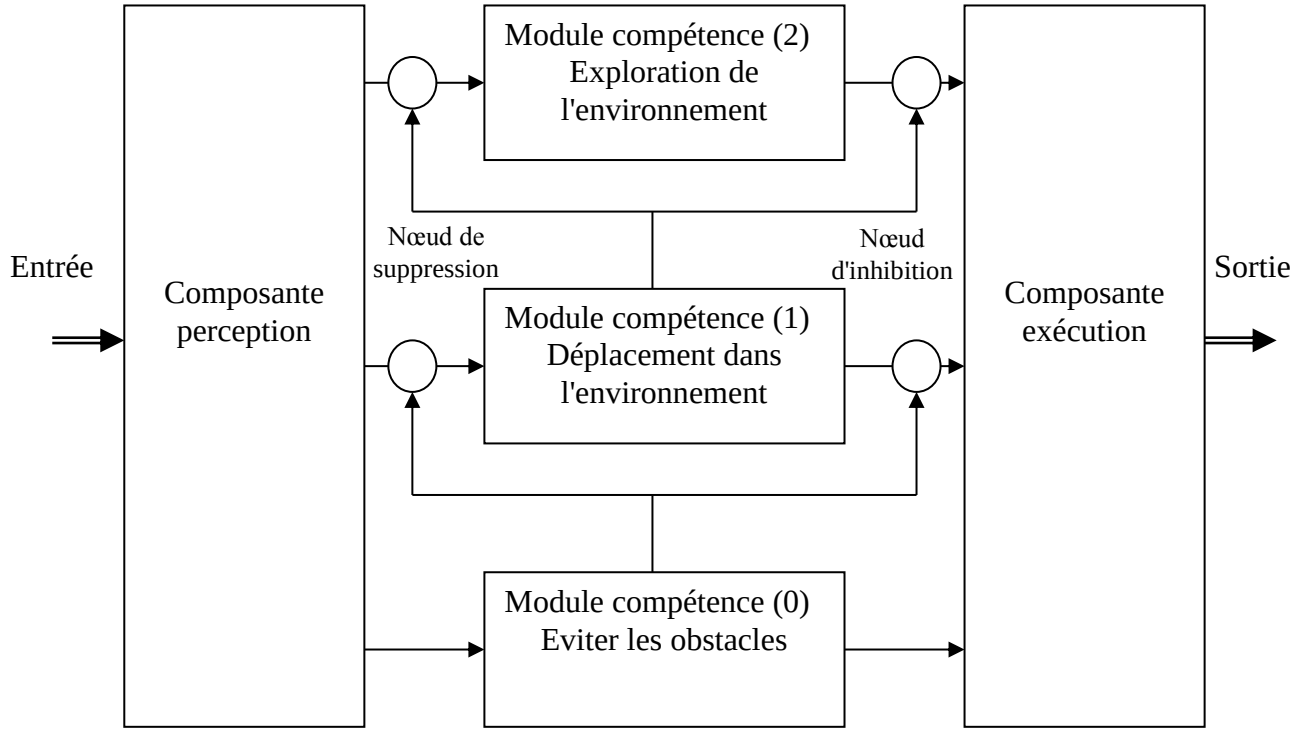


Figure 02. Architecture réactive de subsomption de l'exemple du robot [09]

Cette architecture est composée de trois modules qui sont :

- M0 - un module qui a la compétence d'éviter les obstacles ;
- M1 - un module M1 qui est responsable des déplacements dans l'environnement tout en évitant les obstacles à l'aide de M0 ;
- M2 - un module qui a la compétence supérieure, la plus abstraite, de faire l'exploration systématique de la planète en se déplaçant grâce aux actions du module M1.

Un module sur une couche inférieure a une priorité plus grande qu'un module situé sur une couche plus élevée, parce qu'il est responsable d'une tâche plus simple mais plus "urgente". Dans ce but, le fonctionnement d'un module situé sur une couche supérieure est subordonné à un module inférieur. Un module inférieur peut modifier l'entrée d'un module supérieur au moyen d'un nœud de suppression, et invalider l'action du module supérieur au moyen d'un nœud d'inhibition, comme indiqué dans la **figure 02**. Par exemple, si notre robot veut se déplacer vers l'Est en partant d'une certaine position et qu'il n'y a pas d'obstacle dans cette direction, l'action exécutée par la composante exécution est celle commandée par M1 de se déplacer vers l'Est. Si, par contre, il y a un obstacle, le module M0 prend en compte cet obstacle par sa perception de l'environnement et inhibe le déplacement vers l'Est. M1 essaiera alors de se déplacer dans une autre direction. C'est cette organisation qui justifie l'appellation de subsomption de l'architecture [09].

1.3.2.2. Aspect fonctionnel

Le fonctionnement de l'agent est décrit par un ensemble de règles de comportement, "behaviour rules". Une règle de comportement est semblable à une règle de production et elle a deux parties : Une condition C et une action A.
La condition correspond à une perception de l'environnement, et l'action à une action possible d'un module de compétence.

Soit **Comp = {(c, a) | c dans P, a dans A}** l'ensemble de règles de comportement, où P désigne l'ensemble des perceptions, et A les actions possibles de l'agent.
A chaque module de compétence est associé un sous-ensemble des règles de comportement spécifiques à ses compétences [09].

On peut définir une relation d'ordre ou relation inhibitrice totale D sur l'ensemble Comp, **D : Comp x Comp** qui établit la priorité des modules de compétence. Soit R l'ensemble des règles de comportement qui peuvent être exécutées à un certain moment, notamment les règles (c, a) pour lesquelles la perception p sur l'environnement satisfait la condition c. Alors, les règles qui sont effectivement exécutées sont

{(c, a)| {(c, a) dans R et n'existe pas (c', a') dans R tel que (c', a') < (c, a).

De cette manière, on peut indiquer quel module aura la décision de l'action à exécuter pour une certaine perception sur l'environnement.

1.3.3. Architecture hybride

Une architecture hybride d'un agent est composée d'un ensemble de modules organisés dans une hiérarchie, chaque module étant soit une composante cognitive avec représentation symbolique des connaissances et capacités de raisonnement, soit une composante réactive. De cette manière, le comportement proactif de l'agent, dirigé par les buts, est combiné avec le comportement réactif aux changements de l'environnement [10].

Le système InteRRaP[10]. ("Integration of Reactive Behavior and Rational Planning") est l'une des architectures hybrides les plus connues. InteRRaP est une architecture en couches avec des couches verticales où les données d'entrée, notamment les perceptions, passent d'une couche à l'autre, elle est différente de l'architecture de subsomption qui est une architecture en couches horizontales, où les perceptions sont transmises directement à toutes les couches à la fois.

1.3.3.1. Aspect structurel

L'aspect structurel de cette architecture est présenté dans la figure suivante :

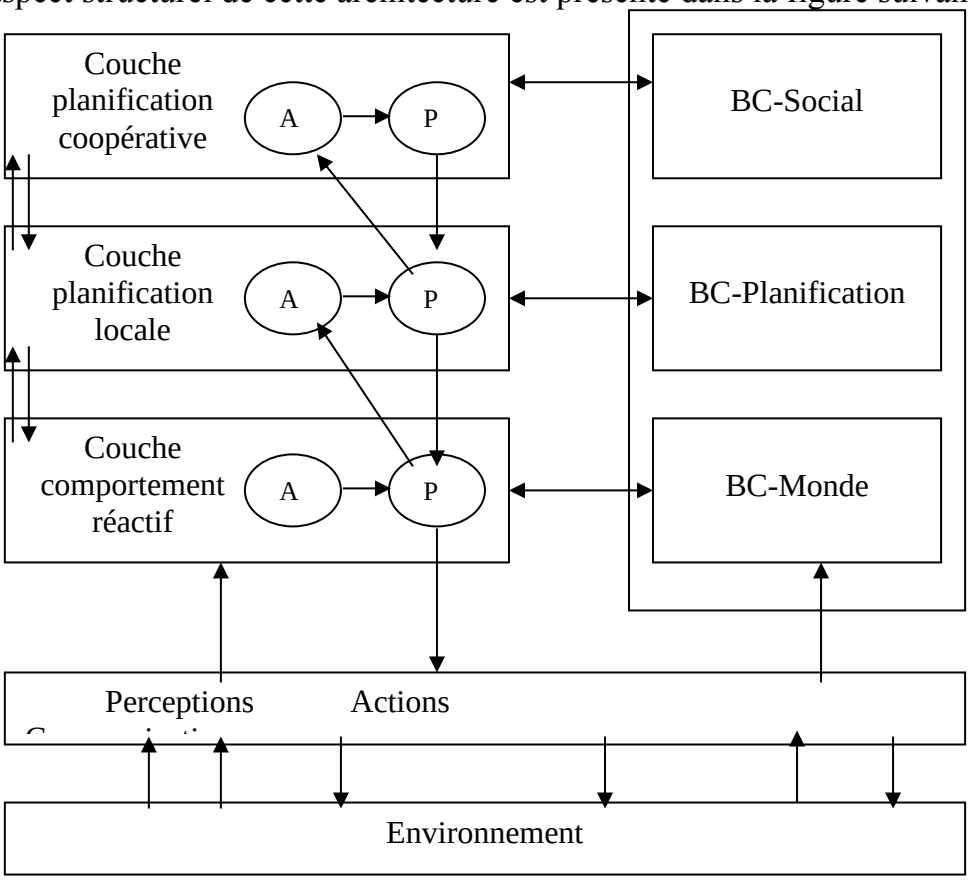


Figure 3. Architecture InteRRaP [10]

L'architecture InteRRaP est composée de trois couches de contrôle et trois bases de connaissances associées qui représentent l'agent et l'environnement à divers niveaux d'abstraction**.** Chaque couche a un ensemble d'opérations spécifiques associées et une couche supérieure utilise les opérations plus simples de la couche d'au-dessus pour exécuter ses opérations plus élaborées. Le flux de contrôle passe de bas en haut, et une couche prend le contrôle lorsque la couche antérieure ne peut plus contribuer, par ses opérations, à l'accomplissement des buts. Chaque couche comprend deux modules: un module pour l'activation des buts et la reconnaissance des situations (AR) et un module de planification et d'exécution (PE). Les perceptions sur l'environnement sont transmises au module AR de la première couche et, de module en module, vers le sommet de la hiérarchie. Le flux de contrôle des actions passe de haut en bas pour arriver à la fin au module PE de la dernière couche, et les actions associées sont exécutées sur l'environnement[10]..

La base de connaissances BC-Monde représente l'information que l'agent possède sur l'environnement (croyances sur l'environnement), la base de connaissances BC-Planification est équivalente à la bibliothèque des plans d'une architecture BDI ; enfin, BC-Sociale représente les croyances de l'agent sur les autres agents du système, et notamment leurs capacités de l'aider à atteindre ses buts.

1.3.3.2. Aspect fonctionnel

Un agent a des buts à atteindre et est capable de coopérer avec d'autres agents pour accomplir ces buts. Les buts d'un agent sont divisés en trois catégories :

Réactions : ce sont des buts simples à accomplir en fonction des perceptions sur l'environnement ;

Buts locaux : ce sont des buts que l'agent peut accomplir par lui-même ;

Buts coopératifs : ce sont les buts qui peuvent être accomplis uniquement par une coopération avec d'autres agents dans le système.

Il est intéressant d'observer que, dans cette conception, la réactivité de l'agent est conçue toujours comme un but, c'est-à-dire au niveau cognitif, mais comme un but très simple à réaliser.

Il faut noter que l'architecture InteRRaP comprend une représentation explicite du processus de coopération d'un agent avec d'autres agents du système.

1.4. Définition d'un système multi agents

On appelle système multi-agent. [07] (ou SMA), un système composé des éléments suivants :

1. Un environnement E, c'est-à-dire un espace disposant généralement d'une métrique.
2. 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.
3. Un ensemble A d'agents, qui sont des objets particuliers (A _ O), lesquels représentent les entités actives du système.
4. Un ensemble de relations R qui unissent des objets (et donc des agents) entre eux.
5. Un ensemble d'opérations Op permettant aux agents de A de percevoir, produire, consommer, transformer et manipuler des objets de O.
6. 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

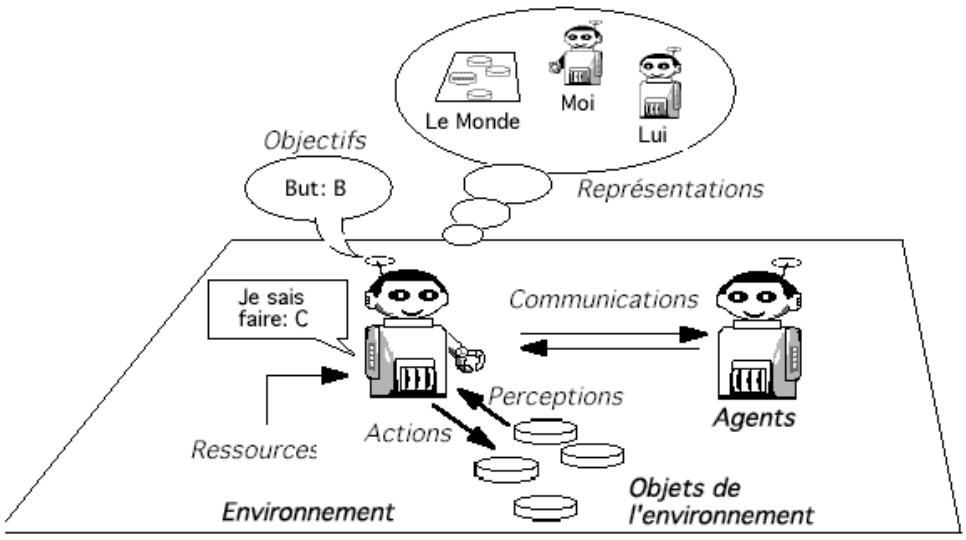


Figure 4 : Représentation d'un agent en interaction avec
son environnement et les autres agents [07]

On peut aussi distinguer les caractéristiques suivantes d'un système multi agents :

- Les agents agissent et travaillent indépendamment les uns des autres.
- Chaque agent est une partie du système.
- Chaque agent a pour but d'accomplir ses tâches.

- Chaque agent communique avec les autres agents lorsque nécessaire.
- Un agent est capable de coordonner ses activités avec les autres agents pour réaliser ses buts.
- Les agents ont un but commun.
- Chaque agent a une vue partielle du SMA.

1.5. Architectures des SMA

Les systèmes multi agents existent sous plusieurs formes à savoir :

1.5.1. Les systèmes centralisés (tableau noir)

L'architecture de tableau noir est l'une des plus utilisée dans les systèmes multi agents cognitifs, elle 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. [07]

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 [07]

• Les sources de connaissance (KS).

• La base partagée (le "tableau" proprement dit) qui comprend toutes les informations que s'échangent les KS.

• Un dispositif de contrôle qui gère les conflits d'accès entre les KS.

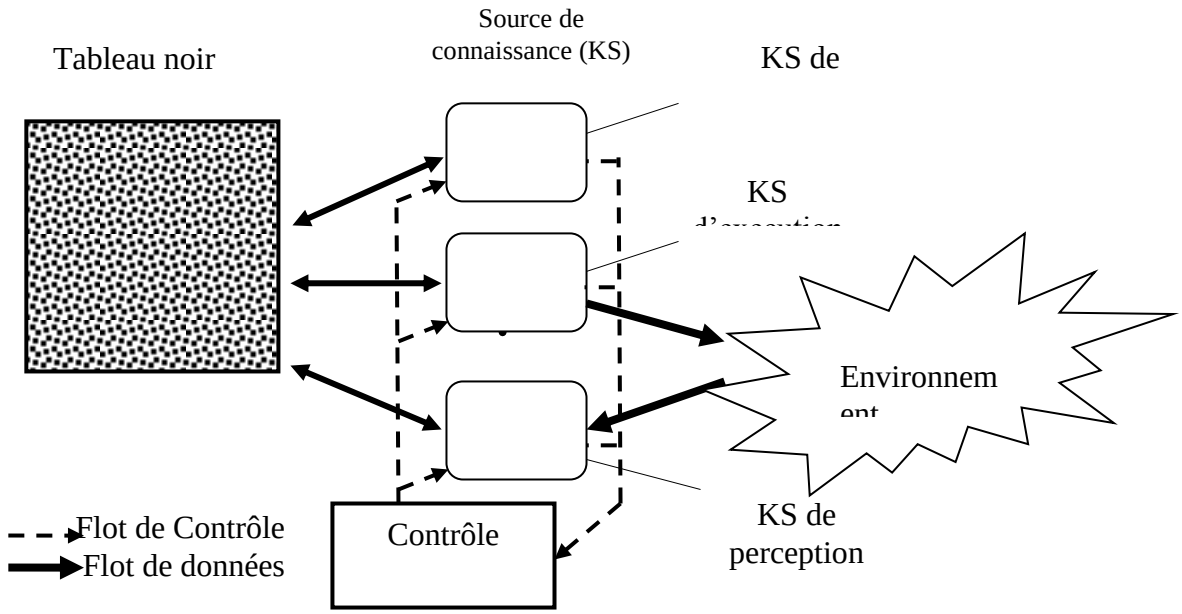


Figure 5: Une architecture de système à base de tableau noir [07]

L'architecture de tableau noir est l'une des plus utilisée dans les systèmes multi agents cognitifs, elle s'est rapidement imposée en IAD comme une architecture souple et puissante pour pouvoir implémenter les mécanismes de raisonnement, de coordination, de communication et d'action[07].

Le modèle de tableau noir est fondé sur un découpage en modules indépendants appelés source de connaissances (KS) qui interagissent indirectement en partageant leurs informations sur un espace de travail (tableau noir) qui contient les états partiels d'un problème en cours de résolution et toutes les informations que s'échangent les KS. Et pour éviter tout conflit d'accès à cette ressource unique, un dispositif de contrôle gère tous les conflits d'accès entre les KS[07].

Si, dans un premier temps, les systèmes à base de tableaux noirs furent considérés comme des systèmes d'IAD, chaque KS pouvant être perçu comme un agent qui interagit avec les autres KS, il n'en est plus de même aujourd'hui. Du fait de leur mécanisme de contrôle très centralisé et de leur manque de mémoire locale et donc de localité des informations, ces systèmes sont maintenant envisagés comme des architectures pratiques pour la réalisation de systèmes "intelligents"[07]

1.5.2. Les systèmes hiérarchiques

Basés sur une structure où les agents répondent à leur supérieur hiérarchique en terme organisationnel, comme par exemple dans une usine chaque groupe de travailleurs est dirigé par un chef, et l'ensemble des chefs de groupe sont dirigés par un superviseur des groupe, et ainsi comme ça jusqu'à un conseil d'administration qui gère un but global de l'usine qui est augmenter la production , d'où on peut énumérer les caractéristiques suivantes [11] :

- Chaque niveau du système est composé d'un réseau dynamique d'agents qui ont le même comportement.
- Chaque agent détaille la méthode de réaction envers une situation locale et les interactions avec les autres agents du même niveau.
- Chaque agent peut considérer les actions d'un sous réseau d'agents comme un niveau inférieur.
- Il n'y a pas d'agent chef qui dirige tous les autres agents : chaque action ou configuration globale est émergente.

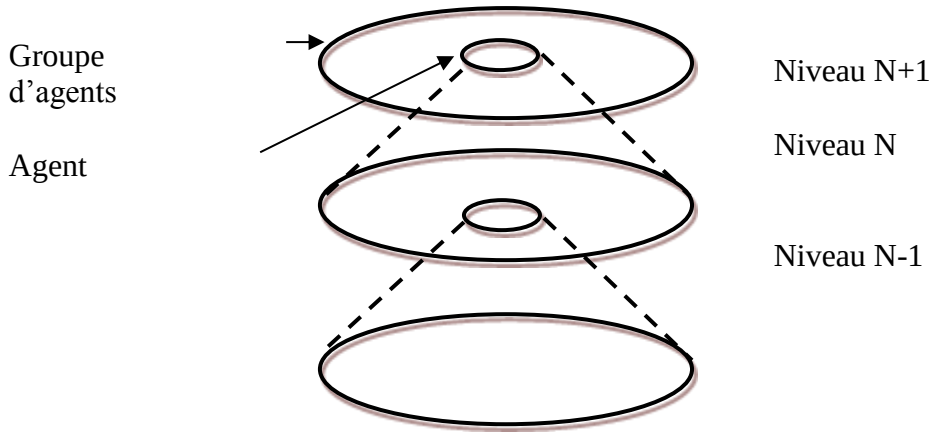


Figure 6: Architecture hiérarchique d'un système multi agents [11]

D'après la figure, le système est divisé en plusieurs niveaux, chaque groupe d'agent d'un niveau inférieur interagissent entre eux dans le même niveau pour aboutir à un but global qui est le but de l'agent chef du niveau supérieur, et ainsi jusqu'à l'arrivée au dernier niveau où le but global du système est émergent.

Le peu de tolérance aux fautes et les limites imposées par les capacités des supérieurs hiérarchiques représentent l'inconvénient de ce type de système.

1.5.3. Les systèmes distribués

La distribution est un atout pour les SMA, elle permet de contrôler la complexité des problèmes à résoudre en décomposant le système en sous-systèmes constitués d'un ou de plusieurs agents et effectuant chacun une partie du travail ce qui conduit à une meilleure adaptation aux changements de l'environnement extérieur [12].

Les raisons qui ont poussé à s'intéresser aux architectures distribuées sont multiples et parmi elles on distingue les nombreuses difficultés liées à l'importance des tailles des données et la multiplicité des sources d'informations pour la conception de bases de connaissances, et la centralisation de ces bases de connaissances génère des difficultés conflictuelles dans le raisonnement. En plus les systèmes centralisés ont une faible résistance aux perturbations externes.

1.6. Les agents mobiles :

Contrairement aux agents stationnaires, nous trouvons des agents dits agents mobiles. Ces derniers, même en dehors de leur environnement, sont capables de communiquer avec d'autres agents et de se déplacer d'un hôte à l'autre. Ils peuvent ainsi exécuter des tâches malgré l'arrêt de leur machine initiale. Ces agents sont souvent qualifiés d'autonomes. Une propriété qui leur permet de se déplacer avec une totale liberté aménageant ainsi leur parcours initial et décidant des tâches à effectuer. Cependant, le paradigme d'agent mobile a soulevé, à ses débuts, plusieurs critiques liées à la sécurité considérée comme un obstacle à sa généralisation. Fort heureusement, les technologies ont évolués entraînant un intérêt pour les chercheurs et industriels au développement d'applications basées sur des agents mobiles [13].

La mobilité est une caractéristique désirable dans les agents pour plusieurs raisons :

- **Efficacité** : si un agent peut se déplacer alors le trafic réseau pourra être réduit, l'agent peut traiter l'information et décider son importance.
- **Persistance** : une fois qu'un agent mobile est mis en place, il n'est pas dépendant du système qui l'a initié et ne doit pas être affecté si celui-ci venait à tomber en panne. La capacité de se déplacer entre les nœuds des réseaux donne à l'agent la capacité de survivre et d'atteindre le plus de ressources possibles.
- **Communication Peer-to-Peer** : un inconvénient majeur du paradigme client/serveur est l'incapacité des serveurs à communiquer entre eux, les agents mobiles sont considérés comme des entités paritaires « peer » et peuvent ainsi adopter la position la plus appropriée à leurs besoins actuels.

- **Tolérance aux fautes** : dans une situation d'échec du serveur ou du réseau pendant une requête, il est difficile pour le client de retrouver la situation initiale afin de se synchroniser de nouveau avec le serveur. Les agents mobiles n'ont pas besoin de maintenir des connexions permanentes étant donné que leurs états d'exécution sont centralisés à l'intérieur d'eux.

Domaines d'application des agents mobiles :

Il existe plusieurs applications pour lesquelles les agents mobiles pourraient être utilisés. La majorité de ces applications s'intéresse à la recherche d'informations au nom de l'usager et possiblement à l'exécution de transactions spécifiques quand les informations appropriées sont rencontrées. Voici une liste d'applications susceptibles d'utiliser le paradigme des agents mobiles.

- **Collecte de données de plusieurs places** : une différence majeure entre le code mobile, comme les applets, et les agents mobiles est l'itinéraire. Alors que le code mobile voyage d'ordinaire d'un point A à un point B, les agents mobiles ont un itinéraire plus complexe et peuvent voyager de manière séquentielle entre plusieurs sites. Une application naturelle par conséquent est la collecte d'informations localisées dans plusieurs ordinateurs d'un réseau.
- **Recherche et filtrage** : étant donné le volume important 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. A la demande de l'usager, un agent mobile pourrait visiter plusieurs sites, chercher à travers les informations disponibles dans chaque site et construire un index des liens pour les portions d'informations qui répondent aux critères de sélection.
- **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

1.7. Conclusion

Dans ce chapitre, on a fait un survol sur les éléments primordiaux dans les systèmes multi agents en spécifiant particulièrement les différentes définitions et architectures des agents à travers l'aspect structurel et fonctionnel de chaque architecture. Les systèmes multi agents sont présentés dans ce chapitre en indiquant la définition et les différentes architectures à savoir l'architecture centralisée (tableau noir), l'architecture hiérarchique et l'architecture distribuée.

Pour qu'un ensemble d'agents puisse former un système multi agents, des méthodologies de développement, des langages de communication, des techniques de coordination et coopération sont nécessaires pour la construction du système. Ces concepts seront détaillés dans le chapitre suivant.

2. Concepts inhérents aux systèmes multi agents

2.1. Introduction

Depuis l'avènement de la notion de SMA, plusieurs concepts ont été développé dans l'objectif de profiter pleinement des systèmes multi agents. Parmi les concepts les plus utilisés dans ce domaine :

- les méthodologies à suivre lors du développement d'un système à savoir la méthodologie *MASE (Multi Agent System Engineering)* [14], *AGR (Agent/Groupe/Rôle)* [15] et *Role Modeling*[16].

- Les types et modèles de communication ainsi que les langages de communication à savoir KQML, FIPA ACL et XML

- L'organisation du système et la coordination et la coopération entre les agents dans les SMA.

2.2. Méthodologies de développement

Depuis quelques années, une multitude de méthodologies pour le développement de SMA ont été proposées et parmi elles :

2.2.1. **MASE (Multi Agent System Engineering) :**

Elle permet une spécification initiale et produit un ensemble de modèles graphiques du système. Son premier objectif est d'aider à analyser, désigner et implémenter un système multi agent [14].

La méthodologie MASE est constituée essentiellement de deux phases (figure 7) : La première est celle de l'analyse réalisée en trois étapes :

1ère étape : Trouver les buts par leur identification à partir des besoins de l'utilisateur et leur structuration en un diagramme hiérarchique du but global aux plus simples.

2ème étape : Appliquer les cas d'utilisation par la création d'un diagramme de séquence qui permet d'identifier un ensemble initial des rôles et les liens de communication entre eux.

3ème étape : Raffiner les rôles par la transformation des buts en un ensemble de rôles et leurs tâches associées.

La deuxième phase représente la phase de désignation, elle contient quatre étapes :

1ère étape : Créer les classes d'agents.

2ème étape : Construire les conversations entre les agents.

3ème étape : Assembler les classes d'agents.

4ème étape : Implémenter le système.

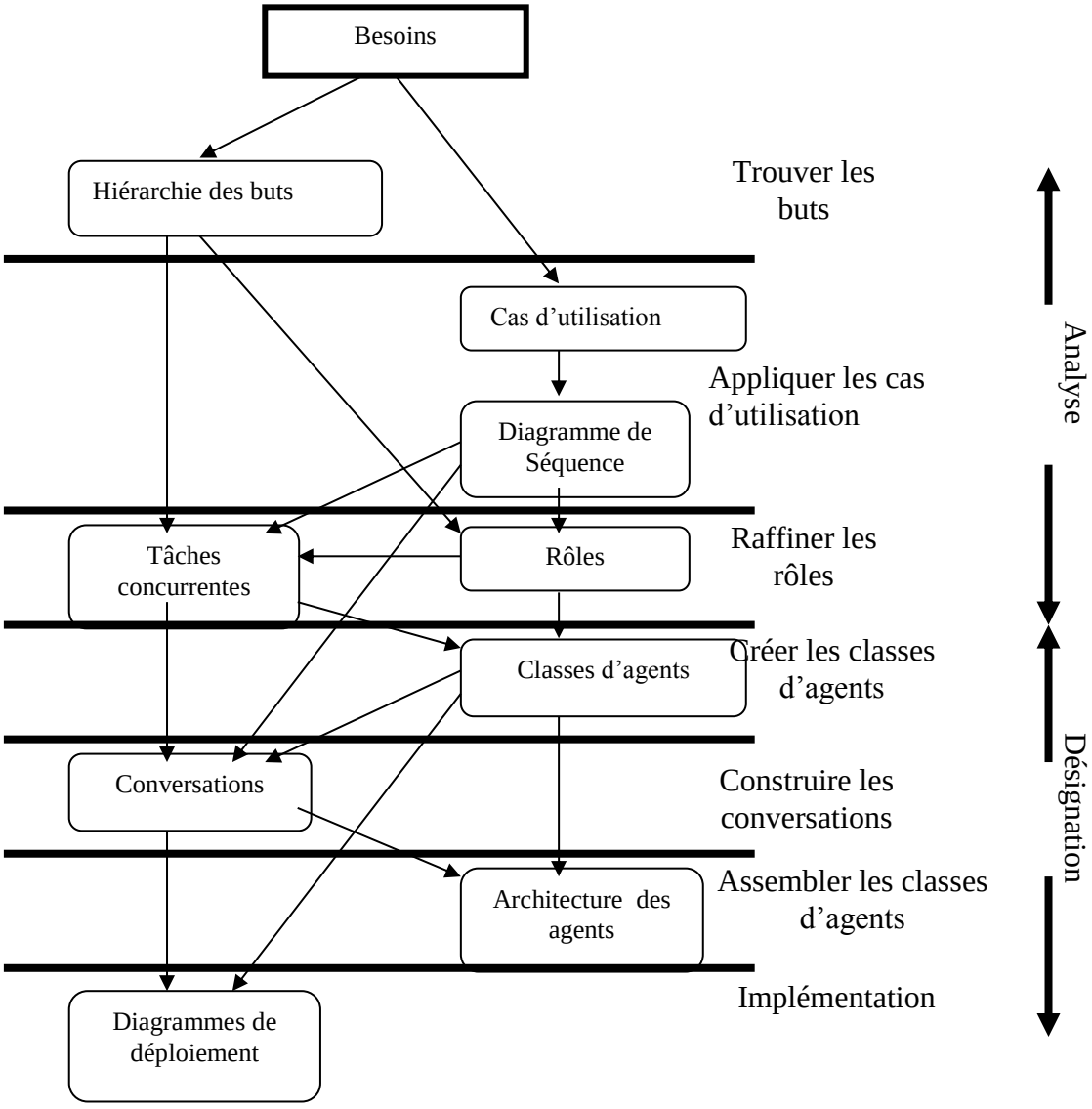


Figure 7 : Méthodologie MaSE [14]

2.2.2. **AGR (Agent/Groupe/Rôle)** :

Aussi appelée Aalaadin, la méthodologie AGR est basée sur un concept simple qui suppose qu'un agent possède un ou plusieurs rôles à l'intérieur d'un ou plusieurs groupes et que chaque groupe contient un ou plusieurs rôles [15] .

2.2.2.1. **Agent**

Le modèle n'impose aucune contrainte sur l'architecture interne des agents. Un agent est désigné comme étant une entité active communicante qui joue des rôles dans des groupes. Cette définition est générale, elle permet aux développeurs de spécifier le modèle le plus adapté aux agents relatif à leur application[15].

2.2.2.2. **Groupe**

Les groupes sont les ensembles atomiques d'agrégation des agents, chaque agent fait partie d'un ou plusieurs groupes. Dans sa plus simple forme, un groupe sert seulement à étiqueter des agents. Dans une forme plus développée et en conjonction avec les rôles, un groupe peut représenter n'importe quel système multi-agents. Les groupes peuvent être en chevauchement, car un agent peut être membre de N groupes en même temps.
La propriété de chevauchement des groupes nous permet de concevoir un monde plat où les agents ne sont pas organisés d'une manière atomique et n'appartiennent pas à des modèles et structures rigides.
Dans chaque groupe, un agent particulier est désigné pour être le représentant de ce groupe lors des contacts avec les autres groupes, il est considéré comme le porte-parole de tous les agents membres de ce groupe [15].

2.2.2.3. **Rôle**

Un rôle est une représentation abstraite de la fonction, du service ou de l'identification d'un agent à l'intérieur d'un groupe. Chaque agent peut jouer plusieurs rôles dans le même groupe, un rôle peut être attribué à un agent à la demande de ce dernier[15].

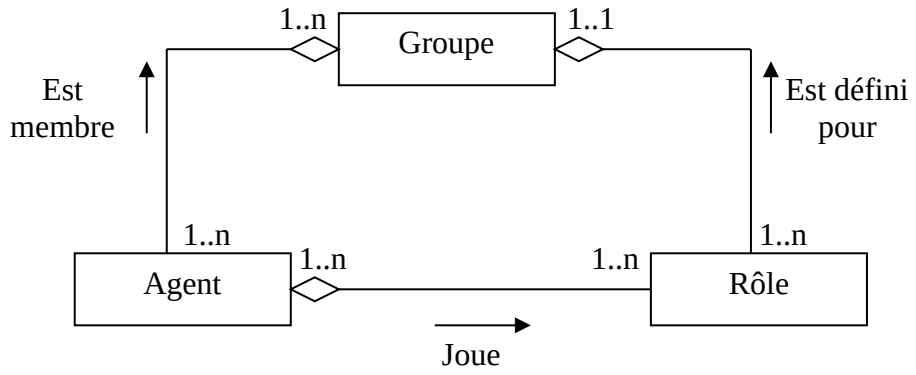


Figure 8 : Méthodologie AGR [15]

2.2.3. **Role Modeling**: Cette méthodologie est constituée de cinq couches dont trois déterminent les composants de l'agent.

La première couche des agents est celle de la définition où l'agent est vu comme une entité autonome capable de raisonner en termes de ses croyances, ses ressources et de ses préférences.

La seconde couche est celle de l'organisation. Dans celle-ci, il faut déterminer les relations entre les agents.

La troisième couche est celle de la coordination où les modes de communication, les protocoles, la coordination et autres mécanismes d'interactions sont déterminés [16] .

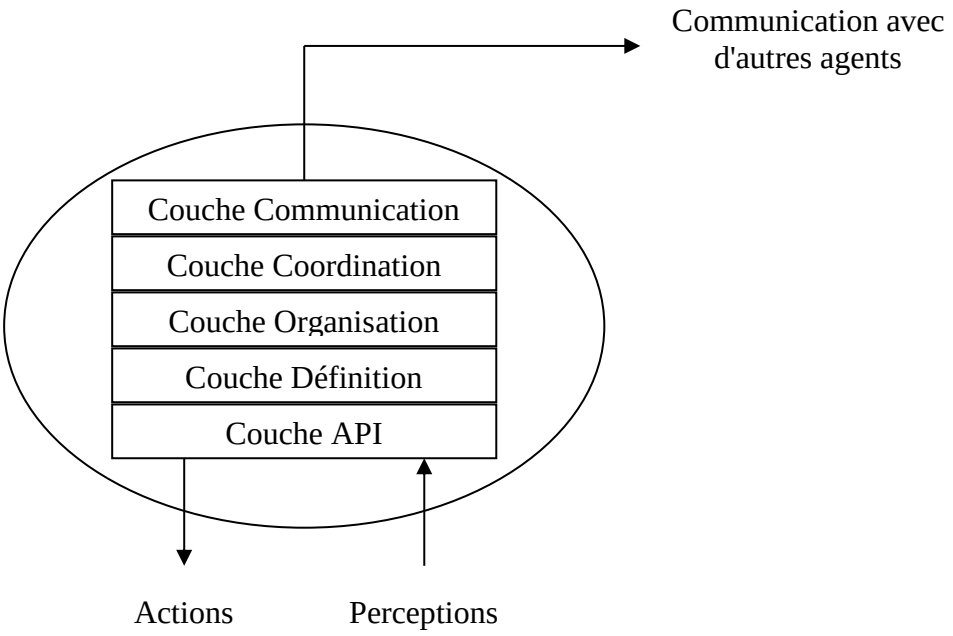


Figure 9 : Méthodologie Role Modeling [15]

2.2.4. **Autres méthodologies :** Plusieurs autres méthodologies et architectures furent développées. En voici quelques-unes : JAFMAS[17], RETSINA[18], dMARS[19], OAA (open agent architecture) [20],Gaia[21], Kaos[22].

2.3. Communication

La communication désigne l'ensemble de processus physiques et psychologiques par lesquels s'effectue l'opération de mise en relation d'un émetteur avec un ou plusieurs récepteurs, dans l'intention d'atteindre certains objectifs [23].

On désigne par le processus physique les mécanismes d'exécution des actions (envoi et réception des messages) le processus psychologique se rapporte aux transformations opérées par les communications sur les buts et les croyances des agents[23].

Les agents communiquent et interagissent entre eux pour synchroniser leurs actions, résoudre les conflits, s'aider mutuellement et succéder aux limites de leurs champs de perception (un agent n'est pas toujours équipé de tous les capteurs nécessaires à la connaissance de son environnement) [23].

Les procédures de communication entre agents sont :

- la communication par partage d'informations
- la communication par envoi de messages.

Ces procédures font appel à des langages de communication à savoir :

- **KIF (Knowledge Interchange Format)** : Les agents ont besoin d'un langage pour communiquer entre eux mais ils ont aussi besoin de comprendre le contenu des messages qu'ils reçoivent. Ceci est possible par l'intermédiaire de KIF [24] qui permet de représenter le contenu des messages par des prédicats et la logique du premier ordre. La description du langage se divise en deux parties : la spécification de la syntaxe (comment se présentera une information en particulier) et la représentation de cette information. Le langage possède aussi des méthodes pour la représentation de la «méta-connaissance » qui peuvent servir de liens entre les bases de connaissances et des langages de représentation des connaissances comme PROLOG et LISP

- **KQML(Knowledge Query Manipulation Language)** : Ce langage de communication est le plus utilisé dans la communication dans les SMA, basé sur la théorie des actes de langages. KQML[25] détermine un format pour les messages et un protocole pour la réception et l'envoi de ces derniers. KQML permet aux agents de partager des informations avec les autres agents du système afin de coopérer pour résoudre un problème.

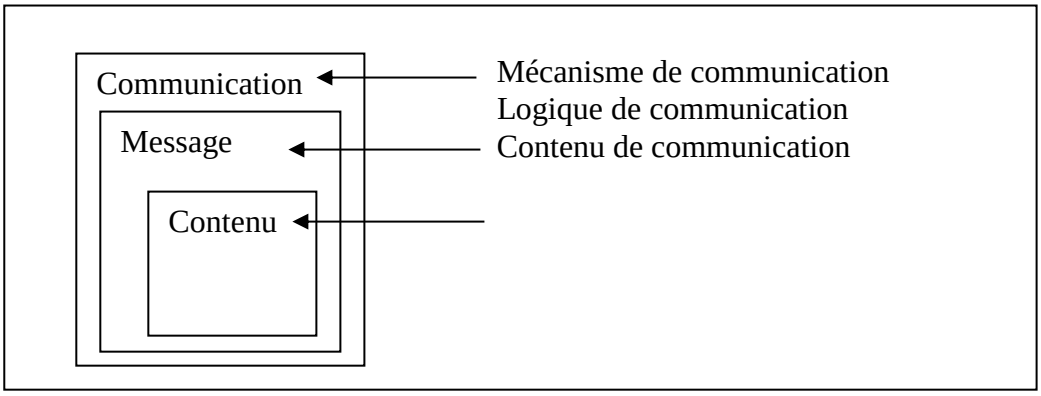


Figure 10 : Les trois couches d'un message KQML

Au niveau conceptuel, le langage KQML se divise en trois couches principales :

- La couche contenue,
- la couche message,
- la couche communication.

La couche contenue est la représentation du message (son implémentation). KQML supporte la représentation des messages de n'importe quel type (chaînes de caractères, messages binaires ou autres). La couche communication encapsule les informations qui décrivent les paramètres nécessaires à la communication comme l'identité du propriétaire du message (l'agent qui envoie) et son GUID (Global Unique Identifier). La couche message détermine le type d'interaction avec l'interlocuteur comme le protocole d'envoi du message, le type de performative et les ontologies utilisées. Le langage fourni au développeur une syntaxe standard pour les messages et plusieurs performatives qui spécifient la ou les forces du message tout en laissant l'utilisateur décider du contenu du message. Parmi les performatives les plus courantes, on retrouve ***tell (Figure 11) ,inform, perform, reply*** et bien d'autres. Chaque performative contient une liste de paramètres qui lui est associée. Le langage se base sur

l'hypothèse que le mode de transport des messages est fiable et préserve l'ordre d'envoi et de réception des messages. Par contre, il n'existe aucune garantie que ces derniers se rendront à destination.

```
Tell
        : content <expression>
        : language <word>
        : ontology <word>
        : in-replay-to <expression>
        : force <word>
        : Sender <word>
        : receiver <word>
```

Figure 11 :Exemple de performative avec KQML

- **FIPA ACL :**
  FIPA (Foundation For Intelligent Physical Agent) [26] est une organisation qui a fixé son rôle principal d'instaurer des standards dans le domaine des SMA. Le langage (FIPA ACL) est un langage semblable à KQML auquel des protocoles d'interaction et autres protocoles populaires ont été ajoutés.

- **XML ACL :**
  XML est un langage très structuré ce qui lui a donné une facilité de lecture et une simple compréhension, mais XML tout seul ne permet pas de comprendre la sémantique des messages ce qui donne l'obligation d'utiliser des outils d'analyse sémantique tel que « Speech Act Theory » ou « Resource Description Framework (RDF) » [27]

2.3.1. Communication par partage d'informations :

Ce modèle de communication est basé sur le partage des informations en utilisant un support centralisé à l'image du tableau noir (méthode décrite au chapitre précédent) où la mémoire partagée est vue comme un tableau sur lequel les agents écrivent leurs messages et trouvent des réponses ou des informations.

2.3.2. Communication par envoi de messages :

Les agents communiquent entre eux par envoi de messages et cela se fait soit d'une manière sélective (un agent connait son interlocuteur) soit d'une manière propagé (un agent envoi un message à tout le monde). Pour communiquer les agents doivent utiliser un protocole qui leur permet de structurer et d'assurer la continuité des communications et des échanges entre un début et une fin de message.

Plusieurs protocoles sont utilisés, les décisions sur la façon dont les agents vont communiquer entre eux sont contraintes par l'organisation choisie [28] :

- Communication sélective ou diffuse : les agents font-ils une distinction entre ceux avec qui ils vont communiquer et les autres? Si oui quels sont les critères pour choisir les destinataires ?

- Communication non sollicitée ou sur demande : sait-on qui veut communiquer avec qui ? la communication est-elle effectuée après demande d'information ou après analyse des besoins des autres agents.

- Communication avec ou sans accusé de réception : le destinataire doit-il ou non indiquer à l'émetteur s'il a reçu l'information ?

- Communication unique ou répétée : une information est-elle envoyée une ou plusieurs fois et à quelle fréquence ?

2.4. Organisation

La notion d'organisation est basée sur un certain consensus entre les différents tenants de théories des organisations et qui permet de caractériser une organisation comme étant :

- des technologies de résolution de problème à grande échelle;
- un ensemble de plusieurs agents (humains, artificiels ou les deux);
- engagés dans une ou plusieurs tâches; les organisations sont des systèmes d'activité;
- dirigés par des buts (cependant, les buts peuvent changer, ne pas être articulables et peuvent ne pas être partagées par tous les membres de l'organisation);
- capables d'agir ou d'être affectés par leur environnement;
- dotés de connaissances, d'une culture, de mémoires, d'une histoire et de capacités distinctes de n'importe quel agent;
- dotés de droits légaux distincts de ceux d'agents individuels.

Une définition plus "informatique" est :

"Les organisations sont des systèmes hétérogènes, complexes, dynamiques Adaptatifs non-linéaires et évolutifs. L'action organisationnelle résulte des Interactions entre des systèmes adaptatifs (à la fois humains et artificiels), de la structuration émergente en réponse à des processus non-linéaires et d'interactions détaillées entres centaines d'acteurs." [28] .

Les organisations d'agents

Dans les déférentes approches de la notion de SMA, les agents possèdent Tous, à des degrés divers, un certain ensemble de connaissances. Il se pose alors la question de savoir quel type de connaissance ils ont à leur disposition et surtout si la connaissance est homogène (c'est-à-dire, d'une certaine façon, du même niveau d'abstraction).

Si la connaissance est hétérogène et fait que les agents impliqués n'échangent pas de message et ne travaillent pas sur la même chose, on peut se demander s'ils ne forment pas en fait plusieurs ensembles d'agents distincts. Dans ce cas on dira qu'ils appartiennent à des organisations d'agents.

Une organisation d'agents est un ensemble d'agents agrèges autour d'un même niveau de granularité (ou encore d'abstraction) de connaissance. Cet ensemble n'est pas fixé une fois pour toute et peut évoluer au cours du temps tant au niveau quantitatif (nombre d'agents par exemple) que qualitatif (changement de granularité ou de nature des messages échangés entre les agents) [28].

2.5. Coordination

La coordination d'actions est l'ensemble des activités supplémentaires qu'il est nécessaire d'accomplir dans un environnement multi-agents et qu'un seul agent poursuivant les mêmes buts n'accomplirait pas[07].

La coordination d'actions est nécessaire pour quatre raisons principales :

- Les agents ont besoin d'informations et de résultats que seuls d'autres agents peuvent fournir. Par exemple, un agent qui construit des murs aura besoin d'être approvisionné en briques, un agent logiciel qui apporte de la valeur ajoutée à un réseau de communication, tel qu'un service d'agence de voyage par exemple, nécessite les services d'autres agents, comme ceux de réservation de chambres et de places d'avions.

- Les ressources sont limitées. On ne fait parfois attention aux actions des autres que parce que les ressources dont on dispose sont réduites et qu'on se retrouve à plusieurs à les utiliser. Qu'il s'agisse de temps, d'espace, d'énergie, d'argent ou d'outils, les actions pour être accomplies ont besoin de ressources qui n'existent pas en quantité infinie. Et la coordination est d'autant plus importante que les ressources sont faibles. Il faut donc partager ces ressources de manière à optimiser les actions à effectuer tout en essayant d'éviter les conflits éventuels.

- On cherche à optimiser les coûts. Coordonner des actions permet aussi de diminuer les coûts en éliminant les actions inutiles et en évitant les redondances d'action.

- On veut permettre à des agents ayant des objectifs distincts mais dépendants les uns des autres de satisfaire ces objectifs et d'accomplir leur travail en tirant éventuellement parti de cette dépendance.

2.6. Interaction et Coopération

Une interaction est une mise en relation dynamique de deux ou plusieurs agents par le biais d'un ensemble d'actions réciproques. Pour un agent « interagir avec un autre » constitue à la fois la source de sa puissance et l'origine de ses problèmes [07]

Les principales situations d'interaction peuvent être classées par rapport à trois critères :

- Les objectifs et intentions des agents : on dit que deux agents sont dans un état de coopération s'ils ont des buts compatibles, c'est-à-dire que l'objectif de l'un dépend de l'objectif de l'autre.

- Relations aux ressources : il se peut que deux agents aient un sous ensemble commun des ressources extérieures utiles à la réalisation de leurs actions, dans ce cas les agents doivent coopérer et partager ces ressources ce qui provoque une situation conflictuelle qui peut être résolue d'une manière ou d'une autre par les techniques de coordination d'action et de négociation.

- Capacités des agents par rapport aux tâches : Est-ce qu'un agent peut réaliser seul une tâche ? bien sûr si la tâche est simple, mais ce n'est pas toujours le cas puisqu'on trouve généralement des tâches complexes à réaliser dans les systèmes.

Un agent peut coopérer suivant les modèles suivants [29] :

- Coopération par partage de tâches et de résultats : Nécessite la décomposition du système initial en un ensemble de sous-problèmes pouvant être traités de façon indépendante et avec un minimum de communication entre agents. Ce type de coopération semble particulièrement adapté pour les domaines où il y a une hiérarchie de tâches, dans le cas où une telle indépendance des sous-problèmes n'existe pas il faut procéder par partage de résultats qui n'exige pas de mécanisme de décomposition

- Commande : Un agent supérieur décompose le problème en sous-problèmes qu'il réparti entre les autres agents ceux-ci le résolvent et renvoient les solutions partielles à l'agent supérieur.

- Appel d'offre : un agent supérieur décompose le problème en sous-problèmes dont il diffuse la liste. Chaque agent qui le souhaite envoi une offre ; l'agent supérieur choisit parmi celles-ci et distribue les sous-problèmes. Le système ensuite travaille en mode commande.

- Compétition : un agent supérieur décompose le problème en sous-problèmes dont il diffuse la liste. Chaque agent résout un ou plusieurs sous-problèmes et envoie les résultats à l'agent supérieur qui fait le tri.

2.7. Environnement

Un environnement est un espace représentant le monde dans lequel les agents évoluent. On fait généralement une distinction entre les agents qui sont les entités actives, et les objets passifs qui se situent dans l'environnement. Lorsque ce dernier dispose d'une métrique (cas général), la capacité de perception d'un agent et sa capacité à reconnaître les objets situés (position, relations entre objets) et la capacité d'action d'un agent et celle de transformer l'état du système en modifiant les positions et relations qui existent entre les objets. Par exemple dans un univers de robots, l'environnement est l'espace géométrique euclidien dans lequel se déplacent les robots (agents) et où sont situés des objets physiques que les robots peuvent manipuler ou doivent éviter[30].

Cependant, Boissier[29] distingue l'environnement du système multi-agents et l'environnement de l'agent. L'environnement du système multi-agents est l'environnement physique support des actions des agents et disposant de ressources. L'environnement de l'agent est l'environnement du système multi-agents et les autres agents. C'est la source des données du problème et le lieu où l'agent peut trouver des ressources.

Russel et Norvig[31] définissent les principales propriétés de l'environnement:

- ·Accessibilité : l'agent dispose de capteurs lui permettant d'accéder à l'état de l'environnement (en général localement).
- ·Déterminisme : le prochain état de l'environnement est fonction de son état courant et de l'action de l'agent.
- ·Episodisme : les prochaines évolutions ne dépendent pas des actions déjà réalisées.
- ·Dynamisme : l'environnement peut changer pendant la réflexion de l'agent.
- ·Discret : le nombre de percepts et d'actions de l'agent est limité.
- ·Concurrence d'accès : d'autres agents souhaitant interagir existent dans l'environnement.

2.8. Conclusion

Dans ce chapitre, on a vu un ensemble de concepts inhérents aux systèmes multi agents, comme les méthodologies de développement, les langages de communication, les techniques de coordination et coopération qui sont nécessaires pour la construction du système.

Dans le chapitre suivant, on va aborder quelques-unes des différentes plates formes pour la réalisation des systèmes multi agents existantes

3. Les plates formes multi agents

3.1. Introduction

Pour permettre le développement de systèmes multi agents, beaucoup d'outils et de plates formes ont récemment vu le jour, les développeurs ont essayé de réaliser des systèmes pour tout type d'agent à travers des plates formes génériques qui peuvent être spécifiées en utilisant les caractéristiques d'un domaine spécifique.

Dans ce chapitre on va présenter un survol concernant quelques plates formes multi agents en spécifiant leurs architectures et caractéristiques

3.2. Jade

JADE (Java Agent DEvelopment Framework) est une plate-forme multi-agents développée en Java par CSELT (Groupe de recherche de Gruppo Telecom, Italie) qui a comme but la construction des systèmes multi-agents et la réalisation d'applications conformes à la norme FIPA (FIPA, 1997). JADE comprend deux composantes de base : une plate-forme agents compatible FIPA et un paquet logiciel pour le développement des agents Java [32].

3.2.1. La norme FIPA pour les systèmes multi-agents

Les premiers documents de spécification de la norme FIPA (FIPA 1997), appelés spécifications FIPA97, établissent les règles normatives qui permettent à une société d'agents d'inter-opérer. Tout d'abord, les documents FIPA décrivent le modèle de référence d'une plate-forme multi-agents (figure 12) où ils identifient les rôles de quelques agents clés nécessaires pour la gestion de la plate-forme, et spécifient le contenu du langage de gestion des agents et l'ontologie du langage [26].

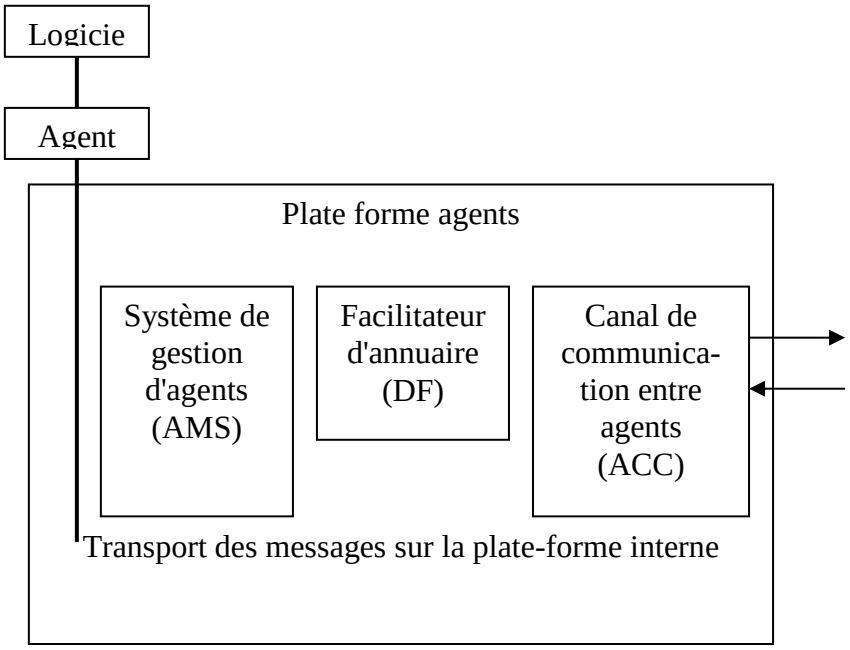


Figure 12 : Le modèle de référence pour une plate-forme multi-agents FIPA[26]

Dans la figure 12, on voit qu'il y a trois rôles principaux dans une plate-forme multi-agents FIPA :

- Le Système de Gestion d'Agents (Agent Management System - AMS) est l'agent qui exerce le contrôle de supervision sur l'accès à et l'usage de la plate-forme ; il est responsable de l'authentification des agents résidents et du contrôle d'enregistrements.
- Le Canal de Communication entre Agents (Agent Communication Channel - ACC) est l'agent qui fournit la route pour les interactions de base entre les agents dans et hors de la plate-forme ; c'est la méthode de communication implicite qui offre un service fiable et précis pour le routage des messages ; il doit aussi être compatible avec le protocole **IIOP** pour l'interopérabilité entre les différentes plates-formes multi-agents.
- Le Facilitateur d'Annuaire (Directory Facilitator - DF) est l'agent qui fournit un service de pages jaunes à la plate-forme multi-agents. Il faut remarquer qu'il n'y a aucune restriction sur la technologie utilisée pour l'implémentation de la plate-forme : émail, basé sur CORBA, applications multi-threads Java, etc.

La communication des agents est basée sur l'envoi de messages. Le langage FIPA ACL est le langage standard des messages et impose le codage, la sémantique et la pragmatique des messages. La norme n'impose pas de mécanisme spécifique pour le transport interne de messages. Plutôt, puisque les agents différents pourraient s'exécuter sur des plates-formes différentes et utiliser technologies différentes d'interconnexion, FIPA spécifie que les messages transportés entre les plates-formes devraient être codés sous forme textuelle. On suppose que l'agent est en mesure de transmettre cette forme textuelle. La norme FIPA préconise des formes communes pour les conversations entre agents par la spécification de protocoles d'interaction, qui incluent des protocoles simples de type requête-réponse, mais aussi des protocoles spécifiques aux agents comme le réseau contractuel et les enchères anglaises et hollandaises.

3.2.2. L'environnement JADE

Le but de JADE est de simplifier le développement des systèmes multi-agents en conformité avec la norme FIPA pour réaliser des systèmes multi-agents inter-opérables. Pour atteindre ce but, JADE offre la liste suivante de caractéristiques au programmeur d'agents[32] :

- La plate-forme multi-agents compatible FIPA, qui inclut
  - le Système de Gestion d'Agents (AMS),
  - le Facilitateur d'Annuaire (DF),
  - le Canal de Communication entre Agents (ACC).

  Ces trois agents sont automatiquement créés et activés quand la plate-forme est activée. La plate-forme d'agents peut être distribuée sur plusieurs hôtes, à condition qu'il n'y ait pas de pare-feu entre ces hôtes.

- Une interface de programmation pour simplifier l'enregistrement de services d'agents avec un ou plusieurs domaines de type DF.

- Le mécanisme de transport et l'interface pour l'envoi et la réception des messages.

- Le protocole IIOP compatible avec le document FIPA97 pour connecter des plates-formes multi-agents différentes.

- Une bibliothèque de protocoles d'interaction compatibles FIPA.

- L'enregistrement automatique d'agents dans le Système de Gestion d'Agents (AMS).

- Un service d'attribution de noms compatible FIPA ; quand on lance la plate-forme, un agent obtient un identificateur unique (Globally Unique Identifier - GUID).

- Une interface graphique utilisateur pour gérer plusieurs agents et plates-formes multi-agents en partant d'un agent unique. L'activité de chaque plate-forme peut être supervisée et enregistrée.

3.2.3. L'architecture de la plate-forme multi-agents

L'architecture du logiciel est basée sur la coexistence de plusieurs Machines Virtuelles (VM) Java et la communication se fait par la méthode RMI (Remote Method Invocation) de Java entre les différentes machines virtuelles (VMs). Chaque VM est un réceptacle d'agents qui fournit un environnement d'exécution complet pour l'exécution des agents et permet d'avoir plusieurs agents qui s'exécutent simultanément sur un même hôte. En principe, l'architecture permet aussi à plusieurs VM d'être exécutées sur le même hôte ; cependant, ceci n'est pas à encourager, car on crée ainsi un surcroît de travail au système qui n'apporte aucun bénéfice. Chaque réceptacle d'agents est un environnement multi- threads d'exécution composé d'un thread d'exécution pour chaque agent et, en plus, des threads créés à l'exécution par le système RMI pour envoyer des messages. Un récipient spécial joue le rôle du frontal de la plate-forme ; il contient les agents de gestion et représente la plate-forme toute entière pour le monde extérieur.

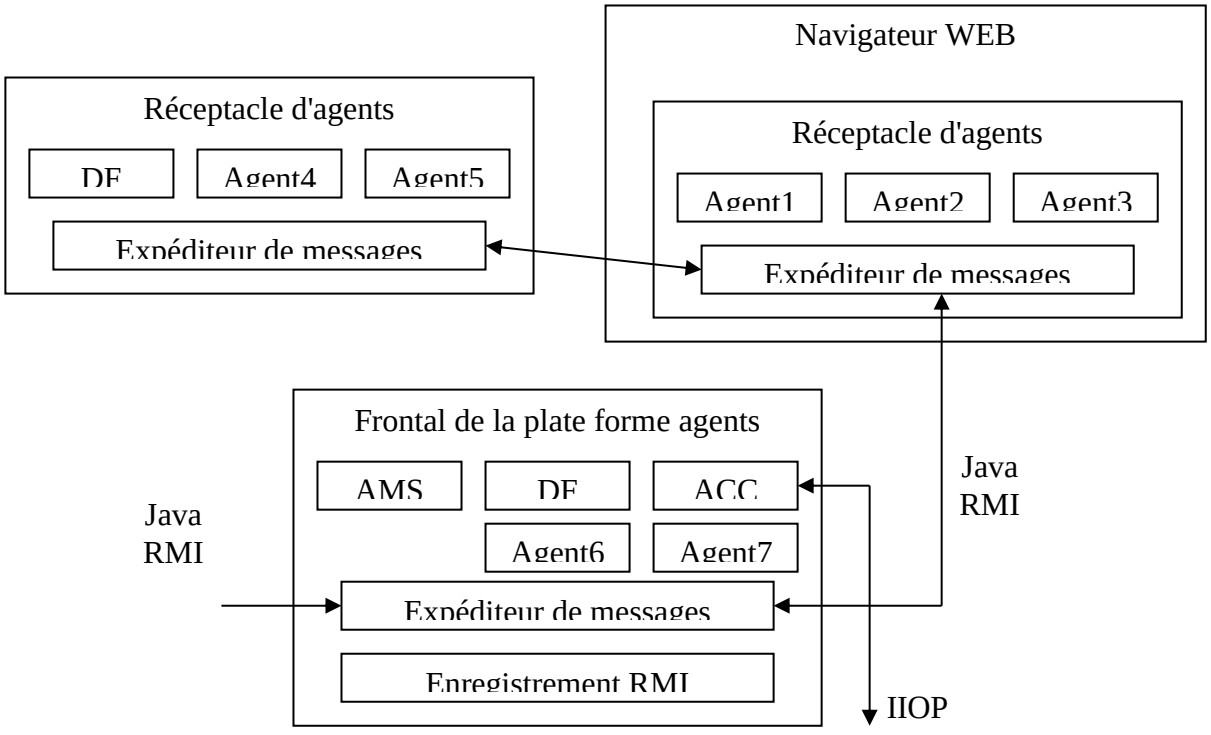


Figure 13 : Architecture logicielle de la plate-forme multi-agents JADE [32]

Une plate-forme multi-agents JADE est alors composée de plusieurs réceptacles d'agents. La distribution de ces réceptacles à travers un réseau d'ordinateurs est permise, à condition que la communication RMI entre leurs hôtes soit conservée. Un réceptacle léger spécial est implémenté pour l'exécution des agents dans un navigateur Web. Chaque réceptacle d'agents est un objet serveur RMI qui gère localement un ensemble d'agents. Il règle le cycle de vie des agents en les créant, les suspendant, les reprenant et les détruisant. En plus, il traite tous les aspects de la communication : répartition des messages ACL reçus, routage des messages selon le champ de destination (: receiver) et dépôt des messages dans les files de messages privées des agents. Pour les messages vers l'extérieur, le réceptacle d'agents maintient assez d'information pour chercher l'emplacement de l'agent récepteur et pour choisir une méthode de transport convenable pour expédier le message ACL.

La plate-forme offre une interface graphique utilisateur (GUI) pour la gestion à distance qui permet de contrôler et superviser les états des agents, par exemple arrêter et remettre en marche un agent. L'interface graphique permet aussi de créer et de commencer l'exécution d'un agent sur un hôte éloigné, à condition qu'un réceptacle d'agents s'exécute déjà sur cet hôte. L'interface elle-même a été implémentée comme un agent, appelé RMA (Remote Monitoring Agent). Toute la communication entre les agents et l'interface (GUI) et toute la communication entre cette interface et l'AMS est faite par ACL via une extension ad-hoc de l'ontologie des agents de gestion FIPA.

3.3. AgentBuilder

AgentBuilder®

AgentBuilder [33] est une suite intégrée d'outils permettant de construire des agents intelligents, elle a été développée en JAVA par *Reticular Systems Inc*, l'élaboration du comportement des agents se fait à partir du modèle BDI et du langage AGENT-0, le langage de communication entre les agents est KQML [34]

AgentBuilder est composé d'une interface graphique et d'un langage orienté agent permettant de définir des croyances, des engagements et des actions. Il permet également de définir des ontologies et des protocoles de communications inter-agents.

Un agent crée avec cet outil est typiquement un agent d'interface, chargé de faciliter la recherche d'information ou la réalisation de certaines tâches à la place de son utilisateur. Un tel agent sera capable de filtrer l'information, de négocier des services avec d'autres agents et dialoguer avec son utilisateur.

AgentBuilder contient deux composants : Le Toolkit et Le système d'exécution (Runtime). Le toolkit contient des outils pour :

- gérer le processus de développement des agents
- analyser le domaine d'influence d'un agent
- concevoir l'environnement de communication sur le réseau
- définir le comportement individuel de chaque agent

- débuguer et tester les agents.

Le runtime fournit l'environnement d'exécution des agents, et la combinaison du programme d'agent et du moteur d'agent permet de créer un agent exécutable [33].
La méthodologie d' AgentBuilder [35] se base sur quatre (04) étapes pour la construction d'agents.

3-3-1-Analyse : L'étape d'analyse consiste en la spécification des objets du domaine et des opérations qu'ils peuvent effectuer, puis en la production d'une ontologie du domaine. Des outils graphiques sont disponibles pour effectuer ces tâches.

3-3-2-Conception : L'étape de conception consiste à :

- Décomposer le problème en fonctionnalités que les agents doivent avoir.
- Définir les agents.
- Identifier leurs rôles et leurs caractéristiques.
- Spécifier les protocoles d'interactions.

3-3-3-Développement : L'étape de développement consiste à :

- Définir le comportement des agents en se basant sur leurs règles de comportements, croyances initiales, intentions et capacités.
- Intégrer les actions externes et l'interface graphique aux agents. Cette étape n'est applicable qu'à des agents adoptant l'architecture BDI.

3-3-4-Déploiement : Les agents engendrés au cours de la phase de développement sont exécutés par un Run-Time Agent Engine, qui interprète le code des agents.

3.4. MadKit

MadKit (Multi-Agents Developpement Kit) est une plateforme multi-agents modulaire et scalable écrite en Java et conçue selon la méthodologie **AGR (Agent/Group/Role)**: des agents sont situés dans les groupes et jouent des rôles. MadKit est développée *au Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM) de l'Université Montpellier II*. Un moteur d'exécution est utilisé dans MadKit où chaque agent est construit en partant d'un micro-noyau. Chaque agent a un rôle et peut appartenir à un groupe. MadKit est doté d'un environnement de développement graphique qui permet facilement la construction des applications [36].

3.4.1. Le modèle Agent/Rôle/Groupe:

L'architecture de la plateforme MadKit a été construite selon le modèle Agent/Rôle/Groupe, développé dans le contexte du projet Aalaadan[36]. MadKit implémente et utilise ce modèle pour ses différentes taches de développement des agents. Dans ce modèle, les concepts organisationnels tels que : les groupes, les rôles, les structures, les dépendances, etc. sont considérés comme les briques de base qui permettront le développement des systèmes scalables et hétérogènes.

3.4.2. Architecture de MadKit:

L'architecture de MadKit est constituée des 3 principaux composants [36]:

3.4.2.1. Le micro-noyau des agents

Le micro-noyau de MadKit est petit (moins de 40 kb) et optimisé, il offre les opérations de base qui permettent le déploiement des agents.

3.4.2.2. Agentification:

La classe de base d'un agent MadKit (AbstractAgent), définit quelques fonctionnalités de base pouvant être nécessaires dans les modèles classiques.

3.4.2.3. Architecture des composants graphiques:

Madkit est doté d'un modèle graphique basé sur des composants graphiques indépendants, utilisant la spécification JAVA Beans dans la version standard. Chaque agent est responsable de ses propres interfaces graphiques.

MadKit est constitué d'un ensemble de paktages de classes JAVA qui implémentent le noyau agents, un environnement de développement graphique ainsi que des modèles standard des agents.

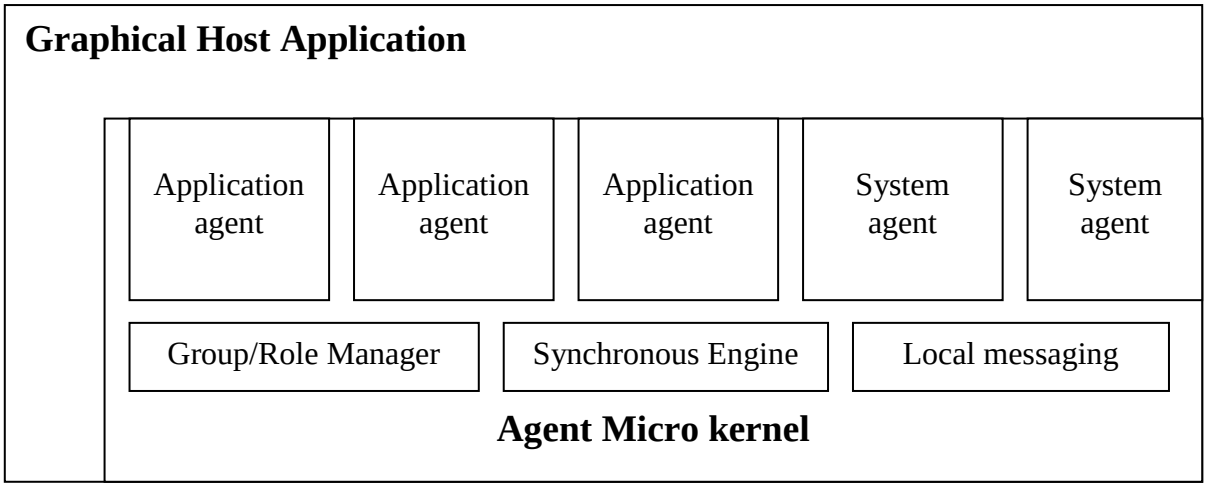


Figure 14 : Architecture de MadKit [36].

La philosophie de MadKit est basée sur l'idée d'utiliser le plus possible la plateforme pour ses propres besoins de gestion. Aucun service autre que ceux offerts par le micro-noyau n'est utilisé par les agents. Les groupes des agents ont été proposés dans d'autres plateformes, mais ils manquent de la capacité de gérer des groupes multiples et des rôles différents pour le même agent au sein d'un groupe.

La méthodologie de MadKit utilise la même décomposition en quatre étapes pour la construction d'un système multi agents[35]

3-4-3-Analyse

Il n'y a pas de méthode d'analyse spécifiquement associée à MadKit. La méthode choisie doit faire une analyse fonctionnelle, une analyse des dépendances, une définition des contextes de groupes et un choix des mécanismes de coordination. Toute méthode d'analyse générique ou spécifique au domaine peut être utilisée.

3-4-4-Conception

MadKit étant basé sur un modèle d'organisation, cette étape comprend la définition du modèle d'organisation (groupes, rôles), le modèle d'interaction (protocoles, messages), et d'autres entités spécifiques (tâches, buts, etc.). Le modèle Aalaadin sert de guide pour la conception, mais aucun outil logiciel n'est fourni. Malgré sa focalisation sur les organisations, le modèle Aalaadin est en fait applicable à toutes sortes de systèmes multi-agents. L'intuitivité du modèle simplifie également sa mise en oeuvre. De plus, les groupes et les rôles définis peuvent être réutilisés par l'intermédiaire de patrons de conception.

3-4-5-Développement

C'est à ce stade que l'on choisit le modèle d'agent et son implémentation. Il n'y a pas de modèle d'agent spécifiquement associé à MadKit, cependant des librairies permettent d'utiliser Lisp ou un moteur Jess pour implémenter des agents. Si les modèles d'agents disponibles ne répondent pas aux besoins, un nouveau modèle devra être implémenté en Java. Cela rend difficile l'utilisation de MadKit pour implémenter des agents cognitifs complexes. Cependant, les nouveaux modèles d'agents développés peuvent être ensuite réutilisés dans d'autres projets.

3-4-6-Déploiement

Le déploiement dans MadKit se déroule dans la G-Box, où les agents peuvent être créés, modifiés et détruits, à l'aide d'une interface graphique. Plusieurs G-Box peuvent être interconnectées par réseau pour distribuer les traitements. Un mode sans interface graphique est également disponible pour simplifier le déploiement. La G-Box permet la configuration dynamique des propriétés éditables des agents, à la manière des composants JavaBeans. Les agents sont encapsulés dans des archives et peuvent être mélangés avec toutes sortes d'autres agents, ce qui permet de réutiliser facilement des agents génériques.

3.5. DIMA

DIMA est un environnement de développement de systèmes multi-agents dont le développement a débuté en 1993, dans le cadre de la thèse de **Z. GUESSOUM** dans le thème Objets et Agents pour Systèmes d'Information et de Simulation (OASIS) du LIP6[37].
DIMA a été utilisé pour développer plusieurs applications réelles (Ventilation artificielle, simulation de modèles économiques,...). Ces applications peuvent être des simulations, des résolutions de problèmes ou des systèmes de contrôles ayant éventuellement des contraintes temps réel. La première version de DIMA a été implémentée en Smalltalk-80 et a été ensuite portée en JAVA. [37]
L'architecture d'agent dans DIMA propose de décomposer chaque agent en différents modules ou composants dont le but d'intégrer des paradigmes existants notamment des paradigmes d'intelligence artificielle. Ces modules représentent les différents comportements d'un agent tels que la perception (interaction agent-environnement), la communication (interaction agent-agent) et la délibération.

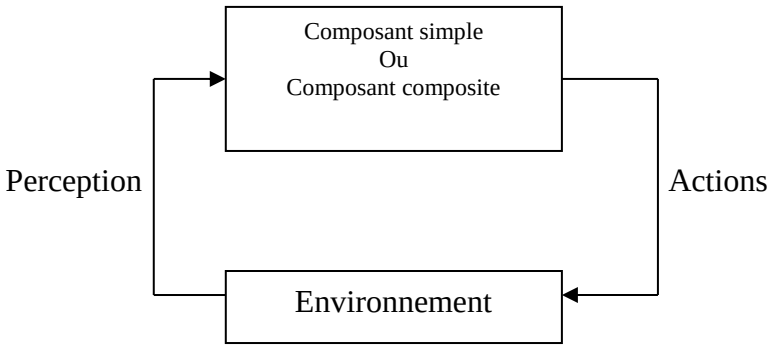


Figure 15 : L'architecture d'agents de DIMA[37]

Pour gérer les interactions entre ces différents composants, DIMA propose un module de supervision représentant le méta-comportement de l'agent. Ce méta-comportement réifie le mécanisme de contrôle de l'agent, il gère les interactions entre les différents modules et permet à l'agent d'observer ces comportements. Un agent est ainsi une entité proactive et autonome. Cette architecture permet de concevoir et réaliser différents types d'agents.
La principale caractéristique de DIMA est son architecture modulaire et des librairies offrant les briques de base pour construire des modèles d'agents divers. Ces différents briques ont pour but d'offrir à l'utilisateur une grande variété de paradigmes (par exemple automate, règle, etc.) d'une part, et d'autre part, une implémentation des différentes propositions conceptuelles introduites par la communauté multi-agents (BDI, KQML, ACL, etc.).

La modularité offre plusieurs avantages [37]:

- Possibilité de définir des agents à granularité variable.
- Possibilité de définir des agents à structure adaptative. Chaque agent peut dynamiquement changer ses composants ainsi que les relations entre ces différents composants.
- Possibilité d'intégrer différents modèles d'agents.
- Possibilité de définir des bibliothèques de composants réutilisable.
- Réduire le temps d'implémentation.
- Faciliter le développement pour des non-spécialistes des SMA.
- Etc.

Dans DIMA, le développement d'un système multi-agents nécessite les principales étapes de développement : Analyse, Conception, Implémentation et déploiement[37].

3-5-1 - Analyse

Dans DIMA, l'analyse commence par la modélisation du domaine. Cette modélisation du domaine est similaire aux processus d'élaboration de modèles conceptuels dans les méthodes objets.

3-5-2- Conception et Implémentation

Les principales étapes pour la conception et l'implémentation d'un agent sont les suivantes :

- détermination de l'ensemble de ses compétences (ou comportements).
- détermination du type de l'agent et son méta-comportement.

3-5-3- Déploiement

Après avoir modélisé le domaine et l'ensemble des agents, DIMA offre deux possibilités :

- exécution des agents sur une seule machine.
- répartition des agents sur plusieurs machines.

L'exécution des agents sur une seule machine nécessite l'initialisation du réseau de communication avant l'activation des agents. La répartition des agents DIMA sur plusieurs machines s'appuie sur le middleware DarX. La répartition des agents nécessite :

- initialisation du serveur DarX sur les différentes machines.
- activation des agents.

3.6. Zeus

Zeus est un environnement intégré pour la construction rapide d'applications à base d'agents collaboratifs. Il est développé par l'Agent Research Program du British Telecom Intelligent System Research Laboratory. La documentation de Zeus est abondante, et insiste particulièrement sur l'importance des aspects méthodologiques de Zeus (« The agent creation methodology is vital to the use of the Zeus toolkit »). La méthodologie de Zeus utilise la même décomposition en quatre étapes pour la construction d'agents, nommées respectivement analyse du domaine, conception, réalisation et support d'exécution[38].

3-6-1-Analyse

La phase d'analyse, qui consiste en la modélisation des rôles. Pour cette étape, aucun outil logiciel n'est fourni. Puisque les agents sont définis par leurs rôles et leurs comportements, cette étape est spécifiquement applicable aux systèmes multi agents orientés rôle, avec des agents rationnels. La modélisation des rôles se fait à l'aide de diagrammes de classe UML et des patrons, ce qui évite l'introduction de nouveaux formalismes et la rend accessible à une plus grande audience. Du point de vue réutilisation, un grand nombre de modèles de rôles récurrent sont donnés, qui couvrent la gestion d'informations, la vente et les processus industriels.

3-6-2-Conception

La conception des agents, qui consiste en la définition de solutions qui remplissent les responsabilités des rôles définis lors de la phase précédente, est appuyée par trois études de cas [38] : FruitMarket (vente), PC Manufacture (chaîne de distribution) et Maze Navigator (agents à base de règles). Le modèle sous-jacent de Zeus limite la conception à des agents collaboratifs orientés tâches, cherchant à satisfaire des buts. Etant donné que le processus menant à trouver une solution à un problème donné est extrêmement difficile à systématiser, cette étape nécessite principalement des qualités de concepteur, mais n'est techniquement pas difficile. Les trois études de cas sont un bon point de départ pour la réutilisation. La simplicité du formalisme de conception utilisé par Zeus facilite la réutilisation, mais n'est pas suffisamment formel, car il est composé principalement de phrases en langue naturelle.

3-6-3-Développement

L'étape de développement des agents est à la fois couverte par les trois études de cas fournies, et par l'Application Realisation Guide [38]. Les cinq activités qu'implique le développement d'agents sont toutes supportées par des outils logiciels graphiques. Ces cinq activités sont : création d'ontologie, création d'agents, configuration de l'agent utilitaire, configuration de l'agent tâche, et implémentation des agents. Ces activités demandent une bonne connaissance des outils, qui sont heureusement bien documentés. Bien entendu cette étape ne s'applique que pour le modèle d'agent de Zeus. Les ontologies peuvent être facilement réutilisées. On ne peut pas réutiliser un agent d'un projet à un autre. La

modularité générale des outils de développement permet la réutilisation des fonctionnalités fréquemment utilisées au sein des agents, mais l'ajout d'une nouvelle fonctionnalité nécessite de retourner au code Java (par exemple, pour ajouter une nouvelle stratégie de coordination).

3-6-4-Déploiement

L'étape de déploiement, est documentée dans le Runtime Guide [38]. Ce document décrit comment lancer le système multi-agents engendré, et comment utiliser l'outil de visualisation. Il y a aussi quelques considérations sur l'art du débogage. L'outil de visualisation permet d'obtenir différents points de vue du système multi-agents : organisation et interactions dans la société d'agents, décomposition globale des tâches, statistiques et état interne des agents. Il est également possible de contrôler l'état interne de tout agent, et même de configurer des agents à l'exécution. L'outil de déploiement utilise une interface utilisateur conviviale qui facilite le déploiement. La modification de la composition de la société d'agents ou de sa localisation nécessite une recompilation, ce qui rend la réutilisation au déploiement quasi impossible.

3.7. Conclusion

Les concepts d'agent, et SMA sont très abstraits et impliquent plusieurs notions complexes comme la coordination, l'interaction, la communication, la représentation des connaissances et autres. Nous avons fait un survol rapide de ces notions et comme nous l'avons vu, plusieurs utilitaires ont été développés pour supporter différentes caractéristiques de ce genre de systèmes.

Cependant, il reste encore beaucoup de travail à faire aux niveaux de la standardisation et de l'uniformisation pour permettre un jour d'avoir des outils capable de réaliser des SMA génériques pouvant résoudre, si on peut le dire, tous les problèmes d'un domaine spécifié.

4-approche proposé avec étude de cas

4-1-Introduction :

Dans le modèle client/serveur, un serveur représente un objet géré sur un site et offrant des services aux clients. Les fonctionnalités d'une application sont alors encapsulées dans des services et proposées/exécutées au sein de serveurs. Les serveurs sont ainsi vus comme des fournisseurs de services. On trouve soit : (i) des serveurs de données, dans lesquels des clients accèdent à des données localisées sur chaque serveur (SGBD, LDAP, etc...), (ii) soit des serveurs de calculs, dans lesquels des clients utilisent les ressources de chaque serveur afin d'exécuter une tâche précise.

Dans le modèle client/serveur, seul le code initialement installé sur le serveur pourra être exécuté sur ce dernier, et le rôle du serveur est de répondre aux demandes des processus clients. C'est le client, demandeur d'un service, qui établit une interaction avec un serveur. Nous pouvons envisager que l'exécution d'une tâche sur un serveur nécessite l'interaction avec d'autres serveurs (consultation d'une base de données). Dans ce cas un même processus peut être à la fois client et serveur. Dans le modèle client/serveur seul le client représente une application au sens propre du terme. Le serveur a pour fonction de répondre aux demandes des clients. Les réponses dépendent des requêtes formulées et non pas des applications clientes qui l'interrogent.

Le client envoie une requête au serveur. Cette requête décrit l'opération à exécuter (service demandé) ainsi que ses paramètres. On dit alors que le client invoque une opération sur le serveur. Le serveur exécute le service demandé par le client et revoie la réponse. Il s'agit d'une invocation synchrone. Dans ce type de communication le client qui émet une requête vers le serveur se bloque, en attente de la réponse du serveur

Dans une architecture client/serveur, les défaillances de sites peuvent conduire à un comportement défaillant d'un service et donc le rendre inutilisable par le client. Plusieurs types de défaillances sont à considérer :

– Une défaillance par arrêt (crash, panne) quand un serveur ne rend pas de résultats suite à des invocations répétées.

– Une défaillance par omission quand un serveur omet de répondre à son client.

– Une défaillance temporelle quand la réponse du serveur est fonctionnellement correcte mais n'est pas arrivée dans un intervalle de temps donné.

– Une défaillance de valeur quand le serveur rend des résultats incorrects.

Pour cela nous allons proposer un système basé sur un ensemble d'agents mobiles et cela vu les avantages que présente ce type d'agents, par exemple, l'exécution d'un processus débute sur le site client. Dans la mesure où le client a besoin d'interagir avec le serveur, ce même processus (code, état d'exécution et données) se déplace à travers le réseau pour continuer son exécution et pour interagir localement avec les ressources du serveur. Apres exécution, l'agent mobile retourne éventuellement vers son client afin de lui fournir les résultats de son exécution.

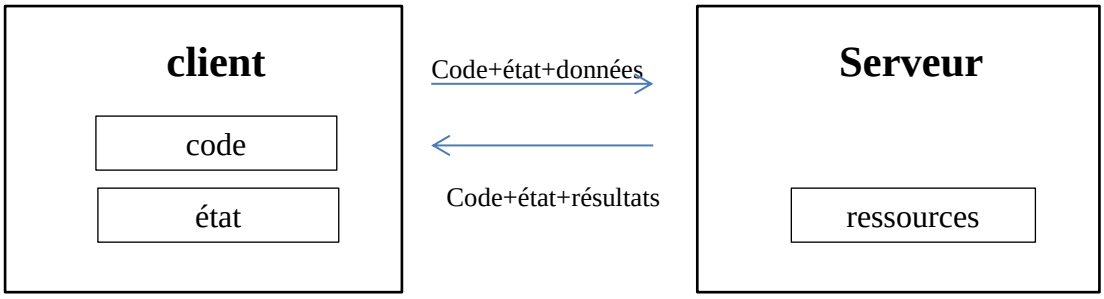


Figure 16 : Agent mobile

Dans ce schéma, le savoir-faire appartient au client, l'exécution du code est initiée coté client et continuée sur les différentes machines visitées

4-2-Objectif du système :

L'objectif de notre travail est de réaliser un Système de collecte des données dans une base de données distribuée, ce système est basé sur l'utilisation d'un système multi agents agissant sur une base de données homogène distribuée. Afin d'atteindre cet objectif, nous avons besoin d'un ensemble d'agents situés et mobiles qui vont se déplacer d'une machine vers une autre, et interagir entre eux.

Le principe est donc de créer un ensemble d'agents mobiles puis les distribués dans le réseau des serveurs de la base de données où ils doivent rechercher et collecter les informations désignées par la requête de l'utilisateur. Cette approche est concentrée sur la comparaison des paquets du trafic réseau par les comportements enregistrés dans une base de flags, le système peut ainsi faire une analyse et exécuter ou non l'action de collection des données.

Ces différents agents doivent effectuer les tâches suivantes :

- ✓ L'agent gestionnaire doit connaitre les adresses IP de tous les ordinateurs du réseau qui sont connectés et créer et distribuer les agents scanneurs et les agents collecteurs.
- ✓ Tous les agents doivent connaitre l'adresse IP du poste administrateur (où il existe l'agent gestionnaire) qui gère le système.
- ✓ L'agent scanneur de paquet (AP) responsable de scanner les paquets circulants dans le réseau, obtenus à partir d'un sniffer (qui a le rôle de capter les paquets circulant dans le réseau)
- ✓ L'agent ACI est responsable de collecter l'information depuis les bases de données du système

4-3-Conception globale du système :

Le système doit contenir :

- Un ensemble d'ordinateur constituant le réseau de l'entreprise, et on parle ici d'entreprise constituée de plusieurs filiales dans des endroits géographiques distinctes.
- Un système d'information distribué, en général, une base de données distribuée gérée par le même SGBD (homogène).
- Un ensemble d'agents situés et mobiles, où chacun a son rôle à réaliser.

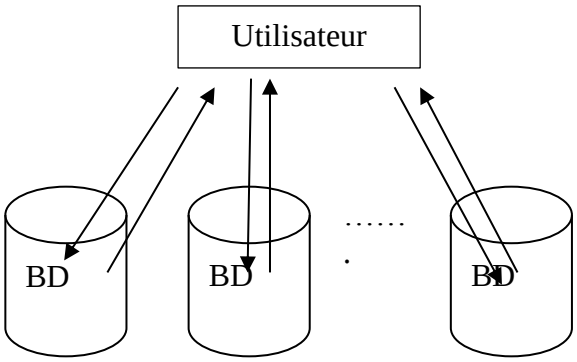


Figure 17 : Conception du système

4-4-Architecture du système :

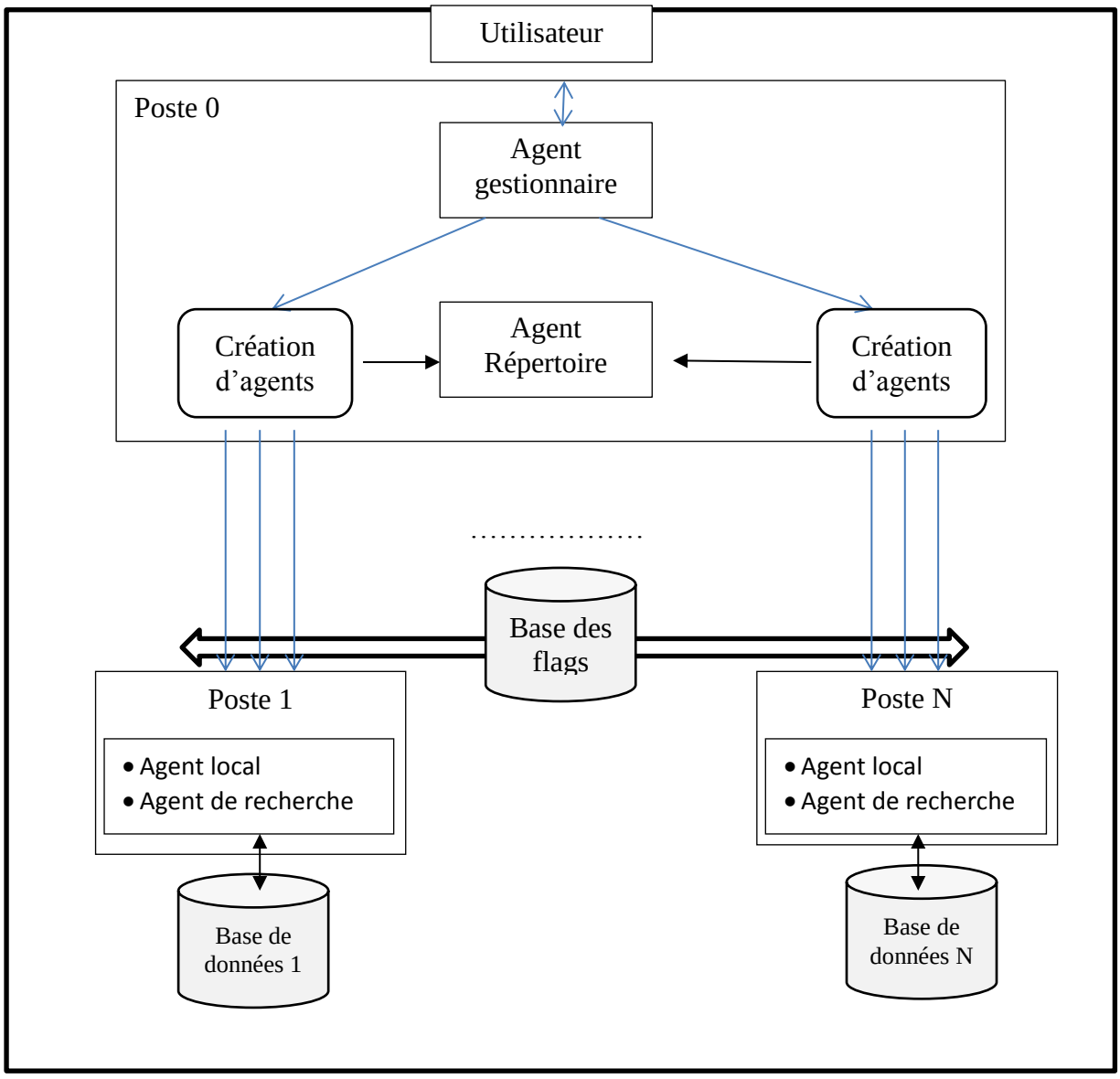


Figure 18 : schéma général du système d'agents mobiles

4-5-**Les agents du système :**

Pour atteindre à l'objectif de notre système, trois agents sont utilisés :

- **Agent gestionnaire :** qui est le centre du modèle (administrateur), Cet agent permet à l'utilisateur d'interagir avec le système. C'est un agent stationnaire qui est responsable d'acquérir la requête des utilisateurs, envoyer ces requêtes aux agents adéquats et présenter les résultats aux utilisateurs.

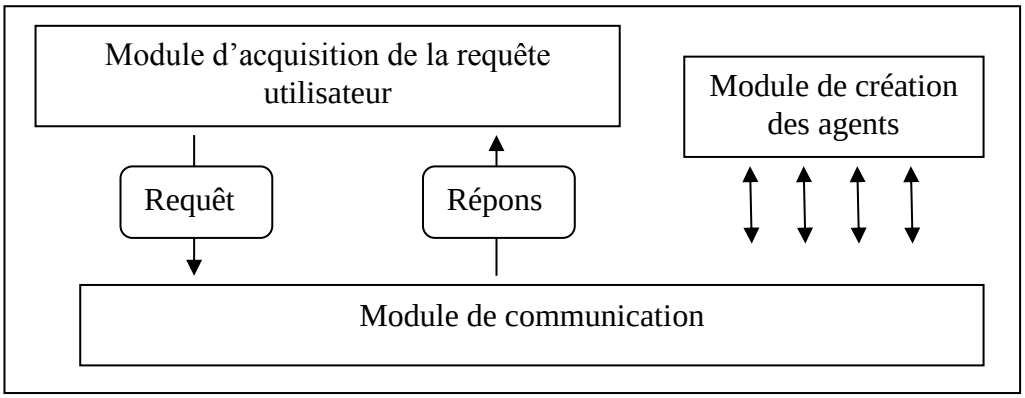


Figure 19 : Architecture de l'agent gestionnaire

Module d'acquisition de la requête utilisateur : Permet de collecter les informations décrites dans un formulaire et transformer les résultats réceptionnés pour être exploités par l'utilisateur.

Module de création des agents : Ce module permet de créer les agents mobiles. Lors de leurs créations, il faut définir l'endroit où l'on souhaite qu'ils démarrent leurs activités.

Module de communication : Cette unité assure le transfert de la requête à l'Agent indexation de la requête et reçoit le résultat de l'agent de recherche.

- **Agent indexation de requête :**

C'est un agent stationnaire qui est responsable de l'indexation de la requête.

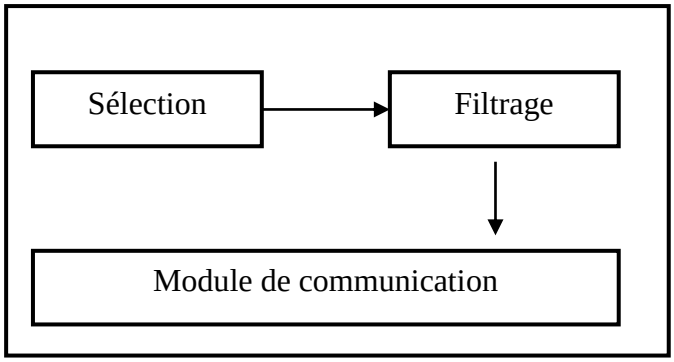


Figure 20 **:** Architecture de l'agent d'indexation de requête.

- **Sélection :** Ce module éliminé les mots vides de la requête afin de ne garder que les importants.
- **Filtrage :** éliminer les mots non utilisés de la requête
- **Module de communication :** Ce module transfert l'index de la requête à l'agent local.

- **Agent répertoire :** gère les références des différents agents du système. Il peut être considéré comme un annuaire décrivant la destination de chaque agent de recherche lors de son déplacement dans le réseau. Dès la création de l'agent de recherche par l'agent gestionnaire, celui-là envoi l'adresse de destination à l'agent répertoire pour permettre d'avoir la trace de l'agent de recherche, et quand cet agent change de site, il envoi sa nouvelle adresse à l'agent répertoire.

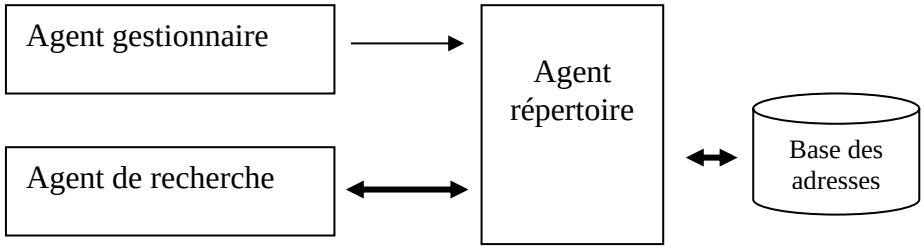


Figure 21 **:** Schéma général de fonctionnement de l'agent répertoire.

- **Agent local :** Il existe dans chaque machine du réseau, cet agent assure la migration de l'agent mobile et lui accorde l'accès aux ressources locales afin d'exécuter le processus de recherche (lancement d'exécution de la requête spécifique à l'utilisateur).

  L'agent local assure les fonctionnalités suivantes :

  - La réception de la requête de l'utilisateur.
  - La migration des agents de recherche.
  - Le renvoi de la réponse à l'agent d'interface.

- **Agent de recherche :** Dans notre système, il y a un groupe d'agents mobiles de recherche. Le nombre d'agents dans ce groupe dépend du nombre d'ordinateurs disponibles dans le réseau, lors de son déplacement, il commence à exécuter son code, et enregistre les informations collectées dans sa mémoire, afin de les envoyer à l'agent local.

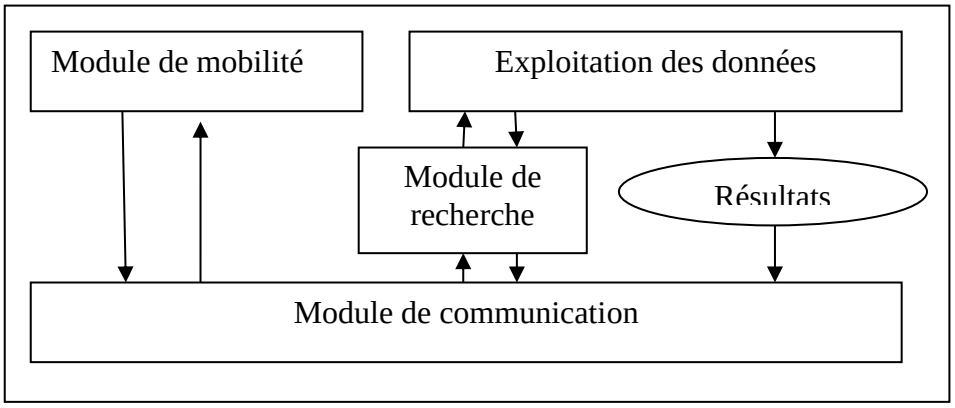


Figure 22 **:** Architecture de l'agent de recherche

Ces différents agents doivent effectuer les tâches suivantes :

- ✓ L'agent gestionnaire doit connaitre les adresses IP de tous les ordinateurs du réseau qui sont connectés et créer et distribuer les agents de recherche et les agents locaux.
- ✓ Tous les agents doivent connaitre l'adresse IP du poste administrateur (où existe l'agent gestionnaire) qui gère le système.
- ✓ L'agent de recherche responsable de scanner les paquets obtenus à partir d'un sniffer (qui a le rôle de capter les paquets circulant dans le réseau)
- ✓ L'agent local est responsable de collecter l'information depuis les bases de données du système

4-6-L'aspect fonctionnel des agents du système :

4-6-1-L'agent gestionnaire :

Les fonctions de l'agent gestionnaire sont représentées dans la figure suivante :

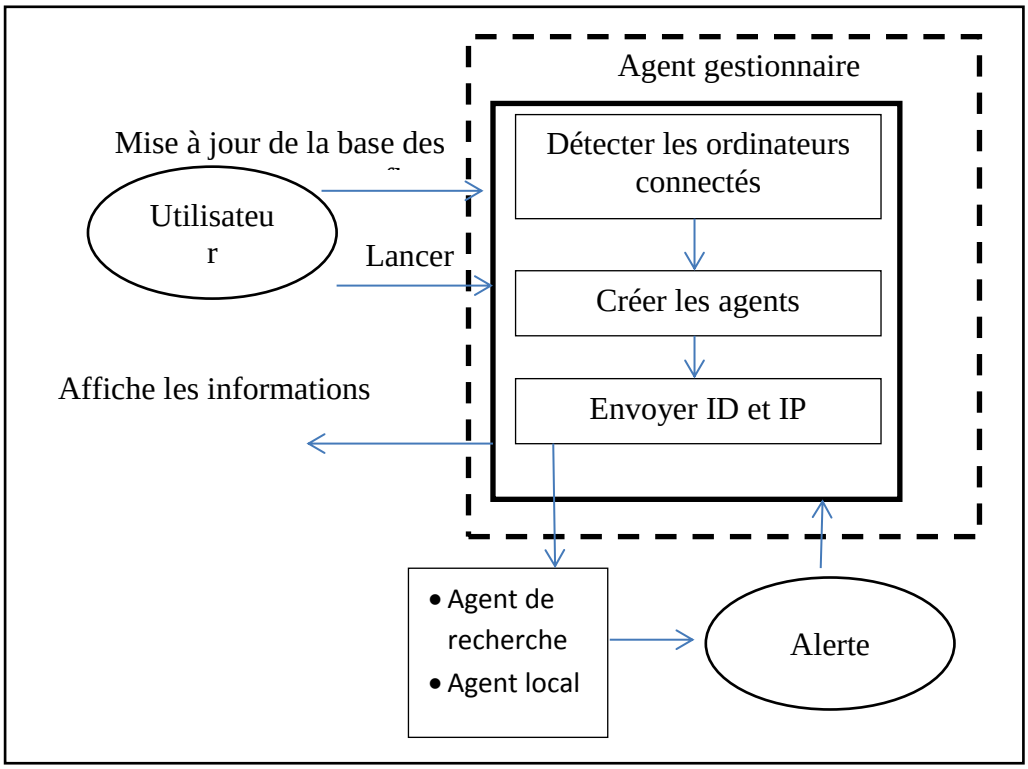


Figure 23 : schéma fonctionnel des tâches de l'agent gestionnaire

En entrée, on a :

- L'agent gestionnaire est lancé par l'utilisateur (après l'identification)
- L'agent gestionnaire reçoit une alerte en cas de détection d'une information
- La mise à jour de la base des flags indiquant la vérification du résultat venant d'un poste du réseau.

En sortie, on trouve :

- La création des agents locaux et de recherche du système.
- L'agent gestionnaire affiche une alerte indiquant l'information et l'action à faire.

Les fonctions de l'agent gestionnaire :

- Détecter les ordinateurs connectés au réseau
- Créer les agents du système
- Envoyer son identificateur (ID) et son adresse (IP) à tous les agents créés.
- Garder les agents internes à l'administrateur et dispatcher les autres.

4-6-2-Agent de recherche :

L'agent de recherche doit effectuer les tâches qui sont représentées dans la figure ci-dessous :

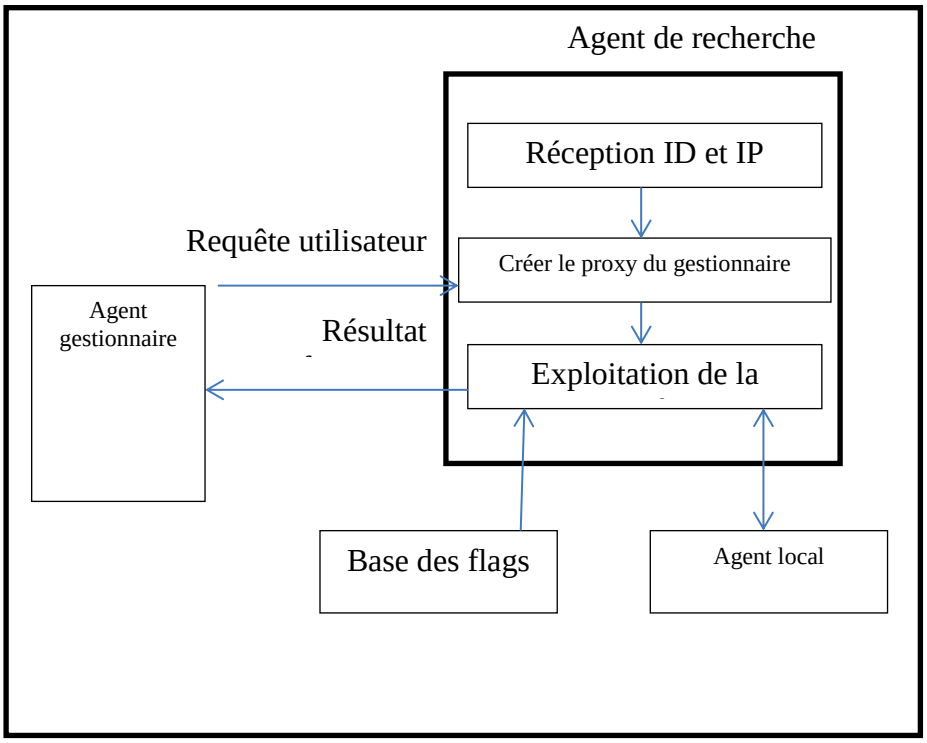


Figure 24 : schéma fonctionnel de l'agent de recherche

Ce schéma présente les fonctions de l'agent de recherche

En entrée, on a la requête utilisateur, et en sortie, il envoie les résultat fournis par l'agent local à l'agent gestionnaire si l'information est détectée.

Les fonctions :

- Créer le proxy de l'agent gestionnaire à l'aide de l'ID et l'IP.
- Recevoir le résultat de la requête après son exécution
- Transmettre le paquet de résultat à l'agent gestionnaire

4-6-3-Agent local :

L'agent local à la tache de collecter les informations depuis le système d'informations, la figue suivante représente les fonctions de l'agent local

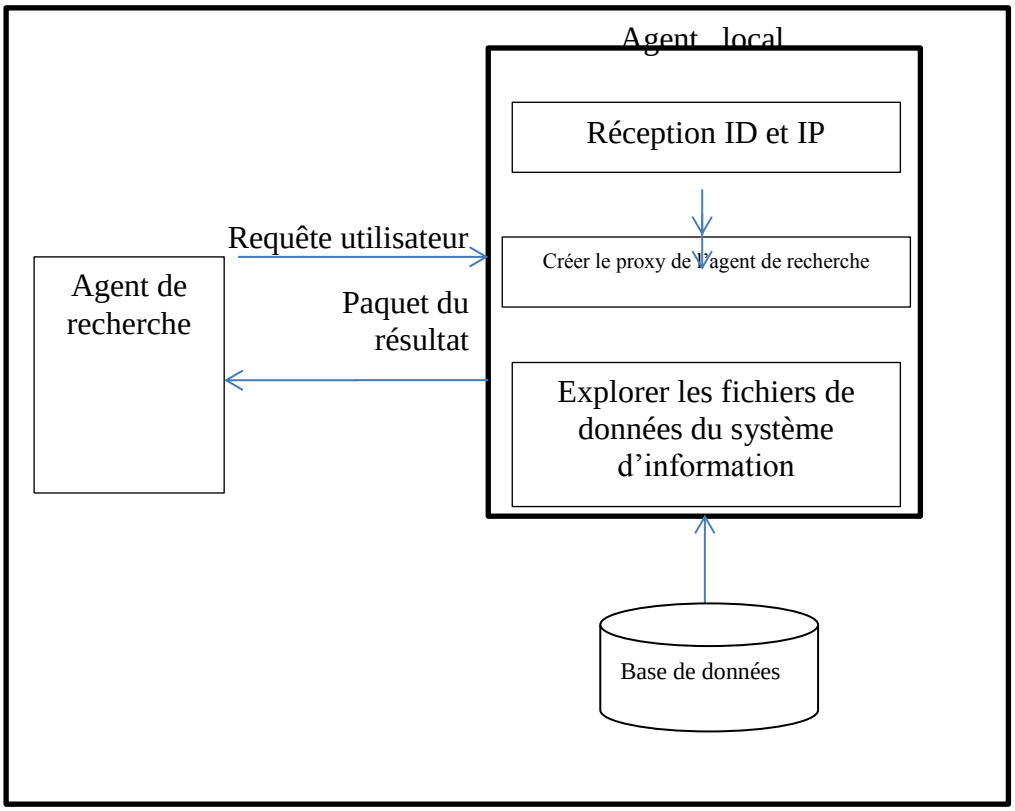


Figure 25 : schéma général des tâches de l'agent local

Ce schéma présente les activités que doit effectuer l'agent local, en entrée, il reçoit la requête utilisateur de l'agent de recherche, et en sortie, il envoie le paquet de résultat à l'agent de recherche si l'information a été détectée.

Les fonctions de l'agent local :

- Créer le proxy de l'agent de recherche à l'aide de l'ID et l'IP.
- Exécuter la requête utilisateur pour avoir les données qui se trouvent dans les tables de la base des données

4-7-Communication entre les agents :

Notre système fournit les primitives de communication permettant aux agents d'interagir entre eux. Cette communication est de type de messagerie asynchrone qui ne bloque pas l'exécution, nous avons utilisé ce type de messagerie parce qu'il est utile en particulier quand plusieurs agents se communiquent ensemble.

4-8- Implémentation de notre approche mobile.

Pour l'implémentation du modèle d'agent mobile précité nous avons choisi la technologie Java plus répandue pour ce type de code. En plus depuis 1998, Sun a défini une spécification basée sur Java permettant l'implémentation des systèmes distribués correspondant à notre approche.

A ce jour, plusieurs implémentations de cette spécification sont disponibles : JSC and Seven, Servicehost, Rio, Harvester, H2O et CoBRA. [39]

Dans notre cas, nous avons opté pour l'implémentation nommée Jini, porte le même nom que la spécification. Jini est une architecture réseau destinée à la construction de systèmes distribués sous la forme de services coopérants modulaires, introduite par Sun dans l'idée de fédérer un groupe d'utilisateurs et des ressources en faisant abstraction de toute dépendance à l'égard des systèmes d'exploitation. Ceci en considérant les périphériques et les logiciels comme des objets ou une combinaison d'objets indépendants pouvant communiquer. Ces ressources matérielles ou logicielles peuvent être connectées à un réseau pour se déclarer et fonctionner dès qu'elles sont branchées. Chaque client qui désire les utiliser peut les localiser et faire appel à elles afin d'exécuter certaines tâches. Le but de cette technologie, où la responsabilité a été transférée à Apache sous le nom de projet River, est de rendre un réseau plus dynamique pour mieux refléter la nature dynamique du travail d'équipe (collaboratif) en donnant les droits d'ajout ou de suppression des services d'une manière flexible.

4-8-1- Les composants de Jini

Les composants de Jini sont une extension réseau de l'infrastructure, du modèle de programmation et des services de la technologie Java.

- L'infrastructure est l'ensemble de composants qui permet d'établir un système fédéré de Jini. Ceci inclut le service de recherche ou de localisation "Lookup Service" ainsi que le protocole d'enregistrement et de découverte (Discovery/Join Proocol)
- Le modèle de programmation est un ensemble d'interfaces qui permet la construction des services fiables incluant ceux qui font partie de l'infrastructure et ceux qui se joignent à la fédération. Ce modèle peut intégrer la notion de bail et les événements distants. Il peut compter sur la possibilité de Java pour déplacer des objets à travers les machines virtuelles (JVM).
- Les services sont des entités dans la fédération. Ils utilisent l'infrastructure pour la communication, la découverte et l'annonce de leur présence.

La séparation entre ces catégories, qui s'appuie sur la distinction, Java n'est pas toujours claire puisque les composants de chaque catégorie sont interdépendants.

	infrastructure	Modèle de programmation	services
Base Java	Java JVM RMI Java security	Java APIs Java beans	JNDI Entreprise beans …..
Java + Jini	Discovery/join Distributed security lookup	Leasing Transactions Events	Printing Transaction manager …..

Tableau 1 : Les composants JINI

Dans notre approche, tous les agents sont considérés comme des composants Jini pour bénéficier de l'infrastructure, les modèles de programmation et aussi les services. Cette considération va leur permettre d'établir une fédération d'agents ce qui rejoint la notion de la communauté.

4-**8-1.1. Services**

La notion de service est le concept le plus important dans l'architecture Jini. C'est son utilisation qui permet au client d'exécuter des traitements, utiliser une unité de stockage et communiquer… Le système Jini se compose de services qui peuvent être regroupés pour exécuter des tâches particulières. Un service peut utiliser un autre service. Autrement dit, un service peut être le client d'un autre service tout en utilisant son propre service. La nature dynamique du système Jini permet à des services de s'enregistrer ou de se retirer d'une fédération de services à tout moment selon la demande, le besoin, ou les conditions du groupe de travail qui les utilise.

Un service Jini est défini par une interface Java. Et peut être identifié par cette même interface. Il peut être implémenté de différentes façons. Toutefois, le client et les différentes implémentations semblent identiques parce qu'ils implémentent la même interface. D'un point de vue global, le fournisseur du service doit implémenter l'interface du service, enregistrer l'instance du service dans le service de recherche (Lookup Service) pour qu'il soit visible dans le réseau et gérer le cycle de vie du service en restant en attente.

Ceci est similaire à la création d'un objet distant basée sur l'appel des méthodes distantes (RMI). En réalité Jini utilise RMI comme protocole de communication. Quant aux transmissions de données, elles se font à travers un système de couches comme le montre le schéma suivant.

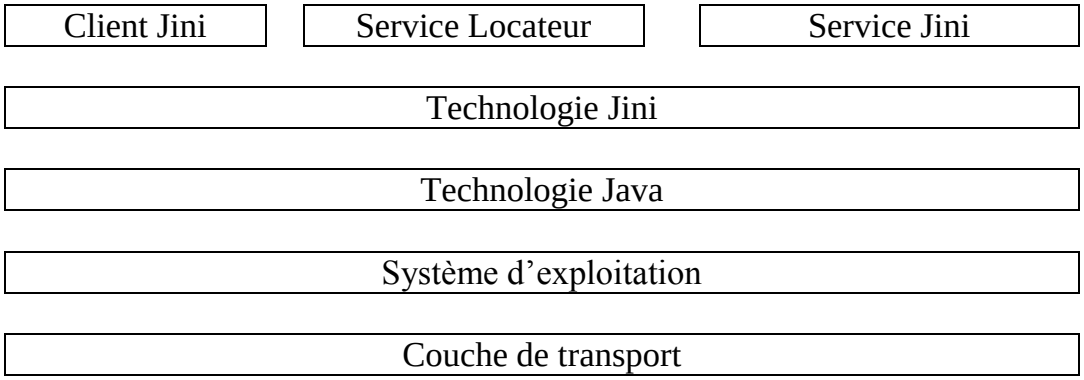


Figure 26 : Architecture de JINI

L'interface d'un service doit hériter de l'interface Remote qui sert à créer le lien avec le système Jini et à identifier les interfaces dont les méthodes peuvent être appelées depuis une machine virtuelle distante (soit sur la même machine, soit sur une machine différente).

Il faut noter que tous les agents ainsi que les tâches sont des services Jini. Le diagramme de classe suivant présente le lien entre notre modèle et le système Jini.

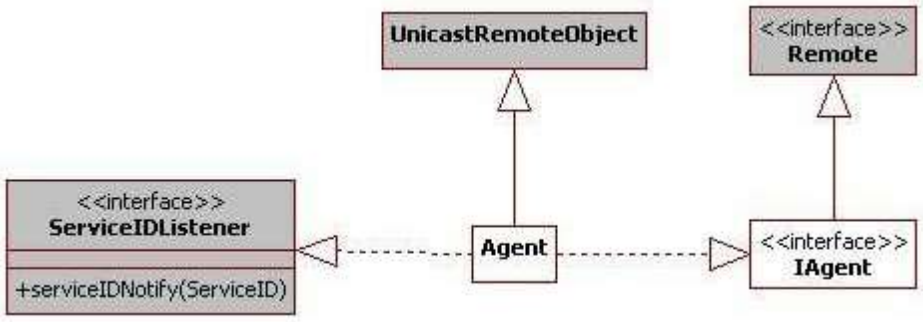


Figure 27 : diagramme de classe du service JINI

L'instance du service peut être sérialisée et transmise au client. Seulement, dans la plupart des cas, cette dernière dépend des ressources locales (ex. CPU, fichier, imprimante…). C'est pour cette raison qu'elle doit rester active sur la JVM locale.

4-8-1.2. Lookup service

Le "Lookup Service" est le mécanisme central de la fédération de Jini. Il fournit le point de contact entre le fournisseur de service et le client.

Pour localiser le "Lookup Service", le client et le fournisseur du service utilisent le mécanisme de découverte. Cette découverte peut se faire de deux façons différentes : si l'emplacement du Lookup service est connu, alors le client utilise "Unicast discovery" pour se connecter directement. Dans le cas contraire, il envoie une requête "Multicast discovery" pour rechercher dans le réseau un ou plusieurs "Lookup Service".

Quand une requête est envoyée au "Lookup service", il retourne un objet de type "Registrar" au client. Cet objet a le rôle d'un "Proxy". Et, toutes les requêtes destinées au "Lookup service" passent par lui.

Quand le "Lookup Service" reçoit le modèle, il recherche dans tous les services enregistrés puis retourne les services qui correspondent au modèle.

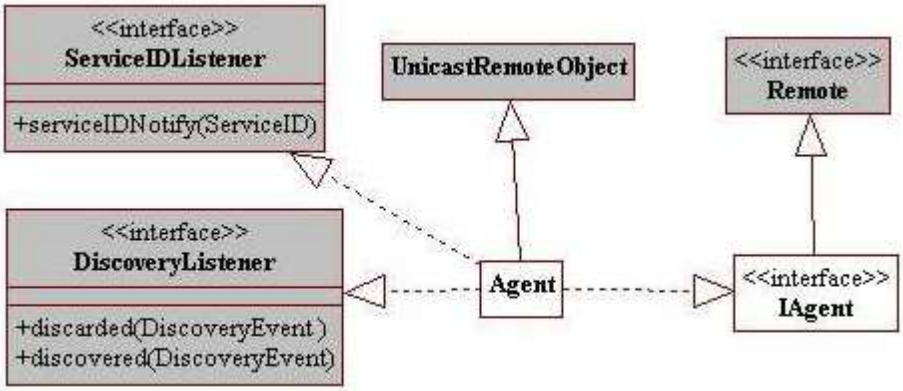


Figure 28 : diagramme de classe du service de découverte

4-8-1.3. Transaction

Les transactions sont des fonctionnalités nécessaires voir primordiales pour des opérations distribuées. Fréquemment, deux objets ou plus doivent synchroniser le changement d'un état, s'il a eu lieu. Ce changement peut se produire dans plusieurs situations, telles que la mise à jour d'une information, où une partie des objets peuvent la faire au même temps que d'autres sont incapables de l'effectuer, en conséquent le résultat final sera incohérent.

Les transactions utilisent un protocole de validation (commit protocol) biphasé. Ceci implique que les participants à une transaction soient invités à voter pour cette dernière. Si tous les participants sont d'accord alors la transaction est validée, dans le cas contraire elle est annulée pour l'ensemble des participants.

Chaque agent doit pouvoir participer à une transaction dans le cadre de leurs activités afin de s'assurer que les données partagées ou informations échangées sont valides. Cela implique l'implémentation de l'interface TransactionParticipant, comme le montre le diagramme de classe suivant, pour utiliser le mécanisme Jini gérant les transactions.

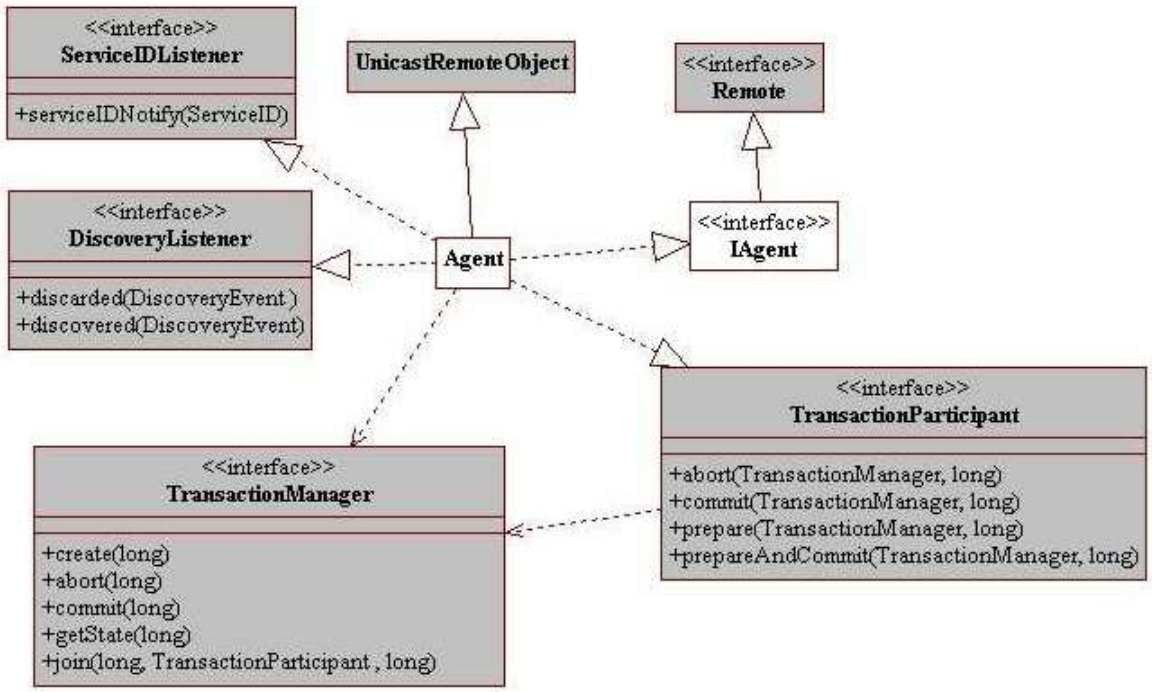


Figure 29 : diagramme de classe du service de transaction

4-8-1.4. Remote Events

Le mécanisme des événements est utilisé en liaison asynchrone dans Jini. C'est un mode crucial à l'utilisation efficace de la technologie. Il est intéressant et précieux pour considérer la nature distribuée des événements. C'est pour cette raison que le modèle existant de délégation d'événements n'est pas utilisé.

Jini utilise un seul type d'événement qui se résume en une simple classe *RemoteEvent*. Cette classe contient le strict minimum d'informations pour réduire les échanges réseau afin d'éviter la latence et la surcharge. L'objectif de l'utilisation des événements est de notifier des applications ayant pour but de monitorer l'activité des agents. Nous pouvons également, utiliser les événements pour que les agents se notifient entre eux pour certains changements d'état, l'inconvénient est que notre modèle sera lié à une technologie chose qui n'est pas souhaitée d'autant plus la classes *RemoteEvent* ne permet pas des échanges riches.

Jini ne spécifie pas comment le client doit s'enregistrer pour recevoir les événements distants. Le développeur du service doit prévoir une méthode pour écouter l'adhésion. Néanmoins, Jini ne spécifie pas une classe qui puisse être utilisée comme type de retour de la méthode d'écoute

4-8-1.5. Communication Protocol

Jini fournit une grande capacité de communication entre les différents composants de son architecture incluant des protocoles ouverts et finis, et définissant un certain nombre d'entre eux. Dans les versions précédentes de Jini (version 1), Jini utilise le protocole Java Remote Method Protocol (JRMP), lui-même basé sur TCP/IP comme protocole de communication. Le JRPM est une implémentation Java de RMI, mais son concept est différent de celui de Jini. D'où le développement de Jini

Extensible Remote invocation (JERI) qui permet une importante flexibilité et résout quelques lacunes de RMI. En voici quelques exemples :

- Le RMI force la génération des souches (Stubs) au moment de la compilation. Contrairement à JERI qui le génère automatiquement quand un objet est exporté. Ainsi, il réduit le temps d'exécution.
- JRMP utilise seulement les connexions TCP, JERI offre d'autres possibilités.
- Un ramasse-miettes distribué (DGC) plus flexible que celui de RMI.
- Une personnalisation beaucoup plus grande. Un développeur peut utiliser sa propre implémentation du protocole de transport ou disposer d'autres couches de JERI.

4-**8-1.6. Configuration**

Pour construire et maintenir les services, plusieurs paramètres sont nécessaires (ex. attributs du service, l'url du chargement dynamique…). Ces paramètres sont fréquemment spécifiés en dur dans le code source. Ce qui implique une recompilation à chaque fois que nous les changeons. Pour cette raison, nous pouvons utiliser la classe Configuration qui permet de spécifier les paramètres à l'exécution.

Une configuration peut être utilisée de différentes façons. Le développeur peut, par exemple, implémenter sa propre configuration, pour rechercher un objet de configuration d'une base de données ou par l'intermédiaire d'un canal bloqué. Mais, la plupart du temps, nous utilisons la classe ConfigurationFile pour spécifier la configuration dans un fichier.

La syntaxe de ce dernier est basée sur le langage Java avec un minimum d'instructions sans itérations ni conditions :

- Affectation de variables
- Création d'objets
- Les méthodes statiques à invoquer.

Un développeur est libre de décider quelles propriétés peuvent être stockées dans un fichier de configuration.

Le code ci-dessous montre un exemple d'un fichier de configuration du Lookup Service. Il permet de définir les paramètres nécessaires pour construire une instance NonActivatableServiceDescriptor. Les paramètres représentent respectivement l'url du chargement dynamique, le fichier des privilèges, le classpath du service, le nom de la classe, les arguments de configuration du serveur qui peuvent être aussi un autre fichier de configuration.

```
import com.sun.jini.start.NonActivatableServiceDescriptor;
import com.sun.jini.start.ServiceDescriptor;
import com.sun.jini.config.ConfigUtil;
com.sun.jini.start {
        private static policy = "priv.policy";
        private static host = ConfigUtil.getHostName();
        private static port = "80";
        private static jskdl = " http://" + host + ":" + port + "/jsk-dl.jar";
        serviceDescriptors = new ServiceDescriptor[]{
                new NonActivatableServiceDescriptor(
                        "http://" + host + ":" + port + "/reggie-dl.jar" + jskdl,
                        policy,
                        "C:\\jini2_1\\lib\\reggie.jar",
                        "com.sun.jini.reggie.TransientRegistrarImpl",
                        new String[] { "transient-reggie.config" })
        };
}
```

4-9-Etude de cas :
-Gestion budgétaire d'un ministère et ses annexes :

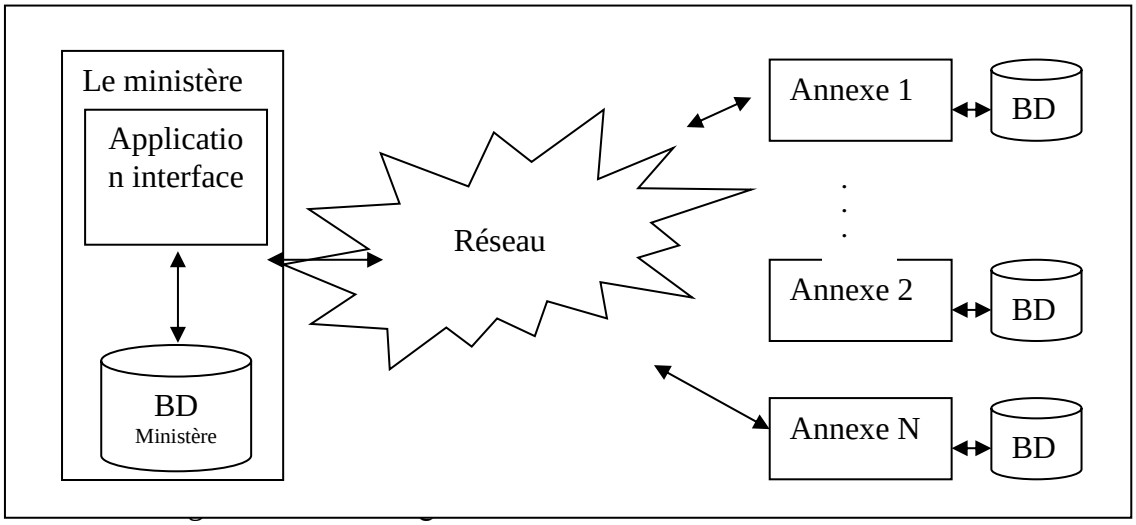
Pour tester notre système nous avons choisi un système de gestion budgétaire entre un ministère et ses annexes (directions de wilaya, établissements à caractère économique…) où, il doit y avoir évidement un réseau liant les différentes annexes au ministère, dans chaque annexe existe un réseau contenant un serveur de données où est située une base de données de la gestion budgétaire.

Comme tous les gestionnaires, la direction centrale du budget dans le ministère doit avoir la situation exacte des consommations de crédits, demander un transfert de fonds excédant dans un établissement, demander les besoins des établissements et toutes les activités relevant de la gestion des budgets, toutes ces transactions se font d'une manière traditionnelle avec des envois par fax , télex ou voie postale, ce qui provoque de différentes anomalies comme, par exemple, la perte des informations ou la lenteur de la procédure de ramassage des informations ce qui aboutit toujours à une distribution non équilibrée entre les direction

Alors pour avoir des réponses exactes, dans des délais très courts le gestionnaire au niveau central, pourra envoyer sa requête à une ou plusieurs directions et recevoir les résultats des informations demandées sans perte de temps ou intervention des gestionnaires des annexes.

Notre système va nous permettre de réaliser une solution distribuée qui va aider les décideurs au niveau central (le ministère) d'avoir l'information exacte à tout moment

Comme tout autre système, certaines conditions doivent être réalisées pour assurer le bon déroulement du processus, et parmi ces conditions, il faut que toutes les bases de données distribuées dans le réseau soient homogènes c'est-à-dire qu'elles sont constituées des mêmes tables et types de données ainsi qu'elle sont prisent en charge par le même système de gestion de base de données.



Fonctionnement du système :

- L'utilisateur et à partir de son poste réseau lance une application interface qui doit obligatoirement démarrer les services Jini pour permettre l'utilisation des différentes classes de création des agents mobiles.
- Le premier agent créé après le lancement du système et l'agent gestionnaire, résidant dans la machine du client, il est délégué de lancer un algorithme de détection des machines connectées et recevoir leurs adresses IP et leurs identificateurs.
- L'agent gestionnaire crée et diffuse les agents de recherche et agents locaux dans le réseau qui ont tous le même rôle et qui est la collecte des données spécifiées par la requête que l'utilisateur à introduit.
- Après changement d'état, les agents retournent les données résultantes à l'agent gestionnaire qui à son tour les affiche pour l'utilisateur.

Le schéma suivant illustre les étapes de fonctionnement du système

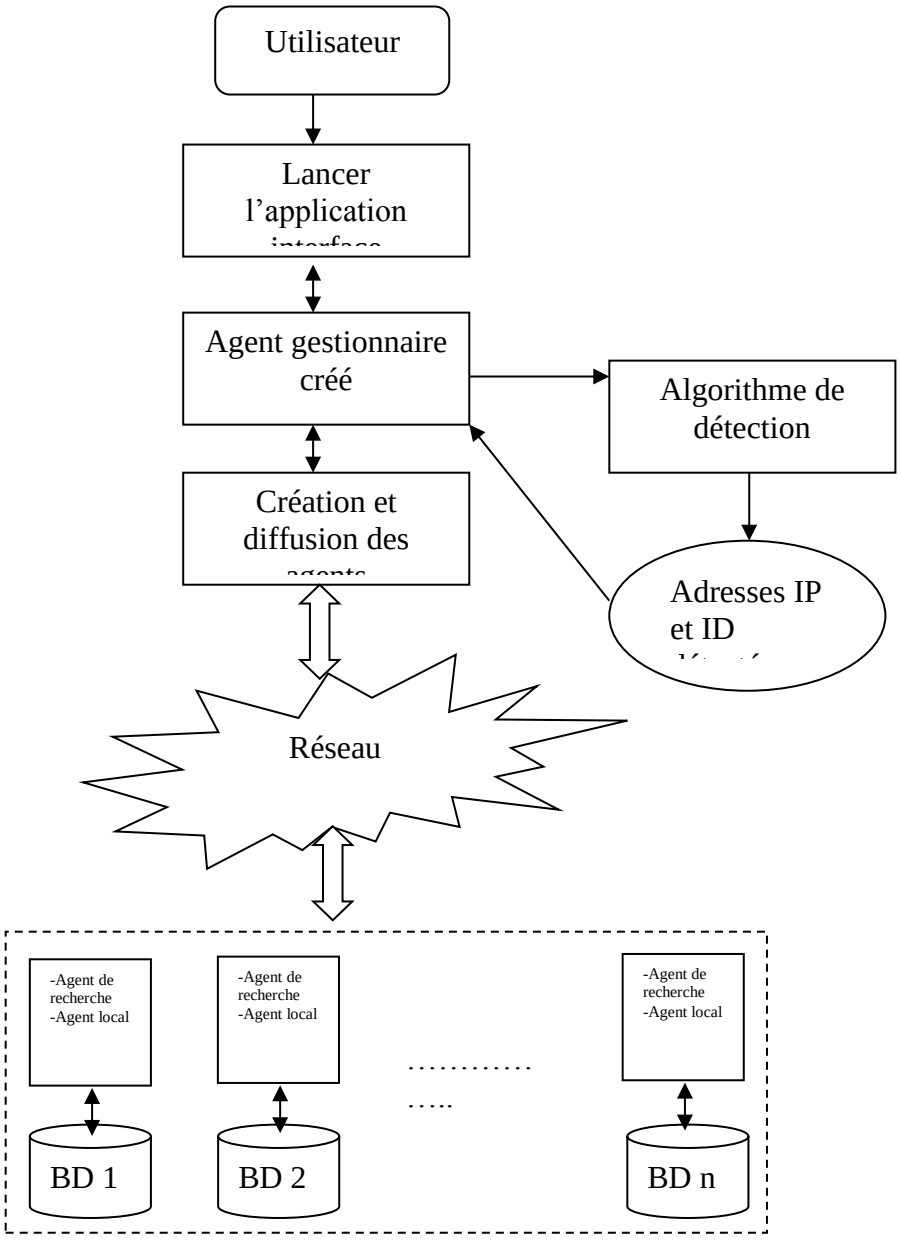


Figure 31 : Schéma fonctionnel du système dans un ministère et ses annexes

Exemple : Le ministère de la jeunesse et des sports

Le ministère de la jeunesse et des sports gère un très grand nombre d'établissements et direction à travers le territoire national à savoir :

Type d'établissement	Nombre
Direction de wilaya	**48**
Office des établissements de jeunesse	**48**
Office du parc omnisports	**35**
Ecole nationale des cadres de jeunesse et sports	**5**
Ecole supérieure des sports	**1**
Ecole nationale/régionale des sports olympiques	**3**
Centre de regroupement des équipes nationaux	**1**
Fédération sportive	**26**
TOTAL……..	**166**

Tableau 2 : Les établissements gérés par le ministère de la jeunesse et sports

Ce nombre important d'établissements et direction indique la grande masse d'informations manipulée par des méthodes traditionnelles, ce qui provoque la lenteur des procédures et l'inégalité dans la distribution du budget global du ministère.

Donc pour remédier à ces problèmes, nous avons entamé, en premier lieu à une opération de réalisation d'une application de gestion budgétaire qui fonctionne localement dans les différents établissements (direction de la jeunesse et des sports de la wilaya de Biskra par exemple, où nous avons installé une application de gestion de budget réalisée sur l'environnement DELPHI 7 et nous avons utilisé Microsoft Access 2010 pour la réalisation de la base de données locale), et puis, nous avons commencé à l'implémentation de notre système en utilisant les outils de programmation java (Eclipse, Jini), mais sur un réseau local pour le moment et cela dans l'attente d'une prise en charge de l'opération de réalisation d'un réseau intranet qui relie toutes les établissements du ministère.

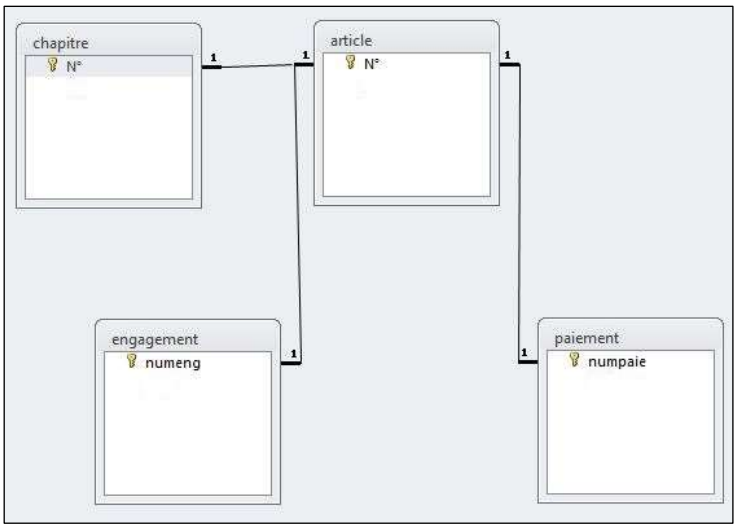


Figure 32 : Un sous schéma de la base de données gestion budgétaire

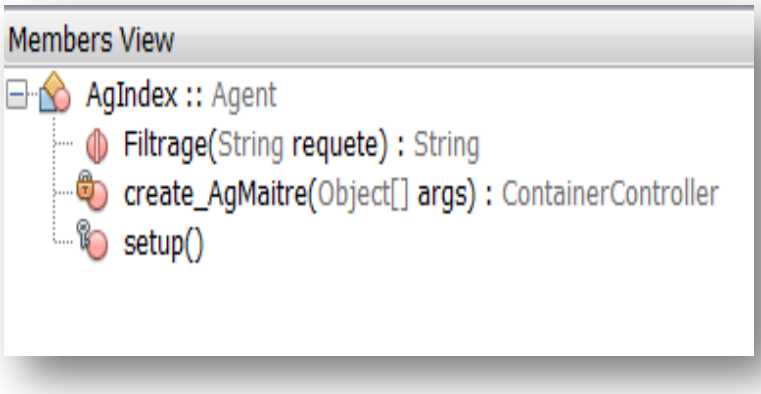


Figure 33 : Pseudo code de l'agent indexation requête.

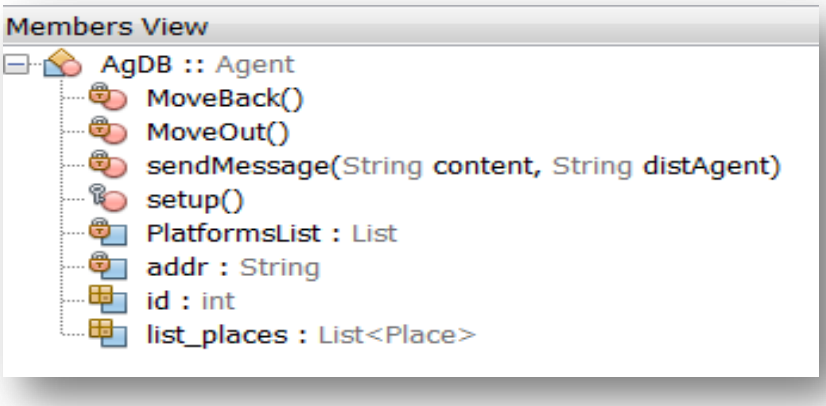


Figure 34 : Pseudo code de l'agent local

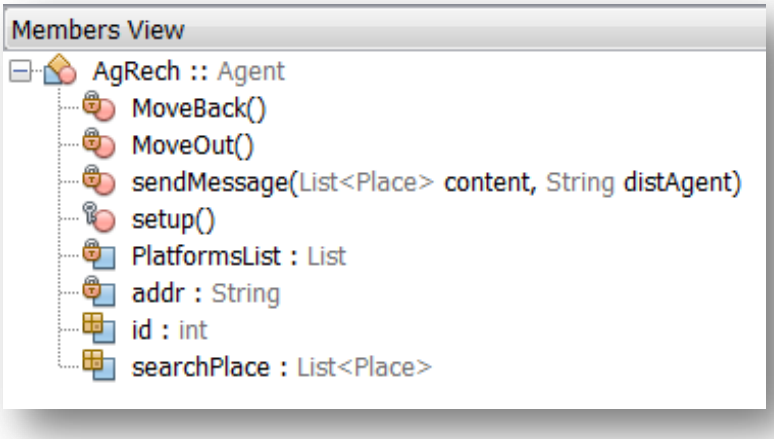


Figure 35 : Pseudo code de l'Agent Mobile Recherche

4-10-Conclusion

Dans ce chapitre, nous avons présenté une architecture basée agent mobile pour la collecte des données dans une base de données distribuée, en définissant chaque agent indépendamment, son architecture, son aspect fonctionnel ainsi la communication entre les différents agents composant notre système.
Après, et pour valider le système, nous avons présenté son implémentation, tout d'abord présenté l'environnement de développement ainsi que les différents outils utilisés, puis nous avons présenté une étude de cas où nous avons utilisé un exemple d'un ministère avec ses annexes pour tester et valider l'implantation du système.

Conclusion générale :

Notre étude a abouti à mettre en place un système à base d'agents mobiles au sein d'un réseau contenant une base de données distribuée, gérée par le même SGBD et construite avec la même structure des tables dans les différents sites du réseau.

En premier lieu, nous avons survolé les éléments primordiaux dans les systèmes multi agents en spécifiant particulièrement les différentes définitions et architectures des agents à travers l'aspect structurel et fonctionnel de chaque architecture d'agent, et les différentes architectures des systèmes multi agents à savoir l'architecture centralisée (tableau noir), l'architecture hiérarchique et l'architecture distribuée.

Ensuite nous avons vu des méthodologies de développement, des langages de communication, des techniques de coordination et coopération qui sont nécessaires pour qu'un ensemble d'agents puisse former un système multi agents. Nous avons fait un survol rapide sur les concepts de la coordination, l'interaction, la communication, la représentation des connaissances et autres, et comme nous l'avons vu, plusieurs utilitaires ont été développés pour supporter différentes caractéristiques de ce genre de systèmes.

En dernier, nous avons développé notre approche basée sur un ensemble d'agents mobiles permettant la collection des données dans une base de données distribuées à travers un réseau d'entreprise ce qui permettra d'avoir une information en temps réduit et fiabilité complète assurée par la non perte des données et la confidentialité pendant leur transfert, et pour assurer la fiabilité de notre système nous avons illustré son utilisation par l'intégration de notre système d'agents mobiles dans le système de gestion budgétaire d'un ministère et ses annexes, permettant aux décideurs au niveau central de contrôler et d'avoir l'information à tout moment et avec grande efficacité, en prenant comme exemple le ministère de la jeunesse et des sports

En perspective, nous voyons que notre système peut être développé et amélioré pour permettre la collection des données d'une base de données distribuée avec moins d'exigence pour l'homogénéité, par exemple des différents SGBD pour la gestion des tables des bases de données d'un système d'information.

Bibliographie

[01]	Idrissa sarr – these de doctorat « Routage des transctions dans des bases de données à large echelle » Octobre 2010
[02]	Y. Shohan, *Agent-Oriented Programming*, Artificial Intelligence, Vol. 60, No. 1, p. 51-92 Mars 1993.
[03]	WEISS G., "Multi agent systems, A modern Approach to distributed Artificial Intelligence", MIT Press, 1999
[04]	J. Quinqueton, M.C. Tomas, B. Trousse, Actes des 5ème journées francophones JFIADSMA97 – Intelligence artificielle et systèmes multi agents Hermes 1997
[05]	J.P. Barthès, V. Chevrier, C. Brassac, Actes des 6ème journées francophones JFIADSMA98 –Systems multi agents – de l'interaction à la socialite - Hermes 1998
[06]	A. Caglayan, C. Harrison, Les agents applications bureautiques, internet et intranet – Masson 1998.
[07]	J.Ferber, les systèmes multi agents, vers une intelligence collective, intereditions, Paris,1995.
[08]	M. d'Inverno et al. A formal specification of dMARS. Dans Intelligent Agents IV, Editeurs: A. Rao, M.P. Singh et M. Wooldrige, LNAI Volume 1365, Springer-Verlag, 1997, p.155-176.
[09]	R. A. Brooks. Intelligence without reasoning. Dans Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91), 1991, p.569-595.
[10]	Muller. A cooperation model for autonomous agents. Dans Intelligent Agents III, LNAI Volume 1193, Editeurs: J.P. Muller, M. Wooldrige et N.R. Jennings, Springer-Verlag, 1997, p.245-260.
[11]	Joël Quinqueton, Outils logiciels des Systèmes Multi-Agents, LIRMM, Montpellier, France
[12]	N. Kabachi, Intelligence artificielle distribuée et systèmes multi agent – support de cours Site : http://liris.cnrs.fr/alain.mille/enseignements/Master_PRO/TIA/SMA/Support_ CoursIAD-SMA.pdf
[14]	Scott A. Deloach ans Mark F. Wood, Multiagent systems engineering : the analysis phase – Technical report AFIT, OHIO - USA – June 2000
[15]	http://www.agentbuilder.com
[16]	http://www.labs.bt.com/projects/agents/zeus
[17]	T. Garneau et S. Delisle, Programmation orientée agent : évaluation comparative d'outils et environnements – JFIADSMA Octobre 2002- Lille, France
[18]	www.cs.cmu.edu/~softagents/retsina.html
[19]	portal.acm.org/citation.cfm?id=974009.974022
[20]	www.ai.sri.com/~oaa/ - 3k
[21]	www.lsi.upc.edu/~jvazquez/teaching/sma-upc/ - 26k
[22]	www.lirmm.fr/~jq/Cours/3cycle/UMINR303-3a.pdf
[23]	D. Grimshaw, CPS 720 Artificial Intelligence Topics with Agents, Fall 2001
[24]	http://www-ksl.stanford.edu/knowledge-sharing/kif/

[25]	http://www.cs.umbc.edu/kqml/papers/desiderata-acl/section3.5.html
[26]	http ://www.fipa.org
[27]	www.fipa.org/specs/fipa00011/XC00011A.pdf.
[28]	Salah el falou – thèse de doctorat « programmation répartie, optimisation par agents mobiles » Novembre 2006
[29]	Mohamed Amine KAMOUN – thèse de doctorat « Conception d'un système d'information pour l'aide au déplacement multimodal : Une approche multi-agents pour la recherche et la composition des itinéraires en ligne. » Avril 2007
[30]	Mounir Beggas, Modélisation par un système multi-agents d'un hypermédia éducatif adaptatif dynamique – Mémoire de Magister en Informatique - Centre Universitaire d'Eloued -
[31]	R. Querrec, Les Systèmes Multi-Agents pour les Environnements Virtuels de Formation, Application à la sécurité civile, thèse de doctorat Informatique, Laboratoire d'Informatique Industrielle (LI2/ENIB), Université de Bretagne Occidentale. Octobre 2002
[32]	http://jade.cselt.it/
[33]	http://www.agentbuilder.com/documentation
[34]	Y. Shoham, « Agent Oriented Programming », Artificial Intelligence, 60(1), 1993, p.51-92, North-Holland
[35]	Pierre-Michel Ricordel Thèse de doctorat : « Programmation Orientée Multi-Agents : Développement et Déploiement de Systèmes Multi-Agents Voyelles » octobre 2001
[36]	http://www.madkit.org
[37]	perso.univ-lr.fr/fcolle/cours/DIMA/.../Presentation_dima.pdf
[38]	J. Collis, D. Ndumu, S. Thompson. « ZEUS Methodology Documentation, Role Modelling Guide, Three case studies, Application Realisation Guide, and Runtime Guide » http://www.labs.bt.com/projects/agents/zeus
[39]	Maamoun Bernichi – these de doctorat « surveillance de logiciels à base de communauté d'agents mobiles » Novembre 2009