الجمهورية الجزائرية الديموقراطية الشعبية

**République Algérienne Démocratique et Populaire**

**Ministère de l'Enseignement Supérieur et de la Recherche Scientifique**

UNIVERSITE SAAD DAHLEB DE BLIDA

**FACULTE DE SCIENCES EXACTES**

**DEPARTEMENT D'INFORMATIQUE**

**Mémoire de projet de fin d'études**
**en vu d'obtention du diplôme**
**d'ingénieur d'état en informatique**

**Option :** Intelligence Artificielle

**Thème**

# INTERROGATION D'UNE BASE DE DONNÉES RÉPARTIES ET HÉTÉROGÈNES A L'AIDE D'AGENTS MOBILES

Proposé et suivi par :

M^r MENACER Djamel Edinne

Présenté par :

M^r STAMBOULI Ahmed
M^r TASSIS Ahmed

Membres du jury :

| | |
|---|---|
| M^me OUAHRANI | : Présidente. |
| M^me OUKID | : Examinatrice. |
| M^elle AOUSSATE | : Examinatrice. |

Année universitaire 2003-2004

# Remerciements


*Tous d`abord nous remercions* **الله عز وجل** *pour nous avoir guidé vers le bon chemin de la lumière et de savoir, pour nous avoir donné du courage et de la volonté afin de pouvoir réaliser ce mémoire.*

*Nous exprimons nos sincères remerciements à nos parents pour tout ce qu`ils ont pu nous apporter.*

*Nous tenons à remercier les membres du jury pour avoir eu l'obligeance d'accepter et d`apprécier notre travail.*

*Nous tenons à remercier notre promoteur M. Menacer Djamel Eddine pour son aide, sa patience, sa disponibilité, et sa compréhensibilité.*

*A tous les enseignants de la faculté des sciences exacts de Blida et surtout les enseignants du département de l'informatique.*

*Nous remercions de profond de cœur : Othmen, Hadj hacene, Hamza, Hichem, Abdellah elhadj, Bagdad, Redhouane, Mouadh, Bialal, Redha, Madani, Djamel, Omari, Abdelkader, Merouane, Mohamed, Zoubir, Hamouche, M'hamed .*

*Enfin, nous remercions, de tout cœur, tous ceux qui ont contribué de prés ou de loin à la réalisation de ce travail*

# Dédicace

*Ce travail est dédié :*

*A mes très chers **parents** pour leur soutien durant*

*toute ma carrière,*

*Pour leur bienveillance, leurs efforts constants dans mes études,*

*Pour leur encouragement,*

*A tous mes frères et à mes sœurs,*

*A toute la famille Stambouli et Kara Bernou,*

*A mon binôme **Tassis** et à toute sa famille,*

*A tous mes amis que j'aime et qui m'aiment.*

*Stambouli*

*Ce travail est dédié :*

*A mes très chers **parents** pour leur soutien durant*

*toute ma carrière,*

*Pour leur bienveillance, leurs efforts constants dans mes études,*

*Pour leur encouragement,*

*A toute la famille Tassis,*

*A mes frères **Mohamed, Merouane.***

*A mes sœurs,*

*A mon binôme **Stambouli** et à toute sa famille,*

*A tous mes amis que j'aime et qui m'aiment.*

*Tassis*

# TABLE DES MATIERS :

# LISTE DES FIGURES

## LISTE DES TABLEAUX

# *Glossaire*

**API : Application Programming Interface.**

**ASDK : Aglets Software Developement Kit.**

**BD : Base de Données.**

**BDOO : Base de Données Orienté Objet.**

**BDOR : Base de Données Objet Relationnel.**

**BDR : Base de Données Relationnel.**

**JDBC : Java DataBase Connectivity.**

**SQL : Structured Query Language.**

**UML : Unified Modeling Language.**

# Introduction générale

*Le gain de notre étude*
*c'est en être devenu*
*meilleur et plus sage.*
**Michel Eyquem.**

Les réseaux à grande échelle peuvent être considérés comme un ensemble des ordinateurs, qui lient une grande quantité de données distribuées, et fournissent d'une façon transparente l'accessibilité à tout genre d'ordinateur et de structures de données.

Une base de donnée répartie permet d'intégrer et de partager des données gérées par des calculateurs différents. Le partage de données réparties repose sur l'interconnexion des calculateurs par un réseau local ou étendu, et un système spécifique pour gérer les données réparties. En particulier, ce système doit fournir un haut degré d'indépendance par rapport à l'environnement réparti qui peut être très complexe.

La recherche qui a été menée dans le cadre de ce mémoire vise à étudier théoriquement et pratiquement un système dont l'objectif est d'interroger une base de données répartie et hétérogène dans un réseau local à l'aide d'agents mobiles.

## Motivations et objectifs :

Notre travail a précisément comme objectif d'étudier l'intérêt d'utiliser la technologie des agents mobiles dans le domaine des bases de données réparties pour développer un système d'interrogation d'une base de donnée répartie et hétérogène à base d'agents mobiles, qui permet la résolution des problèmes imposés par l'intégration des données et l'exécution des requêtes réparties. Les objectifs qui nous nous somment fixés, afin de réaliser ce travail, sont les suivants :

1. Étudier les agents et particulièrement les agents mobiles ainsi que les mécanismes permettant leur implémentation.

2. Présenter le domaine de base de donnée répartie.

3. Présenter les systèmes existant à base des agents mobiles.

4. Proposer une architecture d'un système qui rendre l'accès transparent au base de donnée répartie et hétérogène appelée RIAM.

5. Conception de système RIAM par UML afin de gérer les différentes connaissances.

6. Mise en œuvre de système RIAM.

## Méthodologie suivie :

Afin d'atteindre nos objectifs, nous avons débuté par un état de l'art des domaines des agents (technologie agents), agents mobiles et les bases de données réparties. En suite nous avons présenté des systèmes a base d'agents mobiles pour montrer que les agents mobiles peuvent être utilisés comme un outil pour rendre l'exécution d'une requête répartie autonome. Puis nous avons proposé Une architecture RIAM qui permet d'interrogé une base de donnée répartie et hétérogène avec des agents statiques et des agents mobiles suivis par une conception détaillée. Finalement, nous avons exposé la possibilité de mettre en œuvre le modèle proposé en utilisant Java comme langage de programmation.

## Présentation de mémoire :

La suite du mémoire sera constituée de six chapitres.

Chapitre 1 : nous présentons la technologie agent.

Chapitre 2 : nous présentons les agents mobiles ainsi que les concepts associés de leur implémentation.

Chapitre 3 : Est consacré pour les concepts des bases de données réparties, ainsi leurs objectives et leurs fonctionnalités et les solutions aux problèmes techniques posés par leur mise en œuvre.

Chapitre 4 : Présentation de la problématique et proposition d'une solution, ainsi que une présentation des systèmes existant à base d'agents mobiles.

Chapitre 5 : Décrit la modélisation de système RIAM. En décrivant de manière détaillée les différentes entités du système multiagents.

Finalement, le chapitre 6 présente une proposition d'une implémentation possible d'une application RIAM. Ce chapitre montre, en utilisant le langage de programmation Java, la possibilité de mettre en œuvre l'architecture proposée.

# CHAPITRE I.
# La technologie Agent

## 1. Introduction :

Le concept d'agent est le résultat de recherche dans le domaine d'Intelligence Artificielle (IA), afin de simuler quelques capacités de l'être humain comme l'intelligence, l'apprentissage, la perception…etc. En même temps, l'évolution des applications et l'émergence de la notion d'intégration et distribution a conduit les chercheurs de penser à distribuer l'intelligence sur plusieurs entités, afin de combler les limites de I.A pour résoudre des problèmes complexes[15], c'est le passage de IA à l'Intelligence Artificielle Distribution (IAD). Cette dernière a introduit le concept de Système Multi-Agent (SMA) qui porte sur le modèle de l'agent.

Les SMA mettent en œuvre un ensemble d'agents qui interagissant entre eux. Chaque agent possède un ensemble de caractéristiques qui permettant de le défini. Un agent simple possède au moins les propriétés suivantes : l'autonomie, la réactivité, la pro-activité et la sociabilité.

## *2.* Qu'est-ce qu'un agent ?

D'un point de vue étymologique, le mot **agent** vient du latin "agere" qui signifie agir. Littéralement, l'agent est donc celui qui agit [15]. Et d'un point de vue informatique, il existe à l'heure actuelle, encore plusieurs définitions de ce qu'est un agent, Nous nous appuierons, sur une définition, telle que celle donnée par ( J.Ferber) [1] :

« Un agent peut être défini comme une entité (physique ou abstraite) capable d'agir sur elle-même et sur son environnement, disposant d'une représentation partielle de cet environnement, pouvant communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de ses connaissances et de ses interactions avec les autres agents. Les agents ont deux tendances : une tendance sociale tournée vers la collectivité ( les mécanismes et connaissance associés concernant les

activités du groupe) et une tendance individuelle avec les mécanismes contenant les règles de fonctionnement interne de l'agent.

On peut caractériser un agent par son rôle, ses spécialités, ses objectifs, ses fonctionnalités, ses croyances, ses capacités décisionnelles, ses capacités de communication et éventuellement ses capacités d'apprentissage ».

## 3. Déterminant d'un agent :

On appelle déterminant d'un agent l'ensemble nécessaire et suffisant de ses caractéristiques structurelles, environnementales et comportementales qui permet d'expliquer sa façon d'agir.

Les caractéristiques d'environnement d'un agent liées à la représentation que se fait l'agent de son environnement et de lui-même. Les caractéristiques structurelles d'un agent déterminant l'ensemble de ses composants, alors que les caractéristiques comportementales contraignent l'ensemble de ses comportements en accord avec les caractéristiques environnementales. Par exemple, un avion est un agent dont les caractéristiques structurelles sont la capacité de voler, la vitesse… etc. Les caractéristiques comportementales sont le pilote et le plan de vol et sa caractéristique environnementale est sa position.[50]

## 4. Caractéristiques d'un agent:

Pour qualifier qu'une entité (physique ou abstraite) est un agent/ agent intelligent ou non, il faut que cet agent posséder certaines propriétés, qu'ils sont [2] :

**Autonomie :**

Un agent autonome est celui qui peut prendre des décisions en se basant sur ces propres critères sans l'intervention d'aucun élément extérieur (un autre agent, opérateur humain), et même sans stimulation environnementale.

**Sociabilité :**

Afin d'accomplir et poursuivre les tâches pour lesquelles il est conçu, l'agent communique avec les autres agents ( et possiblement avec des humains) pour échanger des informations et des connaissances, cela grâce à un langage de communication.[8]

**Pro-activité :**

Un agent proactif est un agent qui peut prendre des initiatives au bon moment et capable d'agir sans même que son environnement ait changé. Un tel comportement est fortement souhaitable pour certains agents.

**Réactivité :**

L'agent dans certaine circonstance est entouré par un environnement qui change son état, dans ce cas si l'agent possède la capacité de réagir aux changements de leur environnement, on dit qu'il est réactif. En effet, un agent ayant un comportement

purement réactif, qui ne ferait que répondre aux changements de l'environnement, ne serait pas nécessairement en mesure d'atteindre ses buts.[8]

Les quatre propriétés citées ci-dessus soient communes à toute la communauté de chercheurs en Intelligence Artificielle [14]. Mais Il y a d'autres propriétés qui peuvent être rajoutées. On citant :

**Environnement :**

L'agent autonome est entouré par un ensemble des conditions extérieures, l'ensemble de ces conditions représente pour l'agent un environnement. Ce dernier permet l'évolution contenue pour un agent sachant qu'il peut existe d'autres agents dans le même environnement.

**Comportement adaptatif :**

Un agent adaptatif peut changer non seulement son comportement mais aussi ces objectifs selon ses interactions avec son milieu (environnement, autre agent).

**Intelligence :**

Un agent est intelligent lorsqu'il peut prendre des décisions d'une façon autonome basée sur l'interprétation des événements observés, dans ce cas l'agent exploite ses capacités de représentation symbolique et de raisonnement pour automatiser des tâches à la place de l'utilisateur.

**Perception :**

Un agent possède la capacité de perception qui permet d'acquérir des informations sur son environnement et sur lui-même.

**Intentionnalité :**

Une intention est la déclaration explicite des buts et des moyens d'y parvenir. Pour un agent, Elle exprime la volonté d'atteindre ses buts ou d'effectuer des actions. Un agent intentionnel est un agent guidé par ces buts.[50]

**Mobilité :**

Un agent est mobile s'il possède la capacité de déplacer d'une machine à une autre et au travers de différentes architectures et plates-formes, pour accomplir des tâches sans que l'utilisateur ait le moindre contrôle sur celles-ci.

## 5. Les différentes classes d'agents:

Deux grandes Classes d'agents peuvent être distinguées en fonction de la complexité du mécanisme de raisonnement utilisé. On dira que plus un agent est complexe, plus son comportement est difficile à calculer et plus il est coûteux en ressources (temps, volume, énergie...).[10]

### 5.1 Agents cognitifs :

Les agents cognitifs sont le résultat direct des recherches menées dans le domaine de l'intelligence artificielle. Ce sont des agents qui possèdent une représentation explicite de leur environnement et des autres agents. Ils savent tenir compte de leur passé et fonctionnent selon un mode social d'organisation.

### 5.2 Agents réactifs :

Un agent réactif n'a pas de représentation explicite de son environnement et ne peut tenir compte de son passé. Son mode de fonctionnement est simple de type 'stimulus/réponse', c'est à dire simple réaction à l'environnement.

Nous résumons les différences entre les deux classes, Dans le tableau I-1.

| **Caractéristique d'un système d'agents Cognitifs** | **Caractéristique d'un système d'agents Réactifs** |
|---|---|
| Présentation explicite de l'environnement | Pas de représentation |
| Peut tenir compte de son passé | Pas de mémoire de son historique |
| Agents complexes | Fonctionnement stimulus/réponse |
| Petit nombre d'agents | Grand nombre d'agent |

**Tableau I-1: Comparaison entre agents cognitifs et agents réactifs.**

Dans certaines circonstances, un agent peut présenter une structure mixte avec des aspects cognitifs mais aussi des réflexes purement réactifs à certains stimulation. Dans ce cas, les agents sont dits hybrides.[7]

## 6. Architecture d'un agent :

Nous présentons une architecture générale d'agent cognitif (figure I-1) [50]:

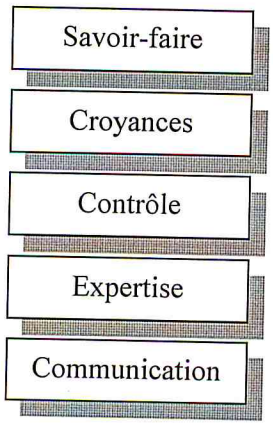



**Figure I-1: Structure interne d'un agent cognitif.**

### Savoir-faire :

Le savoir-faire est une interface qui permet la sélection des agents à solliciter pour atteindre ses objectifs, et aussi permettant la déclaration des connaissances et des compétences de l'agent.[50]

**Croyances :**

Les croyances d'un agent constituent les connaissances que l'agent possède sur lui-même, sur les d'autres agents et même sur leur environnement. Ces connaissances ne sont pas nécessairement objectives ou vrais, peuvent être incorrectes ou incomplètes. Les logiques de connaissances et de croyance s'intéressent à la formalisation de quelques connaissances considérées comme incertaines. Cette formalisation détermine une grande partie de comportement intelligent d'un agent. [50]

**Contrôle :**

Afin d'atteindre ses objectifs, l'agent contrôle tout changement dans son comportement et leur environnement. Les buts, les intentions, les plans et les taches qu'il possède l'agent, représentent leur connaissance de contrôle. [50]

**Expertise :**

C'est une connaissance qui permette à l'agent de résoudre un problème. Par exemple, dans les systèmes experts les règles de production représentent ce type de connaissance. [50]

**Communication :**

L'interaction entres agents est possible grâce à un protocole de communication pour assurer une bonne coopération et coordination d'action. En plus, l'agent peut utiliser d'autre connaissance de communication liée au réseau de communication ( le cas où l'agent n'est pas en liaison directe avec d'autre agent). [50]

## 7. Fonctionnement :

Dans cette sous-section, on va présenter un aperçu sur le fonctionnement d'un agent cognitif. Le modèle qui représente les processus missent en œuvre lors de fonctionnement d'un agent est illustré par la figure I-2. les agents interagissent avec leurs environnements, le comportement d'un agent doit supporte les trois fonctionnalités principales : Percevoir, décider et agir. En outre des fonctions principales, en trouve des sous-fonctions dont les plus importantes sont la détection de conflits, la révision des croyances, la coopération (négociation, coordination), l'apprentissage. [50]

Un agent peut percevoir son environnement et extraire des informations. Il peut aussi interagir avec d'autres agents (communiquer, négocier). Il doit aussi élaborer un plan d'action en fonction des connaissances et des croyances dont il dispose et des buts qu'il se fixe suite à une perception ou une interaction avec le monde extérieure.

Donc, un agent doit citer des buts à retenir et à satisfaire en premier, ensuite élaborer un plan en fonction de ces buts et passer à l'exécution.

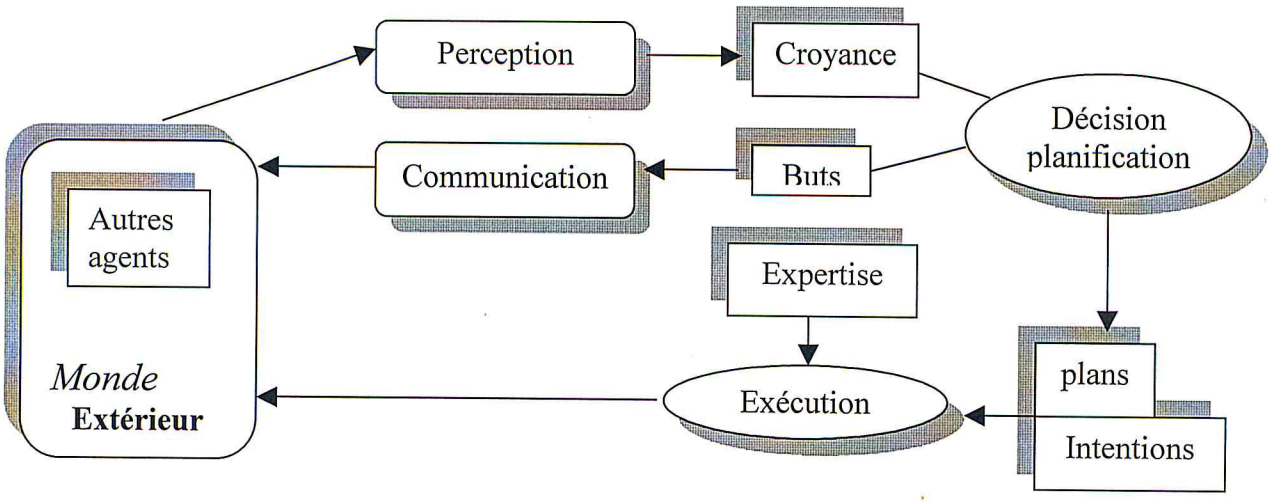



**Figure I- 2: Fonctionnement d'un agent (architecture BDI).**

Une architecture BDI (de l'anglais Belief, Desire, Intention pour croyance, désir et intention) est développée par les chercheurs dans le cadre du raisonnement pratique (tel qu'il se pratique chez les humains) qui orienté vers la prise en compte des états mentaux. [50]

**Perception :**

Les origines des connaissances d'un agent sont multiples :

- Le savoir initial de l'agent.
- La perception de soi (perception proprioceptive) et du monde (extéroceptive).
- La communication avec les autres agents.

Parmi ces connaissances, il y'à ceux qui sont considérées comme des connaissances certaines puisqu'elles n'ont subi aucune mise à jour. Les connaissances considérées comme certaines sont celles qui sont issues de la perception et du savoir initial. Et il y'à ceux qui sont considérées comme incertaines puisqu'elles évoluent sans que l'agent en soie forcément informé.

**Prise de décision :**

A travers ses observations et ses interactions avec le monde, l'agent fixe un certain nombre de buts durant son exécution. L'agent doit donc sélectionner le but à satisfaire en premier et pour chaque but, ce qu'il doit fait pour l'atteindre.

Lorsque l'agent se trouve en face à de telles situations, il analyse les différentes alternatives en termes d'utilité (quel avantage l'agent pourrait-il en tirer ?) et d'incertitude (quelle chance a l'action d'être effectuer en fournissant le résultat attendu ?). Plusieurs techniques de résolutions de conflits existent parmi les : l'utilisation de la carte cognitive (cognitive map) qui est un réseau qualitatif exprimé en terme d'influence (positives ou négatives) et reliant les buts et leurs utilités. Le but qui représente le plus d'influences positives sera le but à retenir.

**Planification :**

Contrairement au IA classique, l'IAD offre des possibilités de négociation autorisant une gestion locale des conflits et une planification dynamique. Par contre, l'IA classique repose sur l'hypothèse d'un univers classique, ce qui est incompatible avec l'approche MultiAgents. En effet, du fait de l'intervention d'autres agents, un plan peut être remis en cause. Pour cela l'agent doit altérer une planification et exécution et réviser des parties de son plan. Dans un SMA, la planification est distribuée : Chaque agent construit son propre plan en coordination avec d'autre (cas d'agent coopératif), il n'existe pas un plan global. Il existe des systèmes d'IAD présentent une planification centralisée, un organe central se chargera de la gestion des conflits et de l'élaboration d'un plan global.

## 8. L'environnement d'agent :

Le monde dans lequel les agents évoluent est appelé « environnement ». Les environnements peuvent être de nature très différent allant du plus simple au plus complexe. A cet égard, il existe différentes propriétés qui permettent de caractériser l'environnement d'un agent [9] :

### 8.1 Accessibilité :

L'environnement est dit accessible par un agent, Si les organes sensitifs de cet agent peuvent accéder à l'état complet de l'environnement. Pour cela, l'environnement fournit tous les aspects importants au processus de choix des actions d'un agent. Dans le cas contraire, on parlera alors d'un environnement inaccessible.

### 8.2 Déterminisme :

Dans un environnement déterministe, le prochain état de l'environnement est déterminé par l'état courant et par les actions effectuées par les agents. A l'opposé, on parle d'un environnement non déterministe.

### 8.3 Dynamique vs Statique :

Si l'environnement de l'agent évolue pendant qu'il décide quelle tâche à accomplir, alors on dit que l'environnement est dynamique pour cet agent, sinon on parle d'un environnement statique. Les environnements statiques évitent que l'agent à chaque sélection d'une action, contrôlé l'environnement.

### 8.4 Discret vs continu :

Lorsque le nombre de perceptions et d'actions est limité et clairement défini, on dira que l'environnement est discret. Par opposition, on dira que l'environnement est continu.

## 9. Les Systèmes multiagents (SMA) :

Les systèmes multiagents ont été développés dans le cadre de l'intelligence artificielle distribuée. L'intérêt qu'ils suscitent est lié à leur capacité d'aborder les

problèmes complexes d'une manière distribuée. Un système multiagents est un système qui s'appui sur le principe suivant : « Au lieu d'avoir un seul agent en charge de l'intégralité d'un problème, on considère plusieurs agents qui n'ont chacun en charge qu'une partie de ce problème. La solution au problème initial est alors obtenue au travers de l'ensemble des comportements individuels et des interactions ». [5]

Un SMA est généralement caractérisé par [6]:

– Chaque agent a des informations ou des capacités de résolution de problèmes limitées, ainsi chaque agent a un point de vue partiel.
– Il n'y a aucun contrôle global du système multiagent.
– Les donnés sont décentralisées.
– Le calcul est asynchrone.

### 9.1 Interaction et coopération entre agents :

L'agent assure des protocoles de communication et d'interaction en se basant sur une structure bien définie par l'agent lui-même. Ce dernier progresse dans un environnement qui est ouvert et contient des agents autonomes et distribués agir soit pour leur intérêt personnel, soit en coopération avec les autres agents de l'environnement.

#### 9.1.1 Coordination :

Afin d'avoir un système multiagent plus cohérent, les agents coordonner leur action pour accomplir une activité dans un environnement partagé. L'un de forme de coordination est la coopération, où la coordination se fait entre des agents non antagonistes, l'inverse on trouve la négociation qui correspond à la coordination en univers compétitif.

Le degré de coordination peut être envisagé comme la quantité d'activité supplémentaire nécessaire pour éviter les deadlocks, Les activités redondantes... etc.

#### 9.1.2 Négociation :

La négociation est l'un des formes d'interaction entre les agents. Elle Peut se faire entre deux agents ou plus, Afin de prendre une décision commune telle que chaque agent atteindre ses buts. Pour cela, il faut un langage de communication, un protocole de négociation et un processus de décision par lequel l'agent décide sa position les concessions, les critères pour un accord … etc. Et on distingue trois types de négociation [12]:

- Négociation un à un.
- Négociation un à plusieurs.
- Négociation plusieurs à plusieurs.

### 9.2 L'hétérogénéité des agents :

L'hétérogénéité dans un système multiagent est définie par la présence de différences entre les agents. Il est très rare que deux agents soient identiques en tous points. On peut trouver avec un système contient des agents cognitifs avec des agents

réactifs. Même si on prend deux agents de même nature, ils peuvent être conçus de façon très dissemblable.

### 9.3 Communication entre les agents :

Les agents communiquent entre eux à l'aide d'un langage et ils utilisent des protocoles de communication. On distingue essentiellement deux types [10]:

#### 9.3.1 Communication par partage d'information :

La communication par partage d'information se fait lorsqu'un ensemble d'agents partage la même structure de donnée, et travaillent en mode coopératif afin de résoudre un problème global de nature centralisée selon tous les agents. Cette structure de donnée contient initialement les données du problème et est enrichie au cours de la résolution jusqu'à l'obtention de la solution.

Ce type de communication est souvent désigné par le modèle du tableau noir (figure I-3).

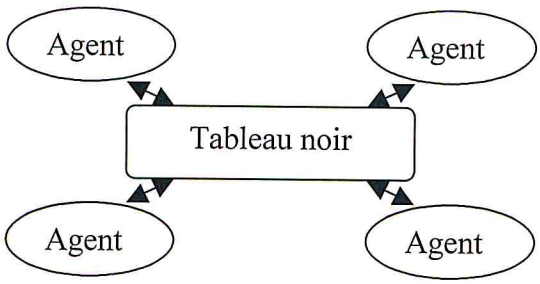



**Figure I-3 : Structure d'un système à base de tableau noir.**

Les agents déposent et lisent une information sur une zone de données commune.

#### 9.3.2 Communication par envoi de massages :

La communication par messages est un outil pour coordonner les actions d'agents. Elle est caractérisée par la distribution totale à la fois des connaissances, des résultats partiels et des méthodes utilisées pour aboutir à un résultat dans un SMA (Figure I-4). Pour que l'agent communiqué, il doit pouvoir envoyer et recevoir des messages.

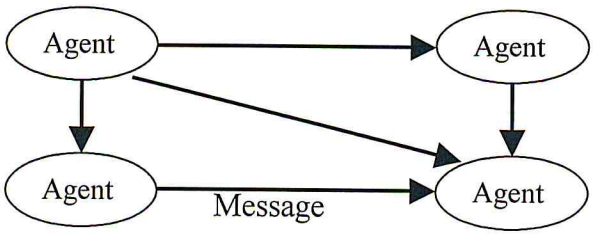



**Figure I-4: Structure d'un système à envoi de messages.**

La communication peut se faire point à point ou par diffusion. Dans ce contexte, l'agent peut avoir les fonctions de maître, esclave. Et les rôles passif, actif (les deux).Un agent Pour tenir un rôle passif, doit être capable de répondre, i.e. accepter une question

externe et envoyer la réponse à la source (assertion). Et Pour tenir un rôle actif, un agent doit pouvoir poser une question et faire des assertions. Il peut ainsi contrôler un autre agent par le biais de question/réponses. Dans un fonctionnement de pair, l'agent a les deux rôles.

### 9.3.3 Protocole de communication :

On définit Les protocoles de communication avec plusieurs niveaux (figure I-5) . Le plus bas définit les méthodes d'interconnexion, celui d'intermédiaire, le format, la syntaxe ou le type de transfert ; au plus haut, on trouve la compréhension et la sémantique, cette dernière faisant référence ( entre autres ) au type du message. Un protocole peut être binaire ou n-aire, on peut le définir avec une structure de donnée à 5 champs : L'émetteur, le ou les récepteur(s), le langage du protocole, l'encodage et le décodage des informations, les actions à entreprendre par le ou les récepteur(s).

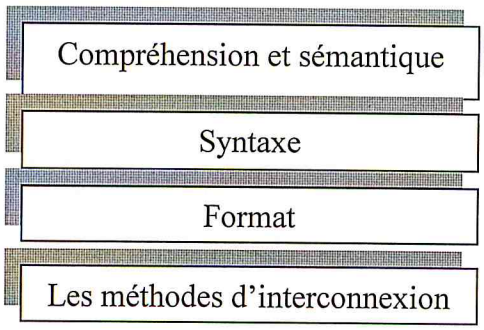



**Figure I-5: Protocole de communication en couche**

### 9.3.4 Actes de langage :

La théorie des actes de langage est un cadre d'analyse des échanges langagiers entre humains. Elle considère la communication comme des actions de requête, suggestion, engagement, réponse, etc.

Dans un cadre de système multiagents, les actes de langage est une façon de définir le type des messages et de contraindre la sémantique de la communication.

### 9.3.5 langages de communication KQML[4] :

**KQML** est un langage destiné à échanger des informations et des connaissances entre agents. Ce langage fournit également la base pour établir des interactions de haut niveau comme la négociation entre agents. Etant proche du langage naturel, Son principal atout est que tout ce qui est nécessaire à la compréhension du message est inclus dans le message lui-même.

La forme de base du protocole est :

(**KQML**-performative
: **sender** <word>
: **receiver** <word>
: **language** <word>
: **ontology** <word>
: **content** <expression>…)

Les performatifs KQML sont modélisés selon les performatifs des actes de langages. Leur sémantique est indépendante du domaine alors que celle du message l'est et est définie à l'aide des champs : :content (le message ), :language (language d'expression du message), : ontology (le vocabulaire du domaine). Les autres arguments d'un message sont : :sender , :receiver, :reply-with, :in-reply-to.

## 10. Langages de programmation des agents :

Plusieurs langages destinés à la conception d'agents ont été développés. Parmi les langages existants nous pouvons citer entre autres[3]:

**TCL | TK** ( Tool Command Language | ToolKit )

C'est un langage Conçu par John K.Ousterhout, de l'Université de Berkeley vers les années 88, TCL est un langage sous forme d'une librairie C. TK est un ensemble d'outils conçus spécifiquement pour développer des applications sous X-window. TK est l'extension de TCL pour la gestion graphique. TCL|TK fournit un environnement de programmation Simple, mais puissant pour développer des applications dans cet environnement graphique.

**JAVA**

Java est un langage de programmation orienté objet, crée par la société Sun Microsystems. Le principal objectif de Java est d'augmenté la productivité des programmeurs. Pour cela, il permet :

- Le "*Write once, run everywhere*","*écrire une fois, utiliser partout*"; pour ne pas être limité au développement pour une seule plate forme (un programme écrit en java tournera de la même façon sur MS Windows ou sur Unix).

- De plus, il y a de nombreuses fonctionnalités liées au monde des réseaux.

- La réutilisation et la maintenance des programmes.
- Pour finir, le minimum nécessaire pour développer une application en langage Java est disponible gratuitement sur le site de Sun Microsystems.

**LISP**

Les langages issus de la programmation logique sont en grande partie adaptés aux nouveaux challenges de la conception d'applications complexes et distribuées, telles que celles de l'intelligence artificielle sur le Web. LISP est l'un des Principaux langages utilisés en intelligence artificielle. C'est un langage fonctionnel très puissant et flexible. LISP sera utile pour implémenter les mécanismes de déduction de l'agent.

**PROLOG**

PROLOG est un langage interprété, de haut niveau, disposant d'une représentation uniforme des données et des programmes. Par ajout de prédicat de communication, les programmes peuvent alors non seulement s'échange du savoir mais aussi du savoir-faire.

Dans le cadre des SMA, et pour permettre au langage prolog de mieux décerner la technologie agents, il existe des passerelles entre Java et ce dernier, soit comme BinProlog (http://clement.info.umoncton.ca/BinProlog) au moyen de dispositif d'interaction de haut niveau entre Java et Prolog, soit par des compilateurs de Prolog en Byte-code exécutable sur une machine virtuelle Java comme Jprolog.

**C/C++**

Ces deux langages restent toujours performants pour la réalisation de certaines fonctionnalités de l'agent. Le langage C++ a été crée en 1972 par Denis Ritchie. Dans le but d'écrire le système d'exploitation (UNIX).

Son succès international a amené l'ANSI (American National Standard Institute ) à définir un ''standard ''(on le qualifie souvent de ''C ANSI'').

Le Langage C présente les caractéristiques suivantes :

- Le suivi des règles de la programmation structurée.
- Un langage orienté système. C'est à dire qu'il est le langage le plus proche du '' Raisonnement'' d'un ordinateur, après l'assembleur.
- Le langage C est un langage compilé. C'est à dire que la totalité du code source est traduite en langage machine, avant l'exécution.
- Le langage C est connu d'un grand nombre de développeurs.

Le langage C++ ce caractérise par :

- Langage orienté objet issu du C.
- Supporte l'abstraction de données.
- Efficace, si le développeur arrive à se libérer du processus mental de la conception modulaire et commencer à penser en termes de classes et d'objets.
- Permet la réutilisation des programmes écrits en C.

**PHP**

PHP est un langage script s'exécutant sur le serveur. Comparable dans ces grandes lignes au JavaScript serveur. Il permet notamment d'interagir entre des SGBDR situés sur un serveur et un poste client équipé d'un navigateur. Cela lui donne de grandes capacités pour la création d'agents en l'associant à d'autres langages tels que le C ou PROLOG.

**PERL**

PERL est un langage interprété optimisé pour parcourir des fichiers textes, en extraire de l'information et imprimer des rapports basés sur cette information. C'est un langage pratique proche du C. PERL peut aussi traiter des fichiers binaires. Ce Langage sera typiquement utilisé pour des outils de recherche.

**AGENT0**

AGENT0 est un langage de programmation orientée agent proposé par Yoav Shoham en 1993 comme un nouveau paradigme de programmation. Les agents sont les éléments centraux du langage, de la même façon que les objets sont centraux pour les langages orientés objets. Ce langage est fondé sur un langage formel utilisant la logique modale pour décrire les états mentaux, les agents sont programmés en termes de règles d'échange mental. Ce langage a trois composantes[44] :

- Un système logique pour définir les états mentaux de l'agent.
- Un langage de programmation pour programmer des agents.
- Un processus pour compiler les programmes agents en un système exécutable.

## 11. Sécurité :

La sécurité est un élément central dès que l'on parle de l'informatique. Les systèmes d'informations sont menacés par des virus, des pirates…etc. Les agents intelligents peuvent générer des failles de sécurité (destruction d'informations, espionnage, contournement des protections…etc.) dont les causes sont les suivantes [11] :

### 11.1. La délégation :

L'agent agir et naviguer en compte de son utilisateur, ce que fait que l'agent est délègue par une partie de l'autorité de son utilisateur. Le problème est que l'agent n'a pas « d'états d'âme ». Dans certain cas, si nous lui demandons d'aller chercher des informations auxquelles nous n'aurions pas accès en temps normal, il fera tout pour les obtenir.

### 11.2. La confiance :

Une des motivations principale de l'utilisation des Agents est d'éviter à l'utilisateur de faire un travail répétitif, mais un humain fait attention aux effets de bord et aux résultats qu'il obtient, contrairement à un Agent.

### 11.3. La mobilité :

Dans un réseau, un agent peut se déplacer de façon autonome entre les machines, ce qui rendre difficile de suivie et de contrôlé les actions d'un grand nombre d'agents.

### 11.4. Les vers :

« Les vers sont des programmes autonomes capables de se répliquer en réseau et de se lancer à distance ». Or cette définition ressemble beaucoup à celle des agents mobiles. Ce qui rend l'opération d'identification des agents importante pour reconnaître un vers d'un Agent .

## 12. Domaine d'applications des agents intelligents :

Les agents intelligents sont largement utilisés dans différent domaine, en les trouve dans la bureautique, Internet, intranet…etc. Nous donnons ci dessous un aperçu des applications typiques à base d'agent selon le domaine. [13]

### 12.1. Bureautiques :

***Les agents système :*** « Ce sont des agents d'interface qui fournissent une aide à l'utilisateur dans l'utilisation du système d'exploitation ».

Comme des exemples d'agents système, nous trouvons sous MacOS l'agent d'apprentissage *Open Sesame !* de Charles River Analytics a observé depuis 1993 les actions d'un utilisateur qualifié afin de mettre en évidence les actions répétitives pour pouvoir ensuite les automatiser. Sous Windows 95, l'agent système tourne en tâche de fond pour exécuter un certain nombre de programmes tels que le contrôle d'erreur ou la défragmentation du disque à des heurs spécifiques par l'utilisateur.

***Les agents d'application :*** « Ce sont des agents d'interface qui fournissent une aide à l'utilisateur pour se servir correctement d'une application particulière ».

À titre d'exemple, on trouve le logiciel *Answer Wizard* dans les applications Microsoft telle que *PowerPoint,* ce logiciel fournit une aide aux requêtes de l'utilisateur selon le contexte. L'interaction s'effectue au travers un écran où l'utilisateur soumet une requête et où le logiciel donne une réponse. Et on trouve aussi *InfoTiker* de Erik Thomas Mueller est un applet Java exécuter en tâche de fond qui extrait des informations, sur le Web dont les URL sont spécifiés par l'utilisateur. *InfoTaker* interroge ces sites Web et résume les résultats obtenus pour l'utilisateur.

***Les agents de suite logiciel :*** « Ce sont des agents d'interface qui fournissent une aide à l'utilisateur pour faciliter son travail avec application corrélée ».

### 12.2. Internet :

***Les agents de recherche du Web :*** « Ce sont des agents Internet qui fournissent des services de recherche dans le Web à un utilisateur ».

***Les agents de serveur du Web :*** « Ce sont des agents Internet qui résident sur un site Web spécifique pour fournir des services ».

***Les agents de filtrage d'information :*** « Ce sont des agents Internet qui filtrent des informations selon des critères spécifiques par l 'utilisateur ».

Il existe plusieurs applications dans ce domaine telle que *NewHound* de San Jose Mercury, ce dernier recherche automatiquement des articles et des annonces qui correspondent au profil d'un utilisateur et envoie ses résultats par courrier électronique. Les utilisateurs peuvent toujours ajuster les paramètres de la recherche pour obtenir plus au moins d'information sur un sujet donné. Ils peuvent également ajouter des mots-clés pour améliorer la pertinence des informations filtrées. Ces mots-clés permettent d'inclure ou exclure une information. *InfoSage* d'IBM fournit une information personnalisée pour

les professionnels permettant de sélectionner, en temps réel, des données sur la concurrence, des profils des sociétés ou des informations commerciales.

***Les agents de recherche documentaire :*** « Ce sont des agents Internet qui retournent un ensemble personnalisé d'information correspondant à la demande de l'utilisateur ».

***Les agents de notification :*** « Ce sont des agents Internet qui indiquent à un utilisateur des événements qui pourraient l'intéresser ».

*URL-minder* de NetMind est un exemple de ce type d'agent qui conserve une trace de contenu HTML d'un URL particulier afin de prévenir l'utilisateur des modifications de son contenu, au moyen d'un service de courrier électronique.

***Les agents de service :*** « Ce sont des agents Internet qui fournissent des services spécialisés à des utilisateurs ».

***Les agents mobiles*** **:** « Ce sont des agents Internet qui se déplacent d'un lieu à un autre afin d'exécuter des tâches spécifiques d'un utilisateur ».

Les agents *Telescript* de General Magic sont des exemples d'agents mobiles. Les agents *Telescript* intègrent les instructions de l'utilisateur concernant une tâche spécifique et se déplacent aux endroits nécessaires à l'exécution de leur tâche.

### 12.3. Intranet :

***Les agents d'automatisation :*** « Ce sont des agents qui automatisent les tâches d'une entreprise ».

***Les agents de base de données :*** « Ce sont des agents intranet qui fournissent des services agents à l'utilisateur de base de donnée ».

Un exemple de ce type d'agent, l'agent *DSS* de MicrioStrategy facilite le développement des applications OLAP de filtrage des informations de la base de donnée de l'entreprise et de restitution de l'information désirée.

***Les agents courtiers de ressources :*** « Ce sont des agents intranet qui réalisent l'allocation de ressources dans l'architecture client/serveur ».

## 13. Conclusion :

La progression des paradigmes de programmation a conduit aux agents et systèmes multiagents, cette novelle percée ait permis de répondre à certains besoins qui était jusqu'à une date récente accessible que par des humains. Les applications des agents sont multiples. On peut distinguer rapidement l'assistance à l'utilisateur, la recherche d'informations, activités commerciales, loisir et jeux ...etc. Ce domaine demeure aujourd'hui un domaine rempli de défis à surmonter autrement dit un domaine très ouvert pour la recherche.

Le chapitre suivant détaille d'une manière approfondie un des domaines les plus intéressants des agents : les agents mobiles, ses caractéristiques, les concepts nécessaires pour les implémenter et les systèmes à base d'agents mobiles existants.

# CHAPITRE II.
# Les agents mobiles

## 1. Introduction :

La notion d'agent mobile a été émergée en 1995 grâce aux travaux de Jim White et ses collèges à General Magic [24], l'émergence des agents mobiles dotent le monde des agents avec une caractéristique très intéressante qu'est la mobilité ou la capacité de se déplacer d'une machine vers une autre.

L'importance de la mobilité augmente jour après jour avec la forte utilisation des réseaux et de l'Internet. L'émergence de langage de programmation Java rendre la mobilité d'un côté technique facile à établie. Les agents mobiles joueront un rôle fondamental dans l'avenir de l'informatique et notamment dans l'avenir de l'Internet.

## 2. Définition d'un agent mobile :

Un agent mobile est un agent logiciel capable de se déplacer d'une machine vers une autre à travers un réseau hétérogène sous son propre contrôle pour être exécuter à distance, les agents mobiles transportent, à la fois, les données et le programme agissant sur les données, et éventuellement revenir à son site natal, une fois sa tâche accomplie.[13]

## 3. Caractéristiques d'un agent mobile :

Un système d'agent mobile doit supporter les caractéristiques suivantes :

***a)La mobilité :*** la mobilité d'un agent est les mécanismes utilisés pour transporter le entre les différents machines à travers le réseau (qui est pour être le World Wide Web). Il décide lui-même de manière autonome, de ses déplacements[17]. La fonction qui contrôle le déplacement d'un agent mobile d'une machine vers une autre s'appelle ***migration***.

Il existe deux types de migration :

- Migration forte : après la migration, l'agent mobile continu son exécution la où il à été interrompu (transmission de l'état d'exécution et les données d'un agent ).

- Migration faible : après la migration, l'agent mobile reprendre son exécution dès le début (transmission que les données d'un agent ).

***b)La communication :***

Dans un système à base d'agent mobile, l'existence d'un modèle de communication est nécessaire pour que les agents puissent communiquer entre eux, avec des serveurs ou bien avec des humains. En communiquant, les agents peuvent échanger des informations et coordonner leurs activités.

On distingue deux types de communication :

- Communication locale : les agents communiquer entre eux sur le même site.
- Communication à distance : les agents situés sur différents sites et communiquent entre eux.

L'implémentation de la communication inter-agent est réalisée à l'aide des langages spécifiques par exemple : KQML*( Knowledge Query and Manipulation Langage )*, KIF *( Knowledge Interface Format )*, FIPA-ACL*( Agent Communication Langage )*, SmallTalkAgents, AgentTcl*( Tool Command Langage )* [16].

## 4. Comportement d'un agent mobile :

Le comportement d'un agent mobile est composé d'un code correspondant à un algorithme, ainsi que d'un contexte incluant de données. Ce contexte peut évoluer en cours d'exécution, par exemple en collectant des données lorsqu'un agent réalise une recherche d'information sur un ensemble de serveurs. Le code et le contexte de l'agent sont déplacés avec l'agent lorsque celui-ci visite différente serveurs.

La plupart des systèmes à agents mobiles implantent une migration faible, c'est à dire une fonction de migration où l'agent redémarre son exécution depuis le début. En conséquence, le programmeur doit inclure dans le contexte de l'agent des informations sur l'état de l'exécution, et lorsque l'agent redémarre sur un site, le code de l'agent doit vérifier l'état de l'exécution et se brancher sur la partie de l'algorithme devant être exécutée sur ce site.

Des primitives de communication sont nécessaires pour permettre aux agents d'interagir, mais aussi aux agents d'interagir avec les serveurs qu'ils visitent. Ces primitives de communication prennent la forme d'envois de messages ou d'appels de procédures ou de méthodes. [22]

Etant donné l'hétérogénéité des réseaux, il serait préférable que le code de l'agent soit écrit dans un langage interprété (Java, Tcl et Telescript de GMI ).

## 5. La migration d'un agent mobile :

### 5.1. Processus et contexte d'exécution :

La différence entre un agent et un processus n'est pas de nature technique mais de nature conceptuelle, pour cela on va considérer les agents mobiles comme des processus et on va étudier la migration de processus.

Le *processus* est un concept clé de tous les systèmes d'exploitation. La notion de processus fournit un modèle pour représenter l'activité résultant de l'exécution d'un programme sur une machine. [25]

Le contexte d'exécution d'un processus, appelé aussi état d'exécution varie d'un système à un autre. En général, c'est l'ensemble des informations que les actions du processus peuvent consulter ou modifier. Il comprend des informations relatives à la gestion des processus, à la gestion de la mémoire associée au processus ou à la gestion des fichiers manipulés par le processus [26]:

- **Gestion des processus :** la gestion de processus est l'ensemble des informations relatives aux propriétés du processus telles que l'identificateur de processus, la date de son lancement, son état (prêt, élu ou bloqué), le compteur de programme indiquant la prochaine instruction à exécuter et le pointeur de pile indiquant le sommet de la pile associée au processus.

- **Gestion de la mémoire :** Les informations relatives à la gestion de la mémoire sont constituées principalement du programme exécutable, des données manipulées et de la pile d'exécution.

- **Gestion des fichiers :** Les informations relatives à la gestion des fichiers comprennent les identificateurs des fichiers manipulés par le processus.

### 5.2. La migration de processus :

La migration de processus est le déplacement d'un processus en cours d'exécution d'une machine source vers une machine destination, ces deux machines doivent être reliées par un réseau de communication. Il n'est pas indispensable que ces deux machines utilisent une mémoire partagée.

La migration de processus d'une machine source vers une machine destination consiste à interrompre le processus qui s'exécute sur la machine source pour extraire son contexte d'exécution, à transférer ce contexte vers la machine destination, à créer, sur la machine destination, un nouveau processus auquel on affectera le contexte d'exécution transféré et à mettre à jour les liens de communication avec les autres processus. Une fois ceci fait, le processus sur la machine source doit être détruit tandis que le processus sur la machine destination est lancé et représente le processus déplacé. [26]

### 5.3. Les étapes de la migration [23] :

Les figures II-1 à II-5 ordonnées chronologiquement, illustrent les différentes étapes de la migration des agents mobiles.

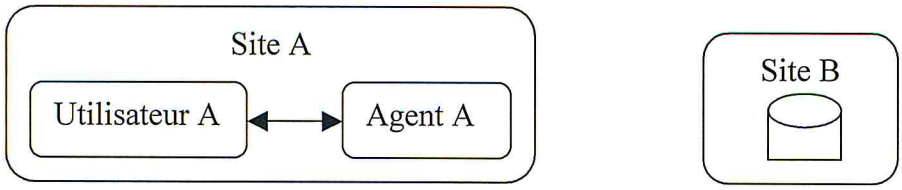



**Figure II-1: Migration ( phase 1).**

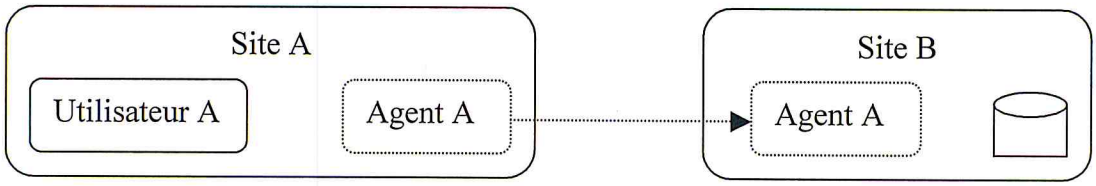



**Figure II-2: Migration ( phase 2).**

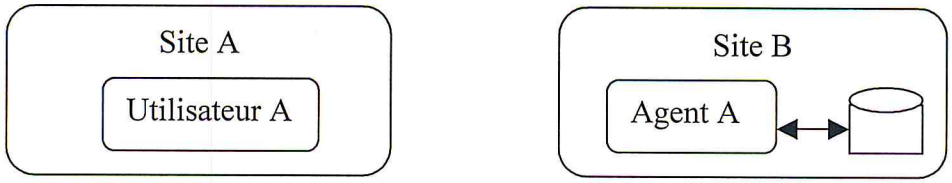



**Figure II-3: Migration ( phase 3).**

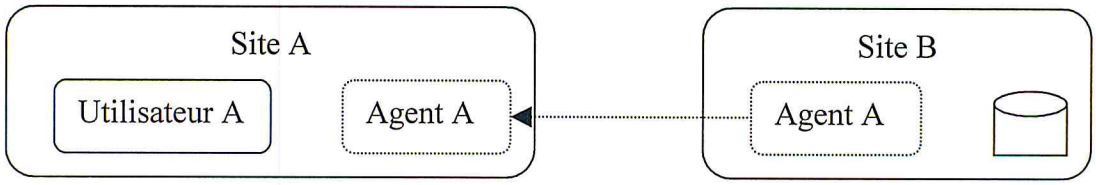



**Figure II-4: Migration ( phase 4).**

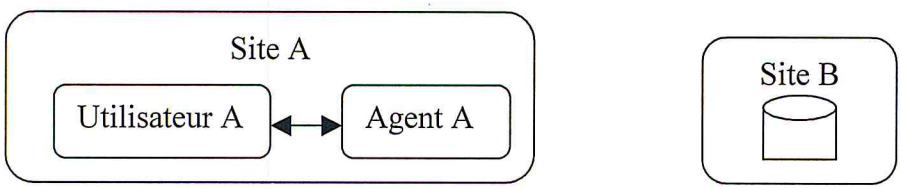



**Figure II-5: Migration ( phase 5).**

L'utilisateur A a besoin des informations sur un tel sujet et ces informations sont disponibles au niveau de site B (sachant que l'utilisateur A travail sur le site A), l'utilisateur A décrit une requête et délègue son agent personnel pour exécuter cette requête et récupérer ces informations, ceci est illustré par la figure II-1. Puis comme il est apparu dans la figure II-2, l'agent déplace de site A vers le site B sur lequel se trouvent les informations requises dans une base de donnée. La figure II-3 illustre que l'agent interroge la base de donnée. Une fois l'agent dispose les informations qu'il cherche, il retourne vers son site natal, ce qu'il est montré par la figure II-4. Finalement, comme il est illustré par la figure II-5 l'agent rapporte –d'une manière interactive- les résultats à son utilisateur.

Une approche basée sur l'utilisation d'un agent statique est illustrée par la figure II-6, l'interaction avec la base de donnée est à travers le réseau.

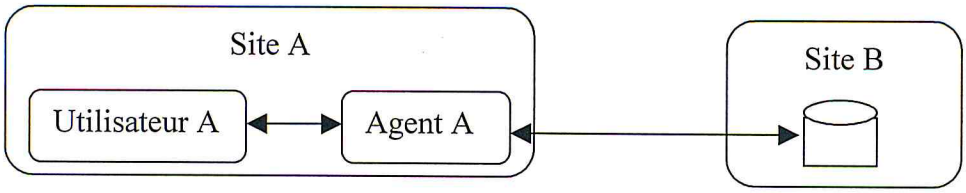



**Figure II-6: Interaction à travers le réseau.**

Dans la première approche - les migrations exceptées - il n'y à pas des communications inter-sites. Détaillant un deuxième exemple encore plus détaillé.

Trois utilisateurs veulent négocier, en déléguant leurs agents personnels, l'heur d'un rendez-vous. La figure II-7 illustre une solution à ce problème basée sur les agents mobiles ( les cinq étapes de migration citées précédemment sont révisés en une seule figure afin de préserver l'espace ). Les trois agents migrent vers un site commun et négocient dans ce site. Lorsque les agents satisfaits, ils retournent vers ses sites origines pour rapporter les résultats.

La figure II-8 montre l'alternative classique basée sur l'utilisation des agents statiques interagissent à travers le réseau. Ce qui distingue les deux approches est le fait que dans la première - basée sur l'utilisation des agents mobiles - il n'y à que les migrations comme communication inter-sites.

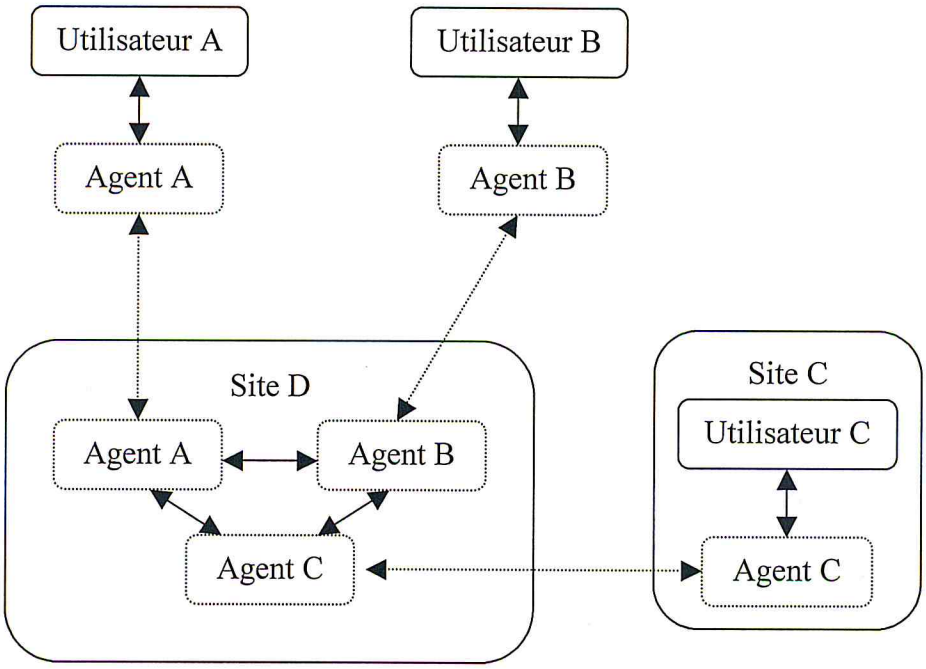



**Figure II- 7: Négociation entre agents mobiles.**

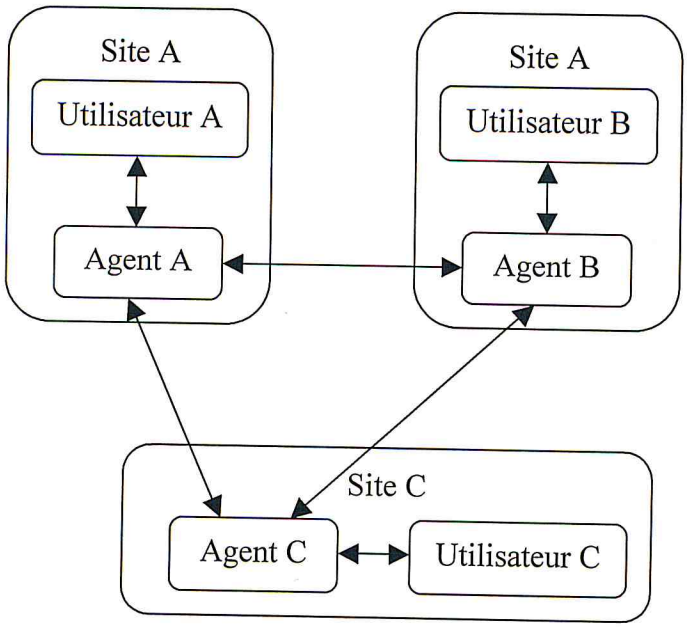



**Figure II-8: Négociation entre agents statiques.**

### 5.4. L'importance de la mobilité [23] :

Avec la large diffusion et la forte utilisation des réseaux informatiques et de l'Internet, l'importance de la mobilité augmente jour après jour et les applications basées sur les agents mobiles remplacent au fur et à mesure les applications basées sur les modèles classiques. Il arrive que le nombre d'agent négocient sur un sujet commun devient très grand, dans ce cas la distribution intelligente des agents devient non seulement avantageuse mais simplement indispensable afin de limiter au maximum les interactions inter-sites.

L'interaction directe avec des services ou avec des agents n'est pas le seul avantage qui peut offrir la migration. En effet, n'importe quel agent conçu pour exécuter certaines tâches. Le but commun entre tous les agents vraiment autonomes est de survivre. Pendant sa vie, l'agent a besoin de ressources informatiques telle que un cycle CPU, la mémoire, une base de donnée …etc. Il arrive parfois que les ressources d'un site donné soient complètement occupées, si un agent réside sur ce site il risque d'être sérialisé sur disque. Si le cas, l'agent a perdu complètement son autonomie. Dans ce cas, pour le réanimer, l'agent n'a que compté sur d'autres unités.

L'éviction de cette situation est fortement conseillée. Une fois la charge d'un site donné devenu trop grand, les agents résident sur ce site - s'ils sont malins - migreront vers des sites moins chargés.

### 5.5. Implémenter la mobilité :

Pour qu'une implémentation de la mobilité soit complète, il faut qu'elle assure le transfert de tous les éléments d'un agent. Ces éléments sont le code, les données (état au niveau conceptuel) et l'état d'exécution (état au niveau de l'implémentation). Le système

Telescript développé par GeneralMagic est un bon exemple d'un système qui implémente complètement la mobilité.[23]

### 5.5.1. Les mécanismes de base [23] :

La figure II-11 illustre les éléments d'un système « Telescript ». Nous y trouvons le site A qui présente un utilisateur A et le site B qui présente une base de donnée. L'utilisateur A cherche un service qui se trouve dans la base de donnée, on trouve aussi six agents. La figure montre outre que ces éléments trois lieux ("lieu : place dans la terminologie de Telescript" [23]). En effet, afin d'assurer et de contrôler les interactions entre les agents, les services et les ressources des sites, les agents ne se trouvent pas comme des processus complètement indépendants dans le site : ils sont contenus dans les lieux. L'avantage d'utilisation de lieux ne pas seulement de faciliter la gestion des agents mais aussi d'offrir un mécanisme qui répond aux questions de sécurité. Pour implémenter la mobilité, il faut qu'on prise en compte la sécurité du système. En effet, plus le nombre de sites accueillant des agents est grand, plus le concept de la mobilité est utile, mais les propriétaires des sites n'accepteront jamais l'immigration d'agents inconnus si l'intégrité de leur système n'est pas garantie. Donc, un nombre important de développeurs des systèmes à base d'agent mobile étudient le problème de sécurité.

Quand on dit la sécurité, c'est-à-dire non seulement la façon avec la quelle on peut protéger les sites contre les actions nocives de la part des agents, mais également la façon avec la quelle on peut protéger les agents eux-mêmes contre les actions des sites et leur cohabitant. La figure II-9 illustre l'agent E déplaçant entre les sites A et B. comme on à dit antérieurement, la migration dans le système Telescript est complète c'est à dire que le code et les données et l'état d'exécution de l'agent E déplacent entre les sites A et B. Pour cela les éléments de l'agent E doivent être sérialisés pour les transférer par le réseau. Après la sérialisation et le transfert, l'agent doit être reconstitué dans l'autre site sur la base de ses éléments sérialisés. Le transfert des éléments par le réseau ne pose pas un problème car ce transfert n'est qu'une simple application de la technologie réseaux. Toutefois, la sérialisation et reconstitution sont plus délicates.

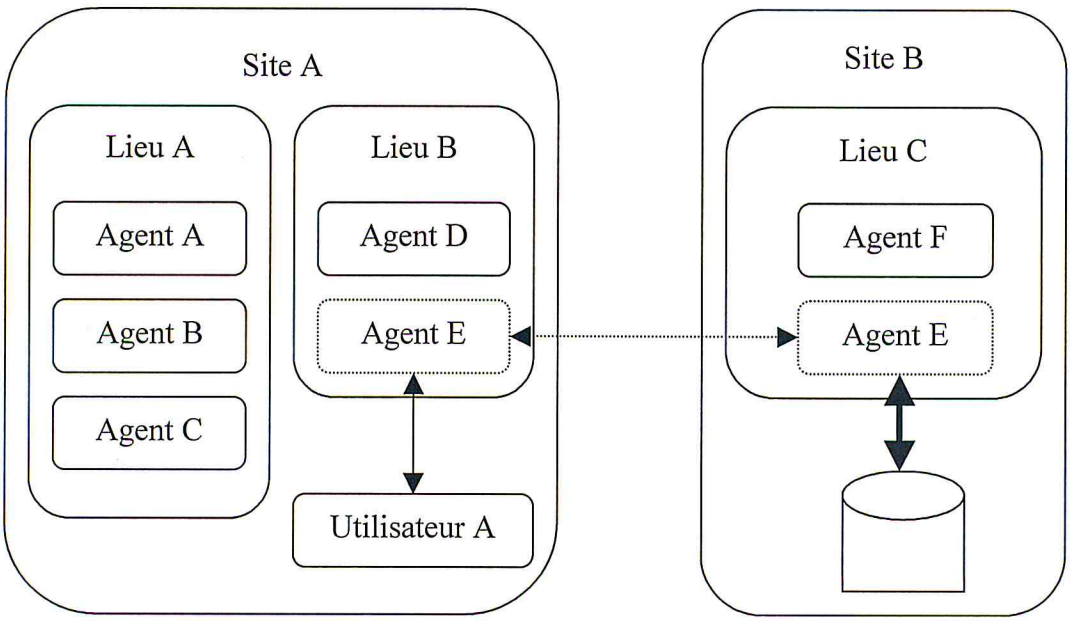



**Figure II- 9 : Eléments essentiels d'un système « Telescript ».**

La sérialisation et la reconstitution sont un peut complexe. Heureusement, les langages modernes tel que Java facilitent la procédure car ces langages supportent la sérialisation et facilitent sensiblement le processus.

L'émergence de Java nous a familiarisé avec la sérialisation et la reconstitution car ce mécanisme était avant l'apparition de Java plus ésotérique. En utilisant Java, comme il est montré dans la sous-section suivante, il est facile de faire déplacer un agent mobile.

La sérialisation et la reconstitution d'état d'exécution d'un processus reste un problème difficile. Le système Telescript supporte cette fonctionnalité, mais peu de systèmes actuels la supportent. Cette fonctionnalité est absente dans la plus part des systèmes actuels car la majorité de ces systèmes utilisent Java comme langage de programmation et ce dernier ne supporte pas, pour des raisons de sécurité la sérialisation de cet état. Toutefois un certain nombre de chercheurs développent des systèmes capables de sérialiser et reconstituer de l'état d'exécution. L'absence de cette fonctionnalité ne semble pas très gênante pour les agents car ce ci implique qu'après chaque migration l'agent commence son exécution à partir d'un point d'entré fixe, " par exemple par l'exécution de la méthode « arrive() » "[23]. Notons que dans certains cas, la migration doit être transparente. Soulignons que dans ce cas, différent du nôtre, le terme « agent mobile » ne devrait pas être utilisé.

De point de vue implémentation, la migration ne se fait pas d'une façon autonome et les agents ont besoins d'une entité externe pour réaliser ce processus. Typiquement cette entrée est appelée le **moteur de migration** (" *engine* dans la terminologie de Telescript" [23]) et doit être installée sur tous les sites entre lesquels les agents souhaitent migrer. Ce moteur, chargé de la gestion des lieux, peut être complètement indépendant. Malgré son indépendance, il peut coopérer avec ces autres systèmes, qui peuvent être intégrés dans le système de gestion. Les figures II-10 à II-13 montrent quatre cas différent. Notons qu'on à fait une abstraction de lieux dans ces figures. Soulignons que d'autres configurations peuvent être utilisées.

La figure II-10 montre la configuration la plue simple. Dans cette configuration, une seule version de moteur de migration est nécessaire et un seul système de gestion est supporté. Les migrations sont effectuées par le moteur de migration. L'agent exécuter au-dessus du système de gestion sous le contrôle de moteur.

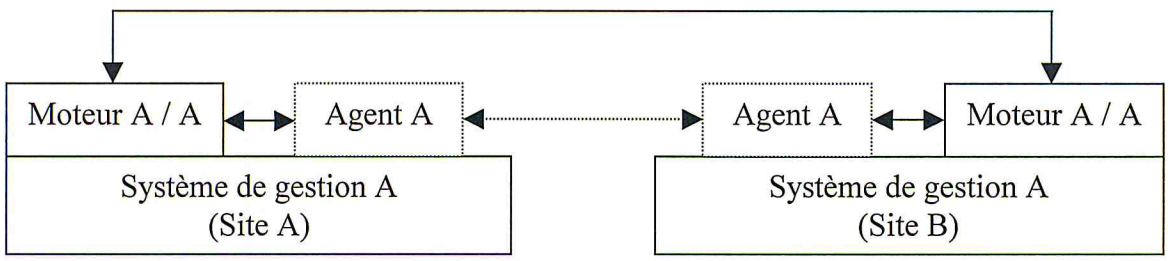



**Figure II-10: Implémentation de la mobilité (cas 1).**

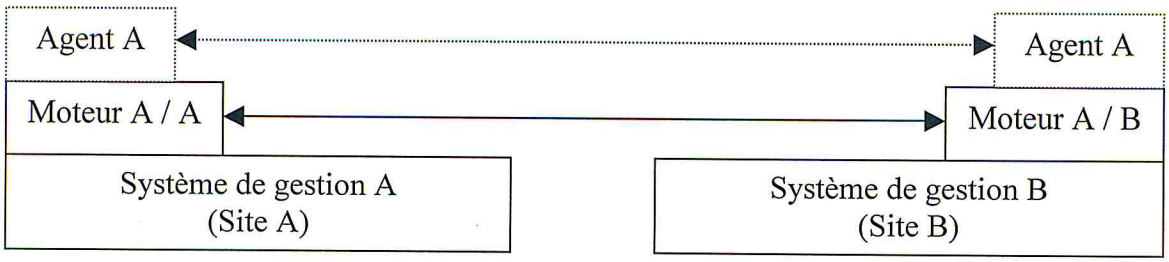



**Figure II-11: Implémentation de la mobilité (cas 2).**

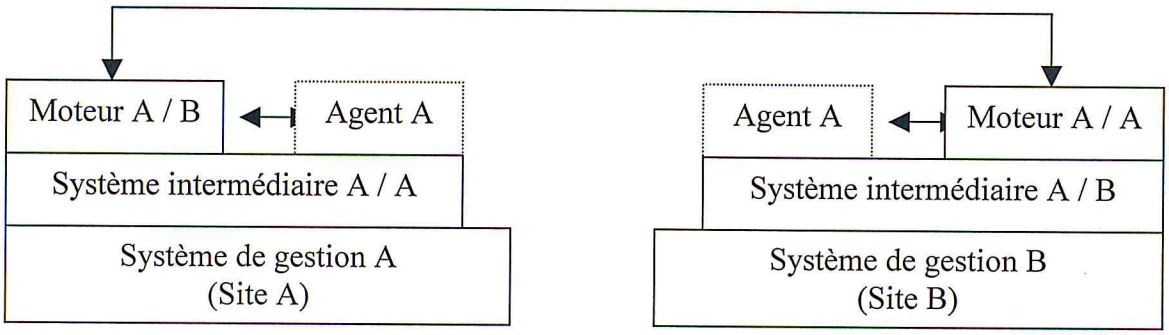



**Figure II-12: Implémentation de la mobilité (cas 3).**

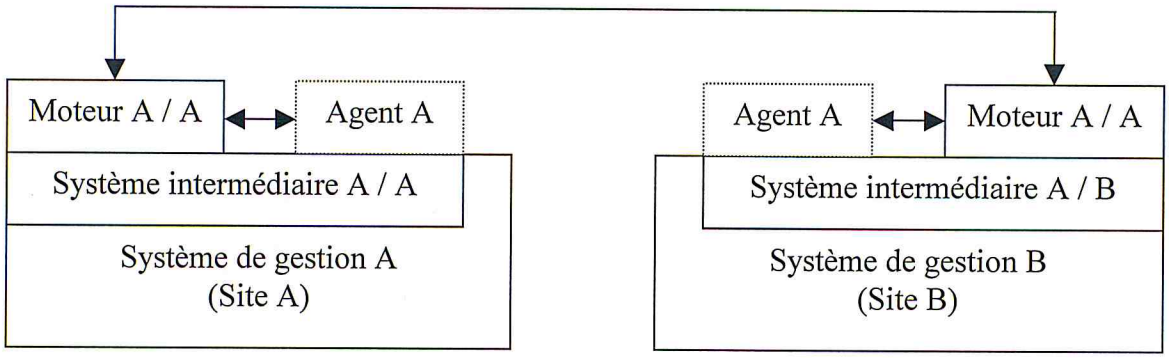



**Figure II-13: Implémentation de la mobilité (cas 4).**

Le système illustré par la figure II-11 supporte plusieurs plate-formes et nécessite plusieurs versions de moteur, une pour chaque plate-forme. Dans ce système les agents ne s'exécutent pas immédiatement au-dessus du système de gestion, ils doivent être interprétés par les moteurs.

Encore une autre configuration, montré par la figure II-12, supporte plusieurs plate-formes et nécessite plusieurs systèmes de gestion. Dans cette configuration, le système de gestion est caché aux moteurs et aux agents par un système intermédiaire. Java est un bon exemple d'un tel système intermédiaire, dont la machine virtuel interprète les moteurs et les agents. Pour que l'interopérabilité soit possible, il faut que le système intermédiaire soit supporté sur les différentes plate-formes.

Le système de la figure II-13 est le même que celle de la figure II-12. La différence entre les deux systèmes est que dans la figure II-15, le système intermédiaire

est intégré dans le système de gestion. Soulignons que Java est de plus en plus intégré dans les systèmes de gestion actuels.

## 6. Les systèmes d'agent mobile existants :

Les plate-formes de développement des systèmes à base d'agent mobile doit fournit toutes les fonctionnalités dont les agents mobiles ont besoins, en particulier le modèle de navigation[23]. Pour le cycle de vie, il faut définir les services de créations, de destruction, de démarrage, de suspension, d'arrêt…etc. Le modèle de calcul indique les moyens de calcul de l'agent. Le modèle de sécurité décrit la façon dont les agents ont le droit d'accéder aux ressources de réseau et aux terminaux. Le modèle de communication définit les modes de communication entre agent et entre un agent et une autre entité, comme le réseau. Le modèle de navigation se charge de tous les transports de l'agent entre deux entités de réseau. [24]

Actuellement, plusieurs plate-formes ou systèmes qui supportent la mobilité existent. Parmi eux, il y'à des systèmes qui sont commercialisés et il y'à des systèmes qui restent comme projet de recherche. Nous en présentons les plus important, mais il existe, à part de cette présentation, plusieurs autres systèmes. Des études sur ces systèmes existent dans la littérateur (par exemple : [25], [26] et [27]).

Pour comparer et valider ces systèmes, en prend en compte plusieurs critères. La distribution du système est un des principaux critères qu'on pris en compte. Si l'utilisation d'un système est limitée à un seul site, ce système ne sert à rien même s'il est parfait : "plus il y'à des sites disponibles, plus la mobilité devient intéressante " [23]. Lorsqu'un système est largement distribué, il devient – typiquement - plus mis à l'épreuve. Deux autres critères sont très importants qui sont la stabilité et la sécurité. L'efficacité, soit au niveau d'utilisation, soit au niveau d'exécution des agents joue également un rôle crucial. La facilité de conception d'application est aussi un critère important. Un système supporte les fonctionnalités de base est moins intéressant qu'un système incorporant des modèles SMA ou des protocoles de communication sophistiqués. Le coût des licences du système joue évidemment un rôle. Finalement, à un niveau plus bas, la facilité de programmation est un dernier critère que nous considérons.

Ci-dessous, on décrit les systèmes les plus répondus, suivi par une courte évaluation[23].

***Agent Tcl (D'Agents) :*** *( http://www.cs.dartmouth.edu/~agent)*

Un des premiers systèmes supportant les agents mobiles a été développé par Robert Gray, David Kotz et d'autres à Darthmouth College [Agent Tcl]. A l'origine, le système n'avait pas de nom et utilisait TCP/IP, le courrier électronique et la commande UNIX « rshell » comme mécanismes de transfert[28]. Plus tard, la même équipe a introduit « Agent Tcl », un système utilisant Tcl comme langage principal de programmation des agents[30][29]. Toutefois, à la base, le système supporte plusieurs langages, à savoir Tcl, Java et Scheme, et récemment le système a été rebaptisé « D'Agents ». Notons qu'Agent TCL est un des seuls systèmes actuels n'étant pas basé directement sur Java.

**Aglets :** ( http://www.trl.ibm.co.jp/aglets)

Le « Aglet Workbench » d'IBM est sans doute un des systèmes les plus connus [Aglets]. Danny Lange, un des pionniers de la communauté, était à la base de son développement au Japon [31] [32]. Le système est facile à charger du Web et à installer sur tout système supportant Java. Il est assez fiable, mais la documentation en ligne n'est pas d'une qualité commerciale. Notons que Danny Lange a quitté IBM pour joindre General Magic dans le développement d'Odyssey.

***Concordia :*** *(http://www.meitca.com/HSL/Projects/Concordia)*

Concordia est également un système basé sur Java, mais développé par le Horizon Systems Laboratory de Mitsubishi [Concordia] [33]. Il s'agit d'un système récent, orienté vers des applications au niveau des entreprises. Par conséquent, la sécurité du système est exceptionnellement soignée. [34]

***Mole*** *: (www.informatik.uni-stuttgart.de/ipvr/vs/projekete/mole.html)*

Le groupe dédié aux systèmes distribués, au sein de l'Institute of Parallel and Distributed High-Performance Systems à l'Université de Stuttgart, dirigé par Kurt Rothermel, était le premier à développer une plate-forme basée sur Java supportant les agents mobiles [Mole]. Récemment, une troisième version de leur système, appelé Mole, a été publiée sur le Web. A travers les versions différentes de la plate-forme, la stabilité et la fonctionnalité du système ont continuellement crû.

***Odyssey*** *: (http://genmagic.com/technology/mobile_agent.html)*

Le successeur de Telescript, développé également par General Magic, s'appelle Odyssey [Odyssey]. Différant du premier, ce dernier est basé sur Java. Il est largement inspiré par Telescript, mais n'a pas exactement la même fonctionnalité. Par exemple, Odyssey ne supporte pas le transfert de l'état d'exécution. Le système n'a toujours pas dépassé les versions *bêta*, et ne joue visiblement pas le rôle important que jouait Telescript dans la stratégie de General Magic. Répétons, toutefois, que Danny Lange a joint l'équipe.

***Telescript*** *: (http://www.genmagic.com/technology/techwhitepaper.html)*

Telescript, développé par General Magic, peut être considéré comme la référence dans le domaine. [24][35] Il est parlant que son inventeur, Jim White, tient un brevet sur la notion même d'agents mobiles. [36] Telescript était un des seuls systèmes supportant le transfert de l'état d'exécution des agents. Toutefois, le système, conçu pour une application industrielle, exigeait des ressources importantes, était plutôt cher (5000 USD par moteur), et offrait une fonctionnalité pas encore demandée par le marché. Par conséquent, le système n'existe plus et sa valeur est surtout historique.

### 6.1. Une courte évaluation :

On à présenté ci-dessus un aperçu sur les plate-formes principales. Bien sur qu'il existe plusieurs autres systèmes et plusieurs autres en phase de développement. Cela veut dire que, au fur et à mesure, des nouveaux systèmes apparaître.

La distribution, la stabilité et la sécurité sont les critères qui dominaient. Selon ces critères, les Aglets d'IBM constituent un des meilleurs choix. Mais, par intuition, la concurrence viennent de systèmes plus récents qui exploitent les nouvelles possibilités de Java et les normes de plus en plus établies de l'informatique distribuée. Actuellement, le système Voyager d'ObjectSpace est un concurrent d'Aglets, malgré que les Aglets sont plus orientés vers les agents et les Systèmes Multi-Agent que Voyager.

## 7. Les avantages des agents mobiles :

### 7.1. Plus sur et meilleure tolérance :

La vie d'un programme classique est liée à la machine ou il s'exécute. Un agent mobile peut se déplacer pour éviter une erreur matérielle ou logicielle ou tout simplement un arrêt de la machine. [19]

### 7.2. Gestion de ressource :

Les agents sont des programmes à exécutés qui peuvent avoir la nécessité d'accéder aux ressources de bas niveau comme un cycle CPU, la mémoire et le réseau …etc. Ou aux ressources de haut niveau comme la base de données, interface utilisateur ou serveur SQL…etc. Il est souhaitable d'interdire les agents d'accéder directement à la ressource et que le système contrôle tous les accès aux ressources par des agents avec différentes politiques imposées. Les contraintes d'accès aux ressources sont classifiées dans différents niveaux, un mécanisme de négociation de ressource pourrait optimiser la répartition des ressources aux agents[37].

Un gestionnaire de ressources est donc nécessaire. Il doit pouvoir établir des contraintes d'accès, limiter les accès aux ressources en fonction des moyens (financiers ) de l'agent, attribuer des coûts aux ressources effectuées. Bien que ce problème soit critique[37], la plupart des infrastructures existent ne définissent aucun mécanisme de gestion.

### 7.3. Réduction de la charge réseau :

Il est généralement plus avantageux, en terme de performances, de faire voyager du code plutôt que des informations. [19]

Par exemple un utilisateur travail au niveau de poste 1 et cherche les numéros de téléphone de quelques personnes. La liste des noms est situe au niveau de poste 2 et la liste qui contienne les numéros de téléphone est située au niveau de poste 3. La figure II-14 illustre brièvement comment l'utilisation de la technologie agent mobile réduire la charge de réseau par rapport au technologie client/serveur. [38]

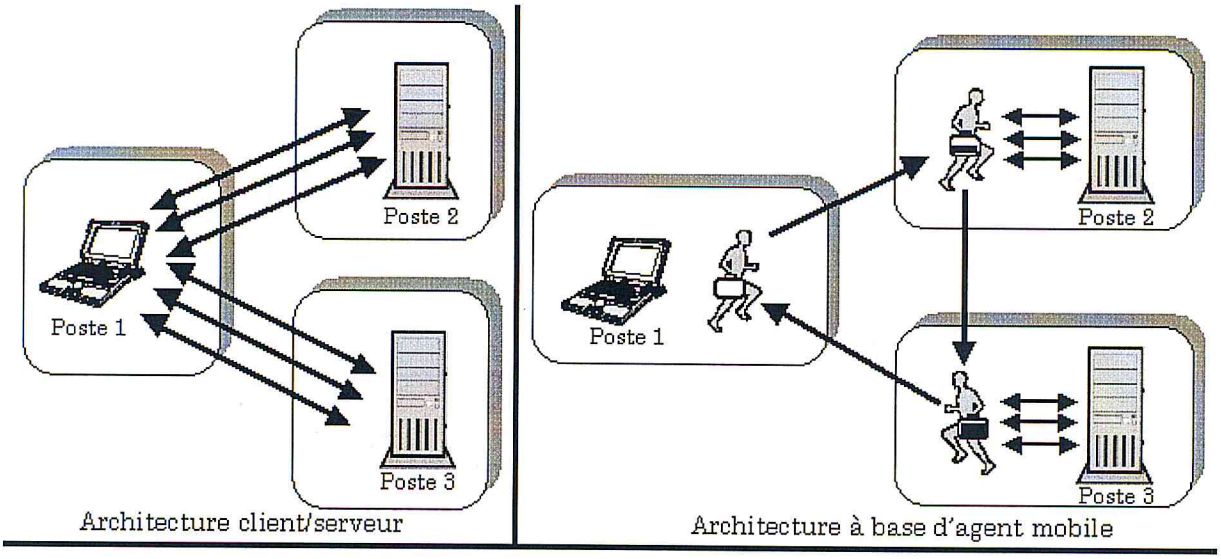



**Figure II-14: Réduction de la charge réseau.**

### 7.4. Extensibilité des fonctions d'un serveur :

Un agent mobile peut éteindre l'ensemble fixe des fonctions d'un serveur. Cette caractéristique permet aux machines clientes d'adapter un serveur à des besoins spécifiques. [20]

Par exemple un utilisateur a besoin d'un document et ce dernier est situé dans un poste connecté avec le poste de l'utilisateur. Le poste qui contient le document cherché ne possède pas un algorithme de compression, l'utilisateur délègue un agent mobile qui inclus dans son comportement un algorithme de compression. L'agent mobile déplace vers le site où se trouve le document cherché, exécute l'algorithme dans ce site, intègre le document compressé dans son comportement et retourne vers son site natal avec le document cherché. La figure II-15 explique brièvement cet exemple. [38]

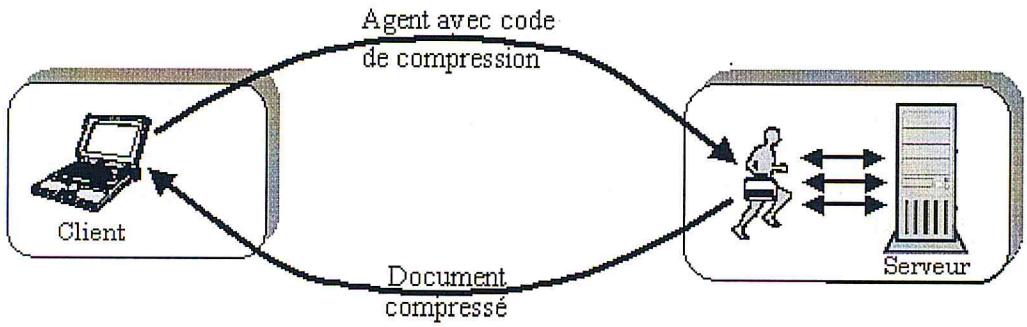



**Figure II- 15: Extensibilité des fonctions d'un serveur.**

## 8. La sécurité :

Un agent doit se protéger des attaques extérieures : par rapport aux autres agents, à ses environnements d'exécution et pendant son déplacement sur le réseau. Son intégrité et la confidentialité des données qu'il transporte doivent être assurés. [18]

Le problème de la sécurité des agents mobiles s'énonce simplement par trois problèmes à résoudre[21] :

- ✓ La protection du système vis à vis des agents s'exécutant dans le système ;
- ✓ La protection de l'agent durant le transport dans le réseau ;
- ✓ La protection de l'agent vis à vis du système où il s'exécute.

La sécurité peut être mise en place au travers des langages agents intégrant des mécanismes de sécurité. *Telescript, Java,* et *Safa-Tcl* sont des langages qui intègrent la partie sécurité dans l'implémentation des agents mobiles. [16]

## 9. Conclusion :

Les systèmes à base d'agents mobiles joueront un rôle très important dans l'avenir des applications réparties, particulièrement pour les applications sur les bases de données réparties, dans lesquelles la sécurité et l'éviction de goulet d'étranglement sont les facteurs les plus importants qu'on prend en compte dans la conception d'un tel système, ce que nous offre les agents mobiles ainsi que d'autres avantages.

L'autonomie et la délégation des agents mobiles remédient aux problèmes de connexion intermittente ainsi qu'aux problèmes imposés par la grande envergure des réseaux.

# CHAPITRE III.
# Les bases de données réparties

## 1. Introduction :

Dès le début de l'informatique et jusqu'à un temps récent, l'approche dominante à la gestion de données dans l'entreprise a été centralisée. Cela veut dire que les bases de données et le système de gestion de base de données étaient centralisées au niveau d'un seul calculateur. Avec l'avènement des années 80, le développement des réseaux d'ordinateurs a permis la connexion des ordinateurs et l'échange de données et le partage des ressources entre eux. Ces facteurs ont donné la naissance d'une nouvelle approche, c'est la « *base de données réparties* ». Cette dernière reprend les mêmes principes généraux que ceux d'une BD centralisée mais en étendant les techniques existantes où en proposant certains concepts nouveaux qui sont particuliers à la répartition des données.

Une base de données réparties peut être vue comme une collection de données logiquement corrélées et physiquement réparties sur plusieurs machines interconnectées par un réseau de communication [40]. Elle offre beaucoup d'avantages par rapport à l'approche centralisée plus vulnérable aux pannes et goulots d'étranglement.

Dans cet chapitre nous présentons le concepts de base de données réparties, ainsi leur objectives et leur fonctionnalité et les solutions aux problèmes techniques posés par leur mise en œuvre.

## 2. Définition :

Une **base de donnée répartie** (ou **BDR** ) est un ensemble de base de donnée coopérants qui résident sur différentes machines, vues et manipulées par l'utilisateur comme une seule base de données centralisée. Cela veut dire que l'utilisateur sur n'importe quel site peut accéder et manipuler des données sur n'importe quel site, et

n'aura pas à se soucier de la localisation physique des données et déléguera cette tache au **système de gestion de base de donnée répartie** (ou **SGBDR**).[40]

Le **SGBDR** est le système qui incombe la tache de la gestion de la BDR, il est constitué d'un ensemble de logiciels systèmes constitués des gérants de données réparties, des gérants de communication et des SGBD locaux résidant sur chaque site. Ces derniers sont appelés **serveurs** du SGBDR et les applications qui accèdent aux informations distribuées par des interfaces sont dites **clients** du SGBDR.[40]

## 3. Système de Gestion de Bases de Données Réparties (SGBDR) :

Les systèmes de gestion de bases de données réparties (SGBDR) ont été inventés à la fin des années 70 afin d'intégrer les bases de données et les réseaux. Le SGBD réparti doit offrir les mêmes services qu'un SGBD, soit décharger les utilisateurs de tous les problèmes de concurrence, de fiabilité et d'optimisation des requêtes.

### 3.1 Objectifs des SGBD répartis :

**Indépendance à la localisation**

Le SGBDR Garantir la transparence de la localisation des données, elle cache le fait que les données ne résident pas nécessairement sur le site utilisateur. L'information concernant la localisation des données est maintenue dans le dictionnaire de donnée et consultée par le SGBDR pour déterminer la localisation des relations impliquées dans les requêtes des utilisateurs. [40]

**Indépendance à la fragmentation**

Une relation dans un système réparti est constituée de différents fragments, situés sur des sites différents. L'information concernant la fragmentation est maintenue dans le dictionnaire des données et utilisée par le SGBDR pour traduire automatiquement les requêtes sur les relations conceptuelles en requêtes sur des fragments. Donc il cache à l'utilisateur le fait que les données soient fragmentées. [40]

**Multiclients multiserveurs**

Dans une architecture client/serveur classique, un client peut demander à un serveur d'exécuter une requête, elle est appelée une requête distante. Un SGBDR doit permettre une fonctionnalité plus générale consistante à permettre à plusieurs clients d'accéder aux données réparties aux niveaux des différents serveurs du SGBDR. Pour cela des mécanismes pour le contrôle de concurrence adaptés à l'environnement distribué doivent êtres prévus. [40]

En plus, un SGBDR doit permettre l'exécution d'une requête répartie. Cette dernière est une requête émise par un client dont l'exécution nécessite l'exécution de N sous requêtes mettant en œuvre N serveurs. Le même raisonnement peut s'appliquer sur les transactions réparties qui sont en général un ensemble de requêtes réparties s'exécutant

d'une manière atomique[41]. En résumé, la Figure III-1 illustre les modes de fonctionnement multiclients multiserveurs.

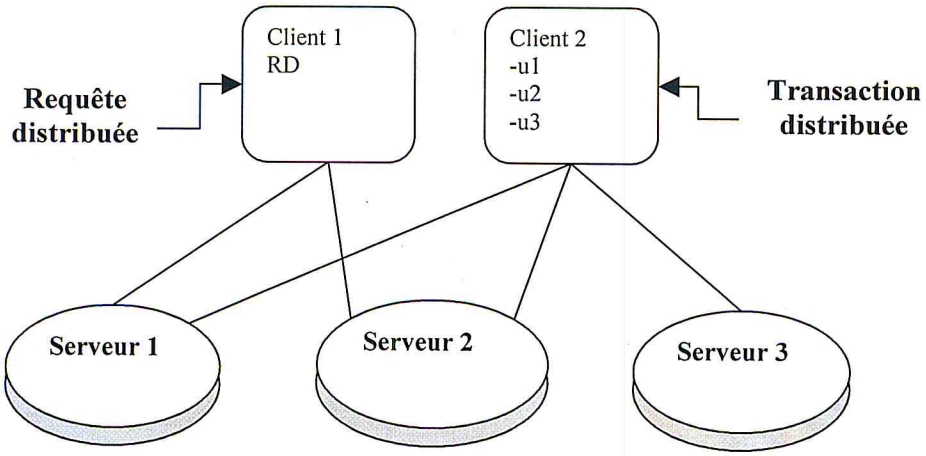



**Figure III-1 : Fonctionnement Multiclients Multiserveurs.[41]**

**Meilleure disponibilité**

Une base de données réparties apparent pour garantir une meilleure disponibilité des données et d'endiguer les problèmes inhérents aux systèmes centralisés qui sont très vulnérables aux pannes et aux goulots d'étranglement[41].

Une fonctionnalité très utilisée dans les SGBDR est la **réplication**. Elle consiste à prévoir des copies de données à travers le système. En cas où le SGBDR en sollicite une et qu'elle est indisponible (pour des raisons de panne par exemple), il aura la possibilité de se replier sur une autre. Bien évidement, le problème de synchronisation entre les différentes répliques de la même occurrence de donnée se pose avec acuité. En général, des protocoles sont prévus pour permettre à ce que l'atomicité des transactions englobe à la fois les données et leurs répliques[41].

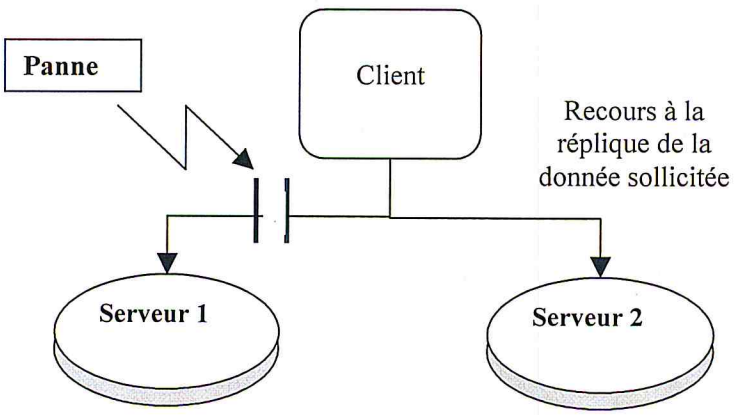



**Figure III-2: Réplication des données.[41]**

### Autonomie locale

L'autonomie locale vise à garder une administration locale séparée et indépendante pour chaque serveur participant à la base de données répartie, et d'éviter la nécessité d'une administration centralisée. Une reprise après panne doit être accomplie localement et ne peut en aucun cas impacter les autres sites. Cela est réalisé grâce à la notion de **schéma local** et de **schéma global.** [41]

Un schéma local décrit la base de données locale gérée par le SGBD local. Et un schéma global permet de définir l'ensemble des types de données de la base répartie. La Figure III-3 illustre ce principe.

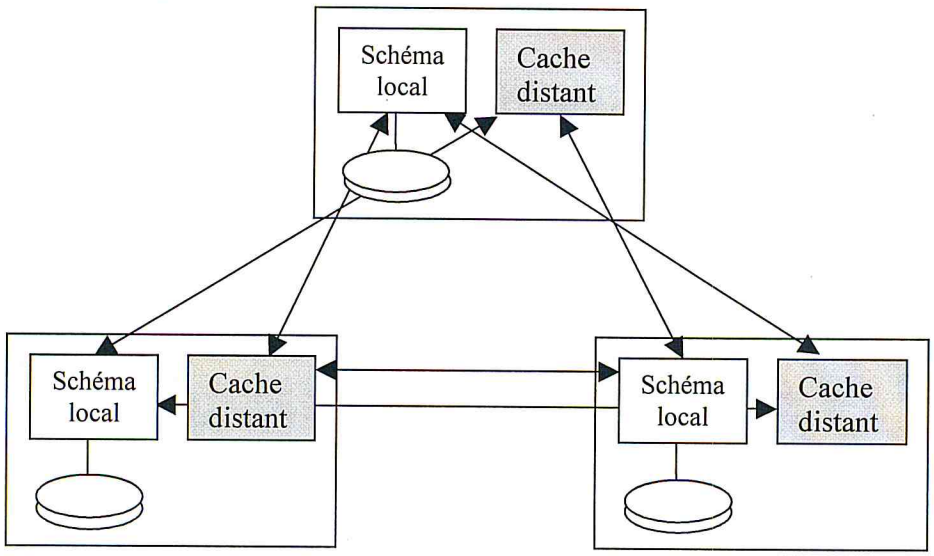



**Figure III-3: Coopération de schémas. [41]**

### Indépendance aux SGBD

Le SGBDR permet de cacher le fait que les SGBD locaux peuvent être différents. Ces SGBD peuvent différer parce que leurs modèle de données (relationnel, Objet, …etc.)et le langage associé sont différents, ou plus simplement parce qu'ils ont été conçus par des groupes différents. [40]

### Extensibilité

L'extensibilité d'une base de données réparties est sa capacité d'augmentation incrémentale, par introduction de nouveaux sites dans le réseau, avec un impact minimal sur les bases de données locaux et les programmes d'applications existants. [40]

### Performance

Dans une base de données réparties, La duplication des fragments (si la fragmentation n'est pas supportée) peut servir à augmenter la fiabilité et la disponibilité des données ainsi que la performance d'accès. En peut atteint une meilleure performance par l'utilisation des traitements parallèles et favorisation les accès locaux. [40]

Un SGBDR rendre la répartition des bases de données transparentes aux utilisateurs. La base de données étant répartie, il faut également répartir certaines fonctionnalités du SGBD. Le schéma d'un SGBD réparti est résumé dans la figure suivante.

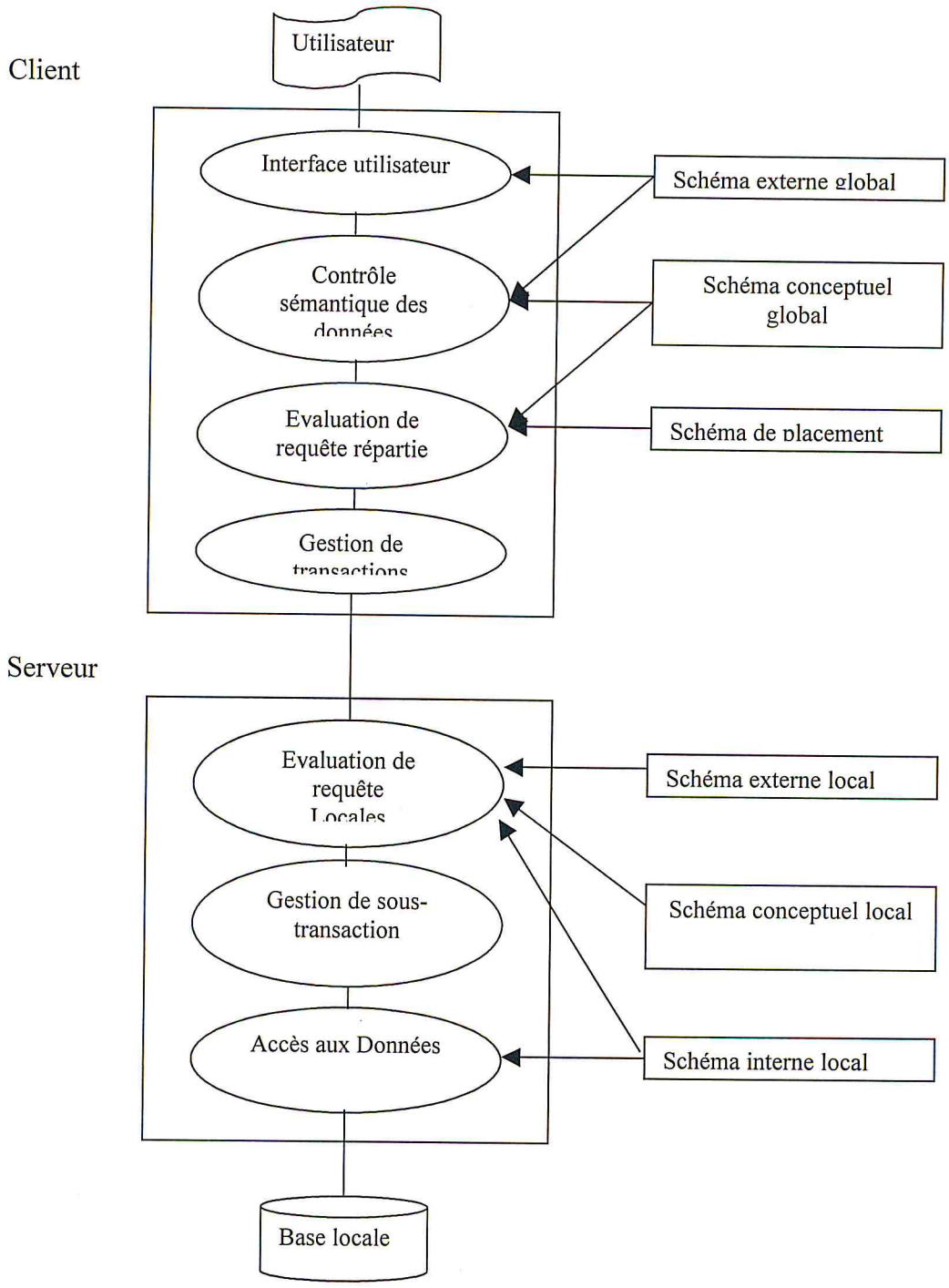



**Figure III- 4: Architecture fonctionnelle d'un SGBDR. [40]**

## 4. Conceptions d'une base de données réparties :

La conception d'une base de données réparties peut être le résultat de deux approches totalement différentes, l'approche descendante et l'approche ascendante [42].

### 4.1 Approche descendante ( *TOP dpwn design* ) :

On commence par définir un schéma conceptuel global de la base de données réparties, puis on le distribue sur les différents sites en des schémas conceptuels locaux. La répartition se fait donc en deux étapes : la première étape est la fragmentation, et la deuxième étape est l'allocation de ces fragments aux sites.

L'approche *top down* est intéressante quand on part du néant. En effet, dans le cas contraire, départ de bases existantes, l'approche devient inapplicable, et la méthode *bottom up* est utilisée. La figure III-5 illustre cette approche :

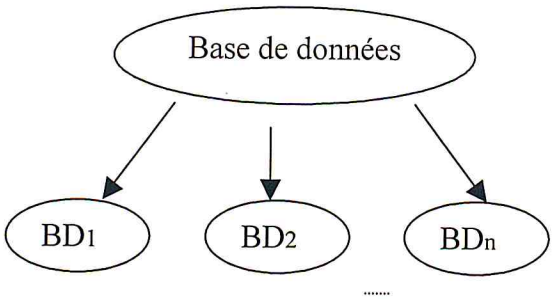



**Figure III- 5: Conception descendante. [40]**

### 4.2 Approche ascendante ( *bottom up design* ) :

La conception ascendante permet l'intégration des bases de données locaux existantes dans une base distribuée. Il s'agit cette fois de construire un schéma global à partir des schémas locaux, généralement existants. En d'autres termes, les schémas conceptuels locaux existent et il faut réussir à les unifier dans un schéma conceptuel global. La figure III-6 illustre cette approche :

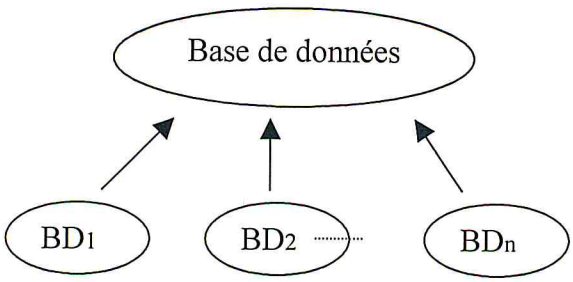



**Figure III- 6: Conception Ascendante. [40]**

### 4.3 Fragmentation :

La fragmentation est le processus de décomposition d'une base de donnée logique en un ensemble de "sous" bases de données. Cette décomposition doit évidement être sans perte d'information pour être acceptable. Pour cela il existe plusieurs technique de fragmentation, définies par l'unité de fragmentation. [42]

### 4.3.1 Répartition des classes d'objet :

Ce sont les classes (relation en relationnel, type d'entité et type d'association en Entité-Association, type d'article en CODASYL, classe en Orienté-objet) qui peuvent être réparties dans des fragments différents. Toutes les occurrences d'une même classe appartiennent au même fragment.

Dans l'exemple suivant (Tableau III-1 à -3) la base de données relationnelle peut être fragmentée en {Compte, Client} et {Agence}.

| NoClient | Agence | TypeCompte | Somme |
|---|---|---|---|
| 174 723 | Laussane | courant | 123 345.89 DA |
| 177 498 | Genève | courant | 34 564.00 DA |
| 201 639 | Laussane | courant | 45 102.50 DA |
| 201 639 | Laussane | dépôt | 325 100.00 DA |
| 203 446 | Genève | courant | 274 882.95 DA |

**Tableau III-1: Relation Compte.**

| NoClient | NomClient | Prénom | Age |
|---|---|---|---|
| 174 723 | Villard | Jean | 29 |
| 177 498 | Cattell | Blaise | 38 |
| 201 639 | Tsellis | Alan | 51 |
| 203 446 | Kowalsky | Vladimir | 36 |

**Tableau III-2: Relation Client.**

| Agence | Adresse |
|---|---|
| Laussane | Rue du Lac, 3, 1002 Lausanne |
| Genève | Avenue du Mont Blanc, 21, 1200 Genève |

**Tableau III- 3: Relation Agence**

### 4.3.2 Répartition des occurrences :

Les occurrences d'une même classe peuvent être réparties dans des fragments différents (avec leur valeur complète). C'est la fragmentation horizontale.

- L'opérateur de partitionnement est la sélection
- L'opérateur de recomposition est l'union

Dans l'exemple précédent, la relation Compte peut être fractionnée en Compt1 et

Compt2 avec la fragmentation suivante :

Compt1 = σ[TypeCompte = 'courant']Compte
et Compt2 = σ[TypeCompte = 'dépôt']Compte
et la recomposition suivante Compte = Compt1 ∪ Compt2

### 4.3.3 Répartition des attributs :

Toutes les valeurs des occurrences pour un même attribut se trouvent dans le même fragment. Une fragmentation verticale est utile pour distribuer les parties des données sur le site où chacune de ces parties est utilisée.

- L'opérateur de partitionnement est la projection.
- L'opérateur de recomposition est la jointure.

Soit le partitionnement de la relation précédente Client en deux relations :

Cli1 = $\pi_{[NoClient,NomClient]}$Client

et Cli2 = $\pi_{[Noclient, Prénom,Age]}$Client

| NoClient | NomClient |
|---|---|
| 174 723 | Villard |
| 177 498 | Cattell |
| 201 639 | Tsellis |
| 203 446 | Kowalsky |

Relation Cli1

| NoClient | Prénom | Age |
|---|---|---|
| 174 723 | Jean | 29 |
| 177 498 | Blaise | 38 |
| 201 639 | Alan | 51 |
| 203 446 | Vladimir | 36 |

Relation Cli2

La relation d'origine est obtenue avec la recomposition suivante : Client = Cli1 * Cli2

### 4.3.4 Répartition des valeur :

C'est la combinaison des deux fragmentations précédentes, horizontale et verticale. Les occurrences et les attributs peuvent donc être répartis dans des partitions différentes.

- L'opération de partitionnement est une combinaison de projections et de sélections.
- L'opération de recomposition est une combinaison de jointures et d'unions.

La relation Client est obtenue avec : (Cli3 $\cup$ Cli5) * Cli4 * Cli6

| NoClient | NomClient |
|---|---|
| 174 723 | Villard |
| 203 446 | Kowalsky |

Relation Cli3

$\pi_{[NoClient,NomClient]}(\sigma_{[Age<38]}Client)$

| NoClient | NomClient |
|---|---|
| 177 498 | Cattell |
| 201 639 | Tsellis |

Relation Cli5

$\pi_{[NoClient,NomClient]}(\sigma_{[Age>=38]}Client)$

| NoClient | Prénom |
|---|---|
| 174 723 | Jean |
| 177 498 | Blaise |
| 201 639 | Alan |
| 203 446 | Vladimir |

Relation Cli4

$\pi$[ NoClient,NomClient] Client

| NoClient | Age |
|---|---|
| 174 723 | 29 |
| 177 498 | 38 |
| 201 639 | 51 |
| 203 446 | 36 |

Relation Cli6

$\Pi$[NoClient,NomClient] Client

### 4.4 Architecture des Schémas :

Un schéma doit être élaboré afin de déterminer la localisation de chaque fragment et sa position dans le schéma global. L'architecture de schémas illustré par la figure III-7 propose une organisation idéale d'une base de données réparties visé à atteindre les objectifs cités précédemment. Cette architecture repose sur un schéma global à trois niveaux : schéma externe, conceptuel et de placement. Et les niveaux de schéma qui décrivent une base de données locale (schéma local).[42]

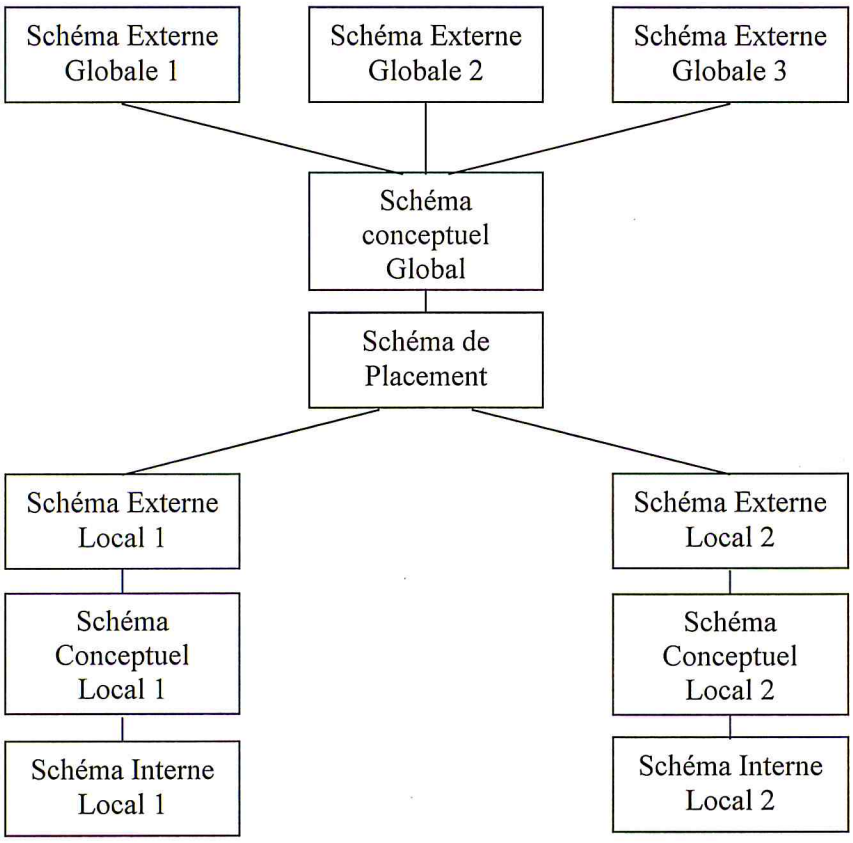



**Figure III-7: Architecture des schémas d'une BDR.[40]**

En effet, le schéma de placement précise la façon dont les relations sont placées sur les différents sites dans le réseau. Il contient tout les informations relatives à la localisation, la fragmentation, en particulier les règles de transformation des données du

niveau de schéma conceptuel global en données localisées, fragmentées et dupliquées. De même, le schéma conceptuel global définit toutes les relations appartenant à la base de données réparties[40].

## 5. Manipulation de données réparties :

Les règles d'exécution et les méthodes d'optimisation de requêtes définies pour un contexte centralisé sont toujours valables, mais il faut prendre en compte d'une part la fragmentation et la répartition des données sur différents sites et d'autre part le problème du coût des communications entre sites pour transférer les données.[42]

### 5.1 Requêtes sur BDs réparties :

Comme pour le traitement de requêtes en Bases de données centralisées, on produit l'arbre algébrique de la requête. Chaque feuille de l'arbre représente une relation, et chaque nœud représente une opération algébrique. On enrichit l'arbre avec les informations sur la répartition des données sur les différents sites, en particulier sur le site où chaque opération de la requête doit être exécutée. [42]

La complexité d'une requête dans une base de données répartie est définie en fonction des facteurs suivants[42] :

- Entrées/ Sorties sur les disques (*disks I/Os*), c'est le coût d'accès aux données.
- Coût CPU : C'est le coût des traitements de données pour exécuter les opérations algébriques (jointures, sélections ...).
- Communication sur le réseau : c'est le temps nécessaire pour échanger un volume de données entre des sites participant à l'exécution d'une requête.

#### 5.1.1 Traitement de requêtes réparties :

Le but est d'affecter de manière optimale un site d'exécution pour chacune des opérations algébriques de l'arbre. Pour cela, on associe à chacune des feuilles le site sur lequel la relation va être puisée. On cherche ensuite à associer à chaque nœud de l'arbre le site sur lequel l'opération algébrique associée à ce nœud sera exécutée.

Généralement, il faut faire localement tous les traitements qui peuvent y être faits. Ainsi, lorsque tous les opérandes d'une même opération algébrique sont situés sur le même site, la solution la moins coûteuse pour exécuter cette opération est le plus souvent de l'exécuter sur ce site. Ceci est notamment toujours vrai pour les opérateurs unaires qui font diminuer le volume d'informations (sélection, projection). Pour diminuer le volume de données transmis d'un site à un autre, il faut limiter les transferts d'information aux seules informations utiles. Pour cela il faut systématiquement rajouter des projections dans l'arbre algébrique pour abandonner les attributs inutiles. Il faut aussi noter que les parties de requêtes indépendantes peuvent être exécutées en parallèle sur des sites différents et donc abaisser le temps total d'exécution.[42]
Exemple : Soit la BDR composée des relations suivantes :

P (NP, NOMP, MADE_IN, COULEUR, POIDS)

U (NU, VILLEU, NOMU)
F (NF, NOMF, VILLEF, ADRESSE, PAYS, COEF)
PUF (NP, NU, NF, DATE, QUANTITE)

P : Produit, U :Usine, F Fournisseur
Les relations U et PUF sont sur le site 𝔸.
Les relations F et P sont sur le site 𝔹.

Et soit la requête suivante :

"Donner les numéros et les noms des fournisseurs qui ont livré un produit italien à une usine située à Lausanne ". cette requête est décomposée pour être exécutée sur la BD répartie de la manière suivante :

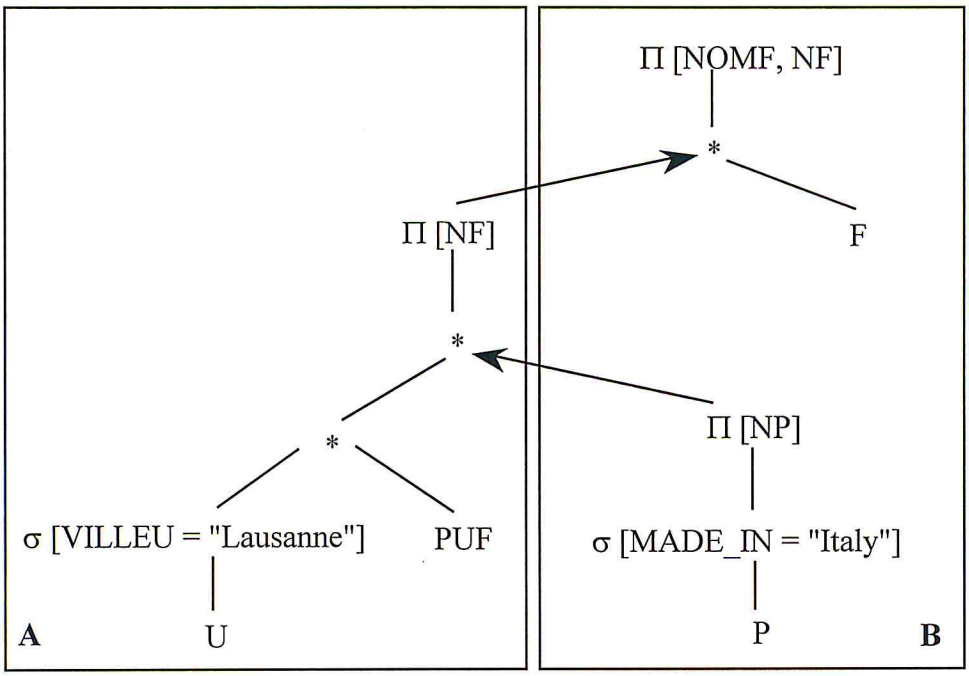



**Figure III- 8: Exécution d'une requête répartie. [42]**

Ceci correspond à la séquence suivante :

sur 𝔹 : R1 = Π[NP] σ(MADE_IN = "Italy")P
transmettre R1 à 𝔸
sur 𝔸 : R2 = Π[NF] (σ(VILLEU = "Lausanne")U * PUF) * R1
transmettre R2 à 𝔹
sur 𝔹 : Π(NOMF, NF) (R2 * F)

### 5.1.2 Optimisation des requêtes :

Le rôle de l'optimisation est de déterminer une stratégie d'exécution, en éliminant les opérations inutiles. Cela permet de minimiser le coût qui est définis par le temps total d'exécution de la requête, qui est aussi la somme de tous les temps d'exécution consacrés par les différents sites participant à la requête[40]. l'optimisation de requête dans un environnement réparti se base aussi sur le temps de communication qui est en fonction des

critères suivants : Volume de données transféré, nombre de communications, temps d'accès et temps de réponse, temps de calcul UC sur chacun des sites.En peut résumer le coût total par la fonction suivant[41] :

| Coût total = a* coût (E/S) + b* coût (CPU) + c* coût (communication) |
|---|

Les coefficients a, b et c dépendent de l'environnement du SGBDR, par exemple réseaux locaux, réseaux longues distances, etc. En général les SGBDR ignorent les facteurs coût(E/S) et coût (CPU) qui sont pris en compte par les SGBD locaux. Actuellement, cela devient totalement inapproprié grâce aux performances de plus en plus élevées des réseaux d'interconnexion.

### 5.2 Mise à jour de BD réparties :

Le mise à jour dans une relation du schéma global se traduit par plusieurs mises à jours dans différents fragments. Il faut identifier les fragments concernés par l'opération de mise à jour, puis décomposer en conséquence l'opération en un ensemble d'opération de mise à jour sur ces fragments.[42]

***Insertion*** **:** Retrouver le fragment horizontal concerné en utilisant les conditions qui définissent les fragments horizontaux, puis insertion du tuple dans tous les fragments verticaux correspondants.

***Suppression*** **:** Rechercher le tuple dans les fragments qui sont susceptibles de contenir le tuple concerné, et supprimer les valeurs d'attribut du tuple dans tous les fragments verticaux.

***Modification*** *:* Rechercher les tuples, les modifier et les déplacer vers les bons fragments si nécessaire.

### 5.3 Transaction réparties :

Une transaction est composée d'une suite de requêtes dépendantes de la base qui doivent vérifier les propriétés d'**A**tomicité, de **C**ohérence, d'**I**solation et de **D**urabilité, qui font l'**ACID**ité d'une transaction.[42]

- **Atomicité** : cette propriété signifie qu'une transaction est traitée comme une seule opération. Toutes les actions sont menées à bien ou aucune d'entre elles.
- **Cohérence** : une transaction est un programme qui amène la BD d'un état cohérent à un autre état cohérent, tel que toutes les contraintes d'intégrité restent vérifiées.
- **Isolation** : c'est la propriété qui impose à chaque transaction de voir la BD cohérente. Une transaction en exécution ne peut révéler ses résultats à d'autres transactions concurrentes avant d'effectuer le *commit*.

- **Durabilité** : c'est la propriété qui garantit lorsqu'une transaction a effectué son *commit*, le résultat sera permanent, et ne pourra être effacé de la BD quelques soient les pannes du système rencontrées.

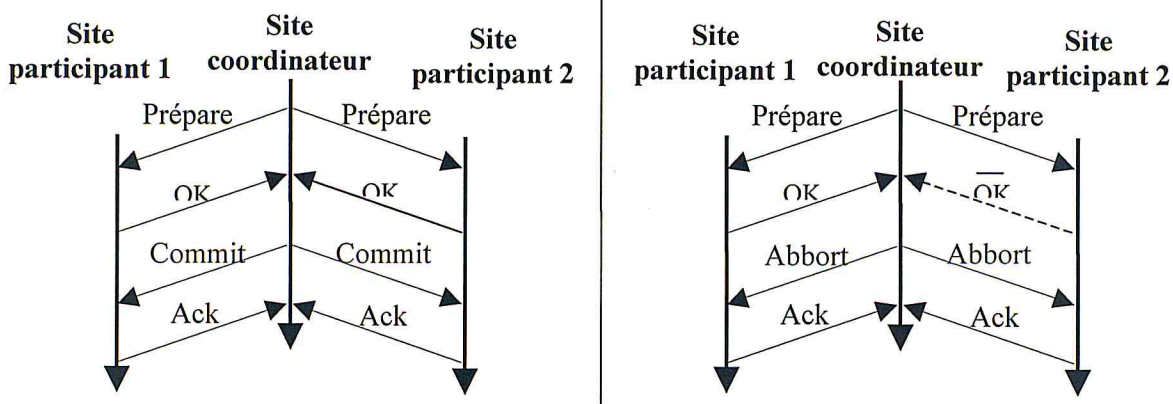
Ces propriétés doivent être garanties dans le cadre d'un système réparti. En particulier il faut assurer que toutes les mises à jour d'une transaction sont exécutées sur tous les sites. Le problème essentiel à résoudre est le risque d'incohérence lié au contrôle. Chaque site peut décider de valider ou d'annuler une transaction. Il faut donc coordonner les validations.

### 5.4 Protocole de validation à deux phases (2-PC) :

Le protocole de validation en deux phases a été proposé afin de coordonner l'exécution des commandes COMMIT par tous les sites participant à une transaction. Le principe consiste à diviser la validation en deux phases[42] :

- Phase de préparation : le coordinateur demande à chaque participant de se préparer pour valider la transaction locale.
- Phase de validation : le coordinateur ordonne à tous les participants de valider ou d'annuler leur transaction. La décision est prise par le coordinateur tenant compte de la réponse de chaque participant.

La Figure III-9 montre les différentes étapes du protocole.




Validation en deux phases avec succès

Validation en deux phases avec panne du participant 2

**Figure III- 9: Gestion des transactions distribuées avec le protocole de validation à deux phases[41]**

Le coordinateur représente le nœud d'un système réparti qui dirige le protocole en centralisant le contrôle. Et Le participant représente le nœud d'un système réparti qui

exécute des mises à jour de la transaction et obéit aux commandes de préparation, validation ou annulation du coordinateur.

### 5.5 Réplication :

La **réplication** est la technique permettant la gestion de copies multiples des données pouvant diverger à un instant donné mais convergent vers la même valeur à terme. Deux contraintes essentielles ont motivées l'utilisation de cette technique : La première est liée au besoin d'augmenter les performances des BDR et cela en permettant l'accès aux données la où elles sont le plus proche évitant ainsi des transferts inutiles et les goulots d'étranglement. La seconde est sensée à améliorer la disponibilité des données et se prémunir des éventuelles pannes qui peuvent survenir aux niveaux de certains serveurs. Dans un tel cas, on a toujours la possibilité de se replier sur une autre copie. [41]

## 6. Base de données réparties hétérogènes :

Une base de données réparties n'implique pas une parfaite homogénéité des sites locaux. c'est à dire, dans la pratique, la mise en place d'une base de données répartie, par une conception ascendante, impose de tenir compte des différents systèmes existants. Dans ce cas en parle sur les Bases de données réparties hétérogènes. Nous distinguons quatre niveaux d'hétérogénéité[43] :

### 6.1 Hétérogénéité des nœuds :

Si les différents nœuds sont utilisés avec des systèmes d'exploitations, cela apporte que très peu de problèmes au concepteur du système distribué. Cette hétérogénéité est englobée dans un paramétrage correct du médiateur qui support même différents protocoles de communication sur un même réseau (TCP/IP, DecNet, …).

### 6.2 Hétérogénéité des SGBD :

La normalisation du langage SQL facilite l'intégration de bases de données traitées par différents constructeurs. La traduction des différents dialectes et la connexion « technique » est assurée par une passerelle dédiée. Dans le monde Windows, on utilise souvent des passerelles génériques de type ODBC/OLEDB.

### 6.3 Hétérogénéité de structures :

Le problème majeur est difficilement maîtrisable réside dans des différences de structure pour une même entité ou pire encore, différents paradigmes de gestion de données (relationnel, objet, Orienté objet, …)

## 7. Conclusion :

Après ce qui précède, il est clair que l'objectif général d'une base de données répartie est de permettre à l'utilisateur de manipuler un ensemble de base de données gérées sur des sites différents comme une base de données centralisée.

Actuellement, il existe sur le marché des SGBD qui permettent la répartition des données sous forme des produits compétitifs qui ont de hautes performances, Par exemple : O2, ORACLE/STAR et INGRES/STAR.

# CHAPITRE IV.
# Problématique et Architecture

## 1. Introduction :

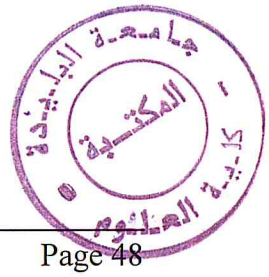
De nos jours, toutes les entreprises sont équipées au minimum d'un réseau local et d'une base de données centralisée gérée par un système. Et pour les plus importantes d'entre elles d'une base de données distribuée sur les sites participants dans ce réseau. Un tel système permet d'interroger cette grande base de données d'une manière plus efficace, en respectant l'optimisation de la bande passante, le coût de communication et d'autres facteurs qui sont prises en considération lors de construction d'une application qui utilise le réseau.

Dans ce chapitre nous présentons une Architecture proposée pour notre système appelé **RIAM** ( **R**echerche D'**I**nformation avec les **A**gents **M**obiles ) qui est basé sur le modèle d'agent mobile, ce système permet de rendre l'accès transparent aux données réparties à grande échelle qui peuvent être hétérogènes soit au niveau de structure de donnée et au niveau de constructeur.

## 2. Problématique :

Dans un contexte d'une base de donnée réparties, l'intégration des bases de données hétérogènes est difficile[42]. Cette difficulté devient de multitude source, ça provient de:

- Les différentes structures des données. Actuellement il existe quatre structures de donnée: fichiers, relationnelle, objet, et orient objet.
- Multitude de constructeur de données : SQLServer,Oracle, Sybase…etc.
- Les différents systèmes d'exploitation : Windows, Linux…etc.



Et dans le même contexte, le plan d'exécution d'une requête répartie fait par un optimiseur afin d'obtenir une meilleure performance. Ainsi que les méthodes d'optimisation dynamique de requête proposées à ce jour sont centralisées[51], elle dépendent du nœud ou du site maître responsable de l'optimisation. Sachant que les méthodes d'optimisation dynamiques sont supervisées par un processus maître qui contrôle tous les processus participant à l'optimisation. Dans ce cas les processus réalisant les opérations en cours d'exécution ou en attente de libération de ressources sont donc entièrement contrôlé par l'optimiseur qui présent un goulot d'étranglement. Il engendre, en plus, un échange de message relativement important sur les réseaux. [51]

Dans un réseau à grande échelle, il est nécessaire de défini une architecture qui permet de l'intégration des données et résoudre le problème d'hétérogénéité. Et de rendre autonome l'exécution d'une requête pour éviter le goulot d'étranglement et réduire le volume de données transférées ainsi que le nombre de message de contrôle.

## 3. Approche :

La plus part des systèmes d'interrogation des bases de données dans un réseau à grande échelle qui se basent sur la technologie agent mobile reposent sur une approche principale, cette approche consiste à utiliser un agent statique et un agent mobile (figure IV-1) [52]. Chacun de ces deux a son rôle qui est commun dans tous les systèmes avec quelques petites différences.

En général, le rôle de l'agent statique est :

- Limité l'autonomie de l'agent mobile.
- Accueillere l'agent mobile.
- Interroger la base de données.
- Affecter les résultats de l'interrogation à l'agent mobile (pour les transférer vers le poste de l'utilisateur

L'agent mobile est conçu pour effectuer les tâches suivantes :

- Transporter la requête vers les postes où se trouvent les bases de données qui contiennent les informations requissent.
- Transporter les résultats des requêtes vers le site d'origine.

Les systèmes misent en œuvre basent globalement sur cette approche avec quelques détails privés pour chaque système qui seront cités (quelques un) ultérieurement.

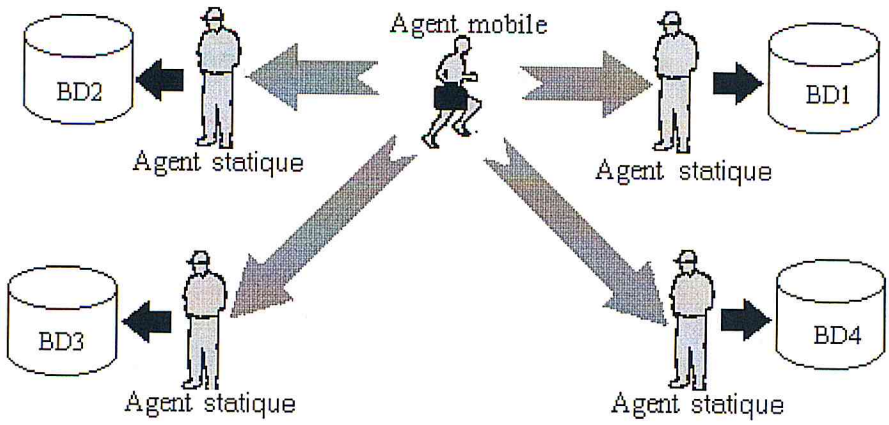



**Figure IV-1: Approche principale.[52]**

## 4. Présentation des systèmes à base d'agents mobiles:

Comme on a dit précédemment les systèmes existants dans ce domaine basent sur l'approche décrite ci-dessus. On va présenter dans la sous-section suivante l'architecture de deux systèmes qui sont presque pareils (coté architecture ) avec quelques différences.

Dans le système [53], les concepteurs utilisent un agent statique contient les fonctions nécessaires pour accéder aux bases de données. De plus il est utilisé pour chaque modèle de donnée ( base de donnée relationnel, objet relationnel,...etc.) un agent statique. Ce qui rend le nombre des agents statiques dans un même site égal le nombre de modèle de donnée qu'il contient.

D'une manière générale, l'utilisation de l'agent mobile que pour transporter la requête vers le poste où se trouve la base de donnée qui contient l'information requise et transporter les résultats vers le site natal. La figure IV-2 illustre cette architecture :

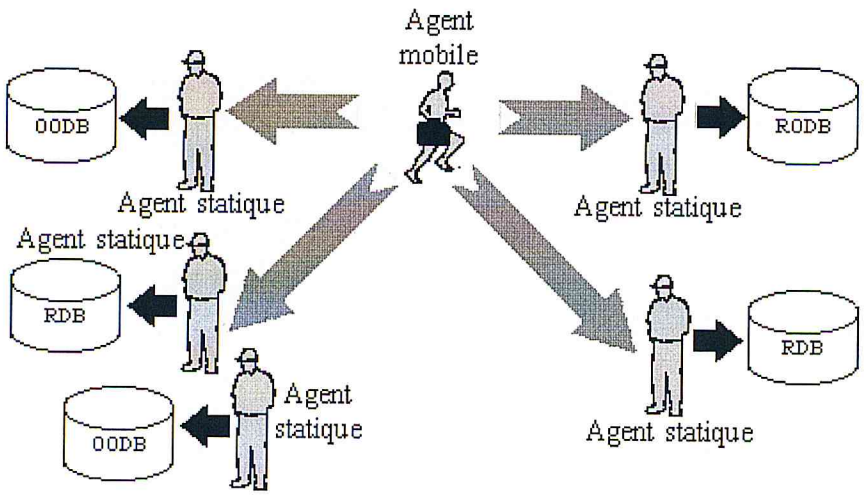



**Figure IV-2: Architecture de système[51].**

Dans un autre système [54] qui s'appelé **Viajerus,** les concepteurs de ce système utilisent un agent mobile et une agence (Agency) qui supporte deux agents statiques

(Gestionnaire d'agent mobile et Agent intermédiaire). Afin de partager le travail de la manière suivante :

– Gestionnaire d'agent mobile (Mobile Agent Manager) pour contrôler les messages et les actions de l'agent mobile crée dans le site de gestionnaire. Chaque instruction, rapport et notification de l'agent mobile sont gérés à travers des messages manipulés par le gestionnaire.

– Agent intermédiaire (Intermediary Agent) : Accueillir les agents mobiles des autres postes et receve les requêtes transportées par ces agents, vérifier si la source d'agents mobile est un nœud.

L'agent mobile est conçu afin de déplacer d'un poste vers un autre, véhiculant avec lui les requêtes décrites par l'utilisateur, quand ces requêtes sont satisfaites il revient vers son site original amenant les résultats de requêtes intégrées dans son comportement.

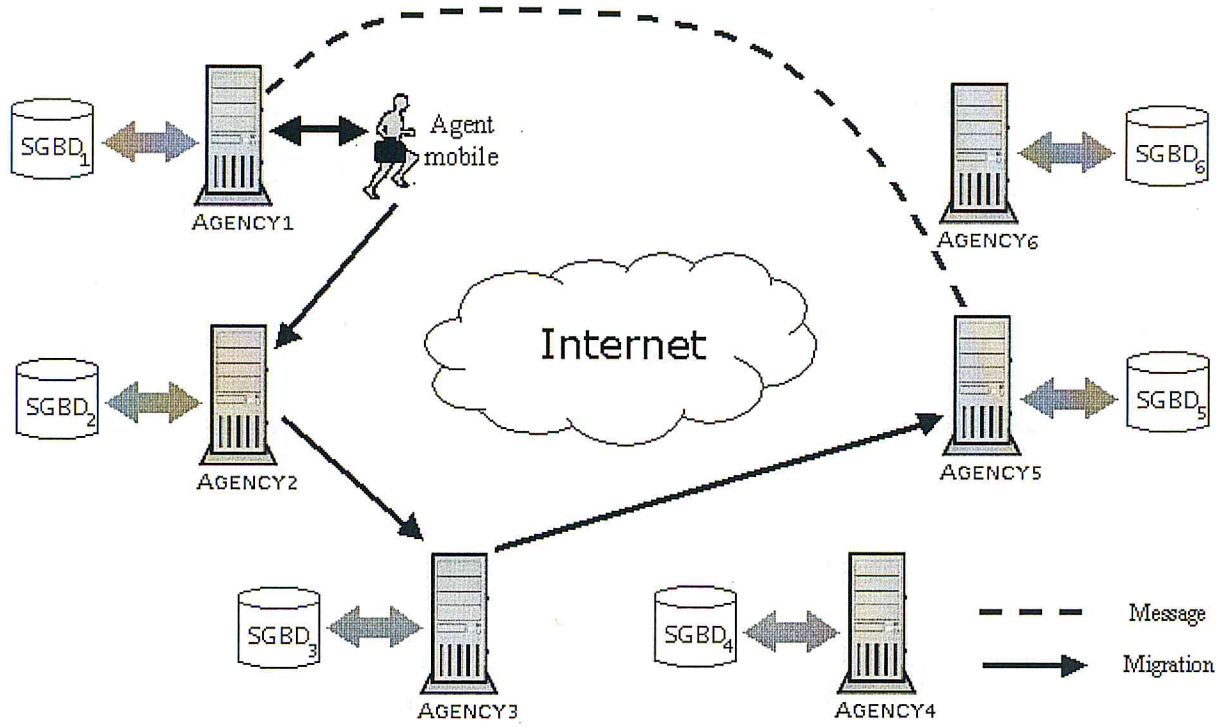



**Figure IV-3: Architecture de système Viajerus.**

Le troisième système [53] est un système qui consulte des bases de données à travers l'Internet à l'aide des agents mobiles, il ressemble le premier système à l'exception que dans ce troisième système l'emplacement des données se trouve dans un site particulier, ce site contient un Middleware afin que ce dernier localise les données à consultées. La communication entre le site de l'utilisateur et le site de Middleware est en mode client/serveur.

Un agent mobile est crée dans le site qui contient le Middleware dont l'objectif est de récupérer les informations cherchées par l'utilisateur, l'agent mobile commence son itinéraire poste par poste jusqu'à ce qu'il récupère toutes les informations qu'il cherche et dans chaque poste il trouve un agent statique qui interroge la ou les bases de données situées dans ce poste et affecte les résultats de ces interrogations vers l'agent mobile. Ce

dernier, près qu'il récupère les informations qu'il cherche, il revient vers son site natal qui est le site qui contient le Middleware ensuite ces résultats sont transmis vers le poste source de requête, cette dernière transmission est en mode client/serveur.

Le figure suivante illustre brièvement l'architecture de ce système :

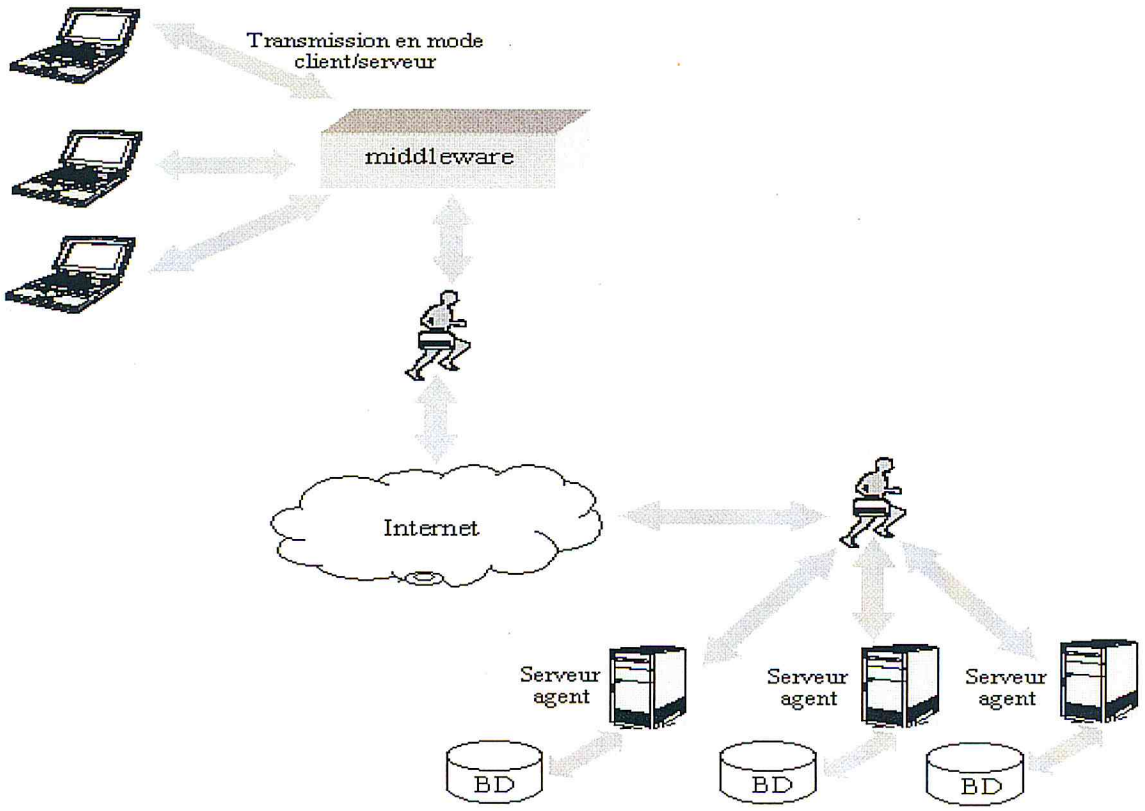



**Figure IV- 4 : Architecture de système[55]**

Les trois systèmes cités précédemment conçus pour atteindre différents objectifs, mais chaque système défini ses propre critères de conception ( autonomie, communication entre agent, itinéraire, stratégie de recherche et sécurité). Le tableau suivant présente une comparaison entre les deux systèmes (décrits précédemment) selon les critères de conception des systèmes.

| | L'autonomie | Commu-nication | Itinéraire | Stratégie de recherche | Sécurité des données |
|---|---|---|---|---|---|
| **Système [53]** | - L'autonomie de l'agent mobile limité par l'agent statique | - Envoi de message | - Définis au pré able après l'excursion de l'agent mobile par l'appel d'une méthode pour trouver l'itinéraire | - Un seul agent transport la demande d'utilisateur et collecte les résultats trouvés | - Faire l'envoi d'une copie de l'agent mobile pour assurer le transport de code agent plus les donné<br>- Limité l'accès aux données par l'agent statique |
| **Système Viajerus** | - L'autonomie de l'agent mobile par l'utilisateur par l'interface d'utilisateur et par l'agent statique | - Envoi de message | - Définis au pré able après l'excursion de l'agent mobile par l'utilisateur | - Un seul agent transport la demande d'utilisateur et collecte les résultats trouvés | - Limité l'accès aux données par l'agent statique |
| **Système [55]** | - L'autonomie de l'agent mobile limité par l'agent statique | - Envoi de message | - Définis par un Middleware qui contient un annuaire des adresses des bases de données . | - Un seul agent transport la demande d'utilisateur et collecte les résultats trouvés | - Limité l'accès aux données par l'agent statique |

**Tableau IV-1 : Table de comparaison entre les systèmes à base d'agent mobile.**

## 5.Présentation d'une proposition :

Dans cette section, nous présentons une solution pour les problèmes cités précédemment.

### 5.1. Architecture globale :

#### 5.1.1. Vue générale :

Le système RIAM qui nous entraîne de réaliser permet de rendre l'accès aux bases de données transparent aux utilisateurs, c'est à dire que l'utilisateur peut interroger plusieurs bases de données qui sont hétérogènes au niveau de constructeur ( Oracle, SQLServer…) ou au niveau de modèle de donnée (RDB,RODB,..) et qui sont réparties dans un réseau dont l'architecture est client/serveur. Pour cela, nous utilisons la technologie agent mobile. La figure IV-4 illustre une architecture générale de ce système.

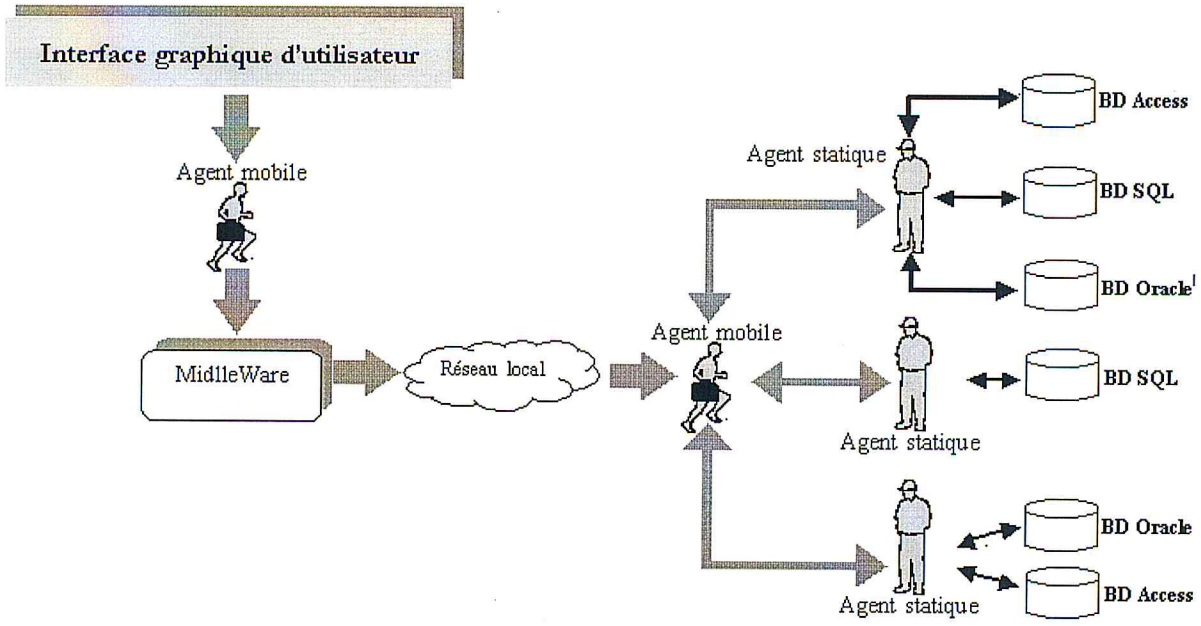



**Figure IV-4 : Architecture générale du système RIAM.**

Cette architecture repose sur trois composants principaux : une agence, agent mobile et un Middleware.

### 5.1.2. Agence :

Une agence est l'entité statique qui offre l'environnement d'exécution pour les agents mobiles locaux et à distance[2]. Une agence dirige et commande toutes les activités dans un site. Elle permet de :

- Créer, envoyer et recevoir les agents mobiles.
- Contrôler les actions et les messages envoyés par les agents mobiles crées localement.
- Assurer l'accès aux ressources pour les agents mobiles.
- Protéger les données contre les accès non autorisés.

Chaque nœud que l'agent mobile visitera doit contenir une agence. Ce dernier crée un autre agent de ressource au niveau de chaque nœud support une base de donnée, cet agent est statique, contient les fonctions nécessaires pour interroger cette base de donnée. Agent de ressource permet de recevoir les demandes des agents mobiles arrivés afin d'accéder aux données désirées. Il permet aussi de traiter et contrôler les actions et les messages des agents mobiles créent localement ( par exemple l'activation, la désactivation de l'agent mobile) pour des raisons de sécurité.

### 5.1.3. Agent mobile :

L'agent mobile est l'entité mobile qui peut déplacer entre les sites du réseau, il représente le mécanisme pour lequel l'utilisateur peut accéder aux ressources locales ou distribuées. Le comportement de l'agent mobile est composé de deux parties : la première est le code d'exécution de l'agent mobile et la deuxième, les informations nécessaires

pour accomplir les taches pour lesquelles l'agent mobile a été conçu. Dans cette partie, l'agent mobile sauvegarde son itinéraire, les Requêtes d'utilisateur et les résultats obtenus.

### 5.1.4. Middleware :

Le Middleware est un outil qui sépare les interfaces utilisateur (IU) et les bases de données BD (figure IV-7). Il est utilisé pour que l'agent mobile trouve son itinéraire à partir des demandes des utilisateurs et ne pas visiter un ou plusieurs sites inutilement, les sites appartiennent à l'itinéraire de l'agent mobile sont celles qui contiennent des bases de données dont parmi les données associées ceux qui sont demandées par l'utilisateur. Pour cela, l'agent mobile déplace vers le Middleware pour donner les besoins d'utilisateur sous forme de liste des demandes, ces dernières sont utilisées par le Middleware pour construire un itinéraire pour l'agent mobile et les différentes requêtes a utilisées. En effet, le Middleware assure à l'agent mobile :

- Un ensemble de site à visiter dans un ordre bien étudié. Qui forme un itinéraire pour l'agent mobile.
- Des requêtes écrites en SQL pour interroger les bases de données qui sont trouvées dans les sites définis dans l'itinéraire.

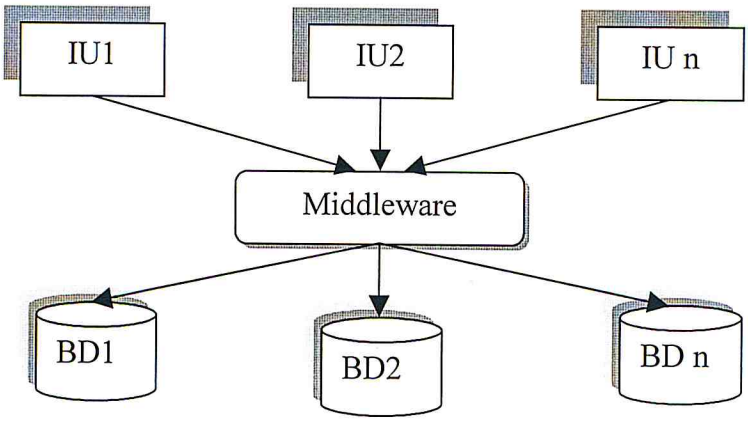



**Figure IV-7 : Middleware.**

### 5.2. Scénario de fonctionnement de système RIAM (coté client) :

L'utilisateur exploite l'interface utilisateur pour exprimer ses besoins qui sont finalement transformeé sous forme d'ensemble des demandes. Il délègue un agent mobile, il lui initialiser avec les demandes d'utilisateurs et l'adresse de Middleware. En suite l'agent mobile déplace vers le Middleware et il lui donne les demandes d'utilisateurs. A ce moment, le Middleware construit un itinéraire pour l'agent mobile avec les différentes requêtes à utiliser. Après que l'agent mobile prend son itinéraire, il doit le suivre. Dans ce moment, l'excursion de l'agent mobile est démarrée dans le réseau.

Dans le cas où l'agent mobile achèverait sa mission, il retourne vers le Middleware puis vers son site d'origine et donner les résultats obtenus à l'interface utilisateur, dans ce cas les informations demandées par l'utilisateur sont disponibles et l'interface d'utilisateur va les rendre lisibles à l'utilisateur.

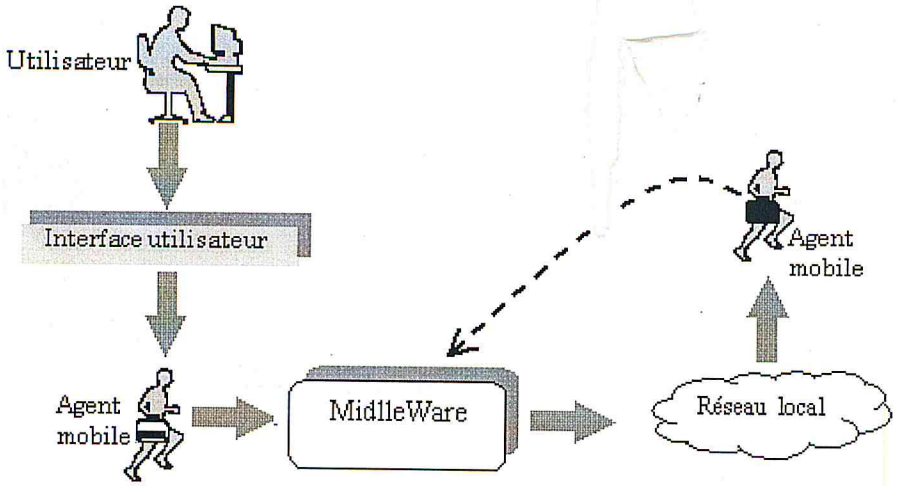



**Figure IV-5 : Conception coté client.**

### 5.3. Scénario de fonctionnement de système RIAM ( coté agence) :

L'agent de ressource (agent statique) est l'intermédiaire entre l'agent mobile et la base de donnée. Dès l'arrivé de l'agent mobile à un site quelconque, l'agent ressource situé dans ce site recevez la requête à exécutée. Ensuite l'agent de ressource reçoit la requête et l'exécuter sur la base de donnée concernée et extrait les informations requissent et envoyer vers l'agent mobile. Ce dernier met les résultat dans son bag et se déplacer vers le site suivant dans son itinéraire pour accomplir sa tache ou bien revient vers son site d'origine (si il est terminer sa tache).

Les requêtes sont écrites en SQL et stockées au niveau de Middleware. D'une façon non complète. Ils sont utilisées par l'agent de ressource avec le nom de table de données pour exprimer les informations demandées par l'utilisateur (SQL utilisé est le SQL standard pour toutes les bases de données).

## 6. Conclusion:

Les bases de données réparties à grande échelle contiennent une quantité d'information importante ce qui rend l'exécution d'une requête répartie coûteuse. Les agents mobiles devient une solution pertinente qui permet de rendre l'exécution d'une requête répartie autonome et permet aussi de décentraliser la décision et éviter le goulot d 'étranglement.

Dans le chapitre suivant nous décrivons une conception de la solution proposée en utilisant UML.

# CHAPITRE V.
# Conception de système RIAM

## 1. Introduction :

Dans ce chapitre, nous utilisons UML (Unified Modeling Language) pour décrire la conception de notre système RIAM dans le chapitre précédent qui montrer qu'avec les agents et les agents mobiles on peut consulter, modifier et supprimer les données d'une ou de plusieurs bases de données même si ces bases de données sont réparties et hétérogènes, si notre système partage les extrémités d'un même réseau avec ces bases de données.

Nous avons utilisé un processus de développement qui est le cycle de vie en cascade, ce modèle décrit par Royce en 1970. Il décrit le cycle de vie d'un logiciel par une suite de phases s'enchaîne dans un déroulement linière (figure V-1). [44]

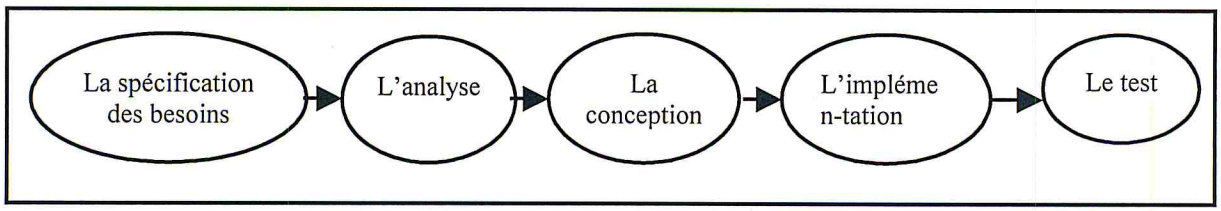



**Figure V-1: Cycle de vie d'un logiciel.[45]**

La notation UML que nous avons utilisée est née de la fusion des trois méthodes qui ont les plus influencées de la modélisation objet au milieu des années 90 : OMT, Booch et OOSE. De ce fait, elle permet de couvrir le cycle de vie d'un logiciel depuis l'analyse du besoin jusqu'au codage. Cette notation est d'une très grande richesse. Elle permet de couvrir à peu près toutes les phases du développement [39]. parmi les avantages d'UML:

- Sa notation graphique permet d'exprimer visuellement une solution objet, ce qui facilite la comparaison et l'évaluation de solutions.
- L'aspect formel de sa notation, limite les ambiguïtés et les incompréhensions.
- Son indépendance par rapport aux langages de programmation, aux domaines d'application et aux processus, en font un langage universel.

Le présent chapitre se divise en deux parties principales : la première décrit une conception globale de notre système et la deuxième partie présente une conception détaillée de notre système et présente brièvement les interactions entre agents.

## 2. Présentation de l'application :

Une des applications les plus importantes dans le domaine des agents mobiles est la recherche d'information. Dans ces applications, des agents se déplacent sur différents sites pour chercher des informations pour leurs clients.

Notre application est dont le but est de chercher des informations sur un ensemble des hôtels dans une ville d'un pays et réserver une chambre dans un hôtel choisi par un client. Dans notre scénario, quatre bases de données doivent être interrogées (figure V-1). La première base de donnée recense des hôtels et permet d'obtenir la liste des hôtels de la ville où le client désire réserver. La deuxième base de donnée est un annuaire et permet d'obtenir des informations sur des hôtels ( numéros de téléphone, fax et adresse de l'un de ces hôtels). La troisième base de donnée est associée à des hôtels où le client peut réserver et entrer ses coordonnées personnelles ( nom, prénom...etc.). Et la dernière est associée à une banque où les hôtels peuvent consulter les comptes bancaires des utilisateurs et faire des mises à jour. Les quatre bases de données sont situées dans des sites différents. Les deux premières bases de données sont gérées par une agence de tourisme, la troisième est gérée par un hôtel et la dernière est gérée par une banque. Les différents serveurs des bases de données fournissent chacun une interface correspondante au service qu'il fournit. La figure suivante illustre une vue générale :

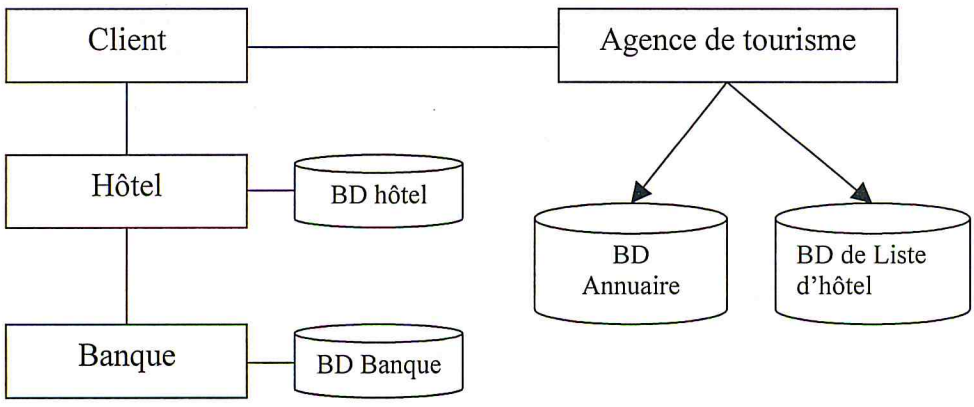



**Figure V-2: Vue générale de l'application.**

## 3. Analyse des besoins :

Dans un processus de modélisation d'un système, la détermination précisée des besoins des utilisateurs de système est la première étape car une bonne informatisation

passe de plus en plus par une vision intégrante, non parcellaire qui s'optimise lorsqu'elle dépasse le point de vue d'une seule catégorie d'utilisateur. [44]
Les besoins d'un utilisateur de notre système sont les suivants :

Les bases de données à interroger sont situées dans plusieurs sites, chaque site contient une ou plusieurs base de donnée. Les bases de données sont hétérogènes au niveau de constructeur (Oracle, SQLserveur, Access…etc.) et au niveau de modèle (BDR, BDOO…etc. Les utilisateurs de systèmes sont autorisés à interroger les bases de données sans faire des mis à jour c'est à dire qu'ils sont autorisés à consulter les données sans les modifier. Les champs des bases de données qui ont l'accès autorisé sont affichés au sein d'une interface graphique afin que l'utilisateur puise spécifier les informations qu'il cherche. L'interface graphique donne à l'utilisateur la possibilité de spécifier une requête englobe n'importe quelle information autorisée. Le résultat de requête doit être affiché et lisible par l'utilisateur. L'utilisateur peut demander les informations suivantes :

- Nom d'hôtel.
- Classe d'hôtel.
- Prix d'une chambre d'un hôtel.
- Ville d'hôtel.
- Pays d'hôtel.

L'utilisateur peut aussi faire une réservation dans un hôtel à travers une interface graphique, cette dernière offre à l'utilisateur la possibilité d'entrer toutes les informations nécessaires pour une réservation qui sont :

- Nom de l'utilisateur.
- Prénom de l'utilisateur.
- Numéro de la carte de crédit de l'utilisateur.
- Nom de l'hôtel.
- Ville de l'hôtel.
- Pays de l'hôtel.
- Date de réservation.
- Durée de réservation.

Ensuite, la validation de réservation et le résultat est affiché. S'il y'a des chambres vides, la réservation est complétée sans problème et un message destiné à l'utilisateur est affiché dont le contenu est que la réservation est complétée avec succès, sinon le contenu de message affiché et que la réservation n'est pas validée a cause de l'indisponibilité des chambres vides.

### 3.1. Les cas d'utilisation :

Les diagrammes de cas d'utilisation représentent les cas d'utilisation, les acteurs et les relations entre les cas d'utilisation et les acteurs. Un cas d'utilisation est une manière de décrire comment utiliser le système. C'est l'image d'une fonctionnalité du système, déclenchée par un acteur.[44]

### 3.1.1. Les acteurs :

Ce sont les catégories d'utilisateur (humain ou systématique) qui interagit avec notre système (les acteurs sont à l'extérieur de système), les acteurs de notre système sont les suivants :

**Acteur principal :**

- L'utilisateur : c'est pour lui que ce système est fabriqué, l'utilisateur peut interroger la base de donnée et peut aussi faire une réservation d'une chambre.

**Autres systèmes :**

- Agence de tourisme : c'est un système avec lequel notre système interagit, il lui fournit des informations sur les hôtels.
- Annuaire : c'est un système avec lequel notre système interagit, il lui fournit des informations sur les hôtels.
- La banque : c'est un système avec lequel notre système interagit, il lui fournit le solde de l'utilisateur qui veut réserver une chambre dans un hôtel. Et faire le transfert d'argent de compte de l'utilisateur vers le compte de l'hôtel afin de compléter la réservation.
- Hôtel : c'est un système avec lequel notre système interagit, c'est la où l'utilisateur peut réserver une chambre.

La figure V-3 présente leurs présentations graphiques :

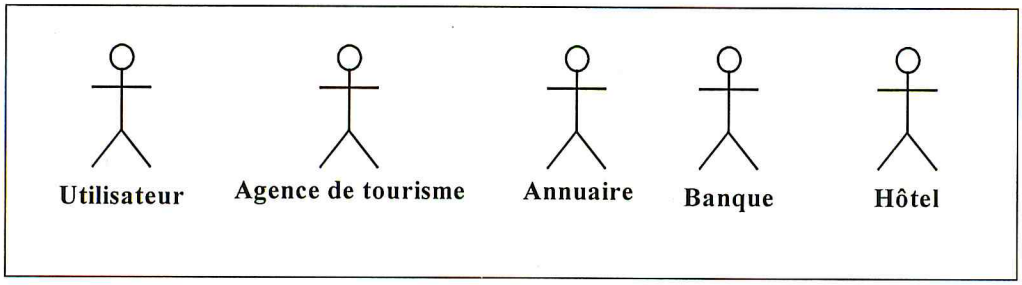



**Figure V-3: Représentation des catégories des acteurs.**

### 3.1.2. Détermination des cas d'utilisation :

« interrogation des bases de données »

- ✓ L'utilisateur doit sélectionner les informations qui représentent un besoin pour lui.
- ✓ L'utilisateur spécifie ses besoins à travers une interface graphique
- ✓ L'utilisateur, et à travers l'interface graphique peut sélectionner n'importe quelle information parmi ceux qui ont l'accès autorisé.

- ✓ Les informations à interroger sont : le nom de l'hôtel, la classe de l'hôtel, le téléphone de l'hôtel, le prix d'une chambre d'un hôtel, la ville de l'hôtel et le pays de l'hôtel.
- ✓ Les informations demandées par l'utilisateur doivent être affichées à l'utilisateur.

On peut représenter le cas d'utilisation d'interrogation des bases de données par le diagramme suivant :

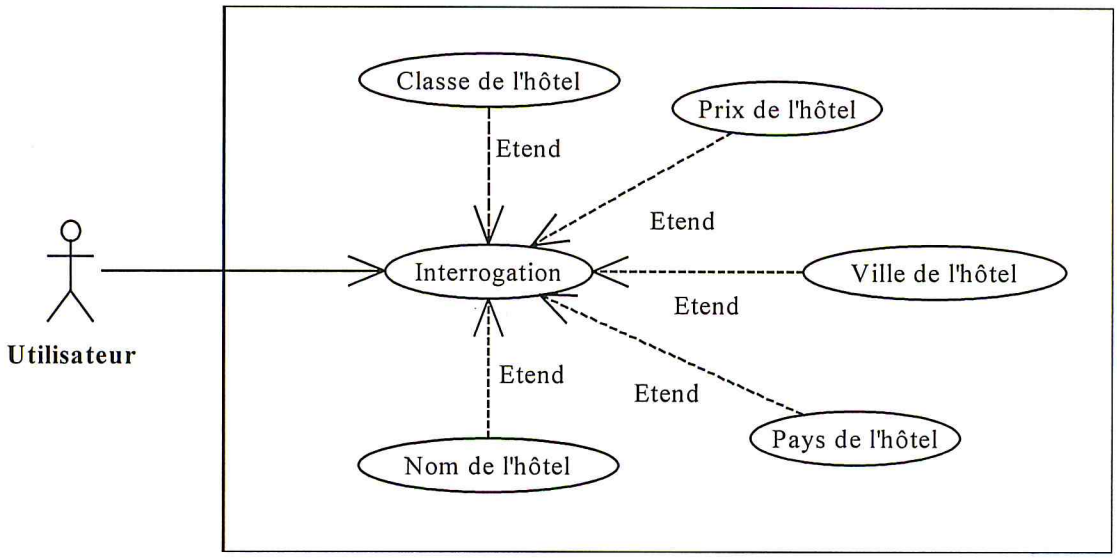



**Figure V-4: Diagramme des cas d'utilisation de l'interrogation des bases de données.**

« réservation »

- ✓ L'utilisateur doit saisir les informations nécessaires pour valider la réservation. Les informations à saisies sont : le nom de l'utilisateur, le prénom de l'utilisateur, numéro de carte crédit, le nom de l'hôtel, ville de l'hôtel, pays de l'hôtel, date de réservation et la durée de réservation.
- ✓ Un message doit être affiché dont le contenu est le résultat de la réservation : validée ou non.

La représentation des cas d'utilisation de réservation est la suivante :

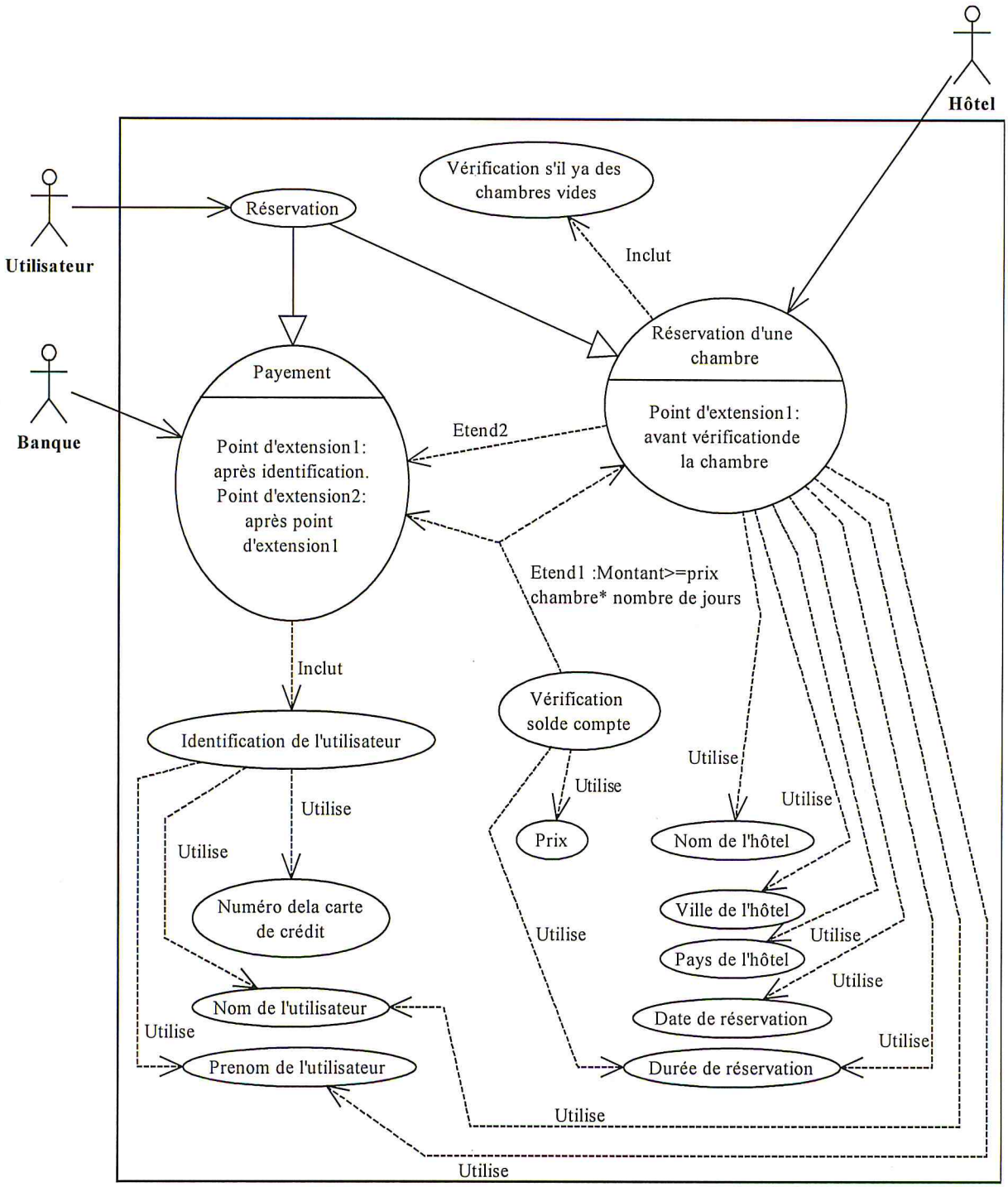



**Figure V-5: Diagramme des cas d'utilisation de la réservation d'une chambre vide.**

Et finalement, le diagramme de cas d'utilisation de l'application :

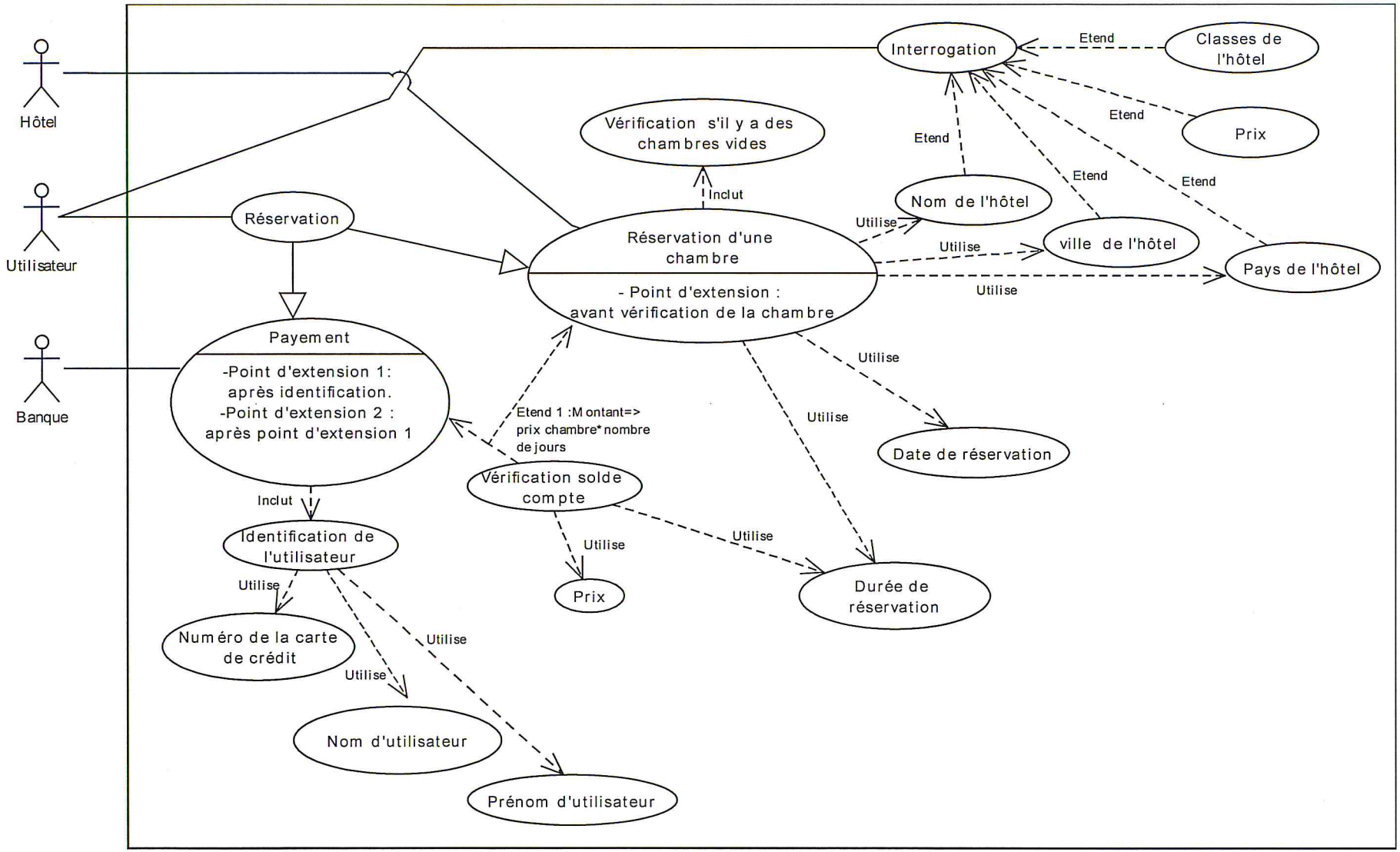



**Figure V-6 : Diagramme de cas d'utilisation de l'application**

# 4. Conception de l'application :

## 4.1. Diagrammes de séquence :

Les diagrammes de séquence nous permettent de montrer les interactions entre objets, ils nous permettent de bien schématiser les scénarios des cas d'utilisation.

La représentation se concentre sur la séquence des interactions selon un point de vue temporel. Ils sont en général, plus aptes à modéliser les aspects dynamiques des systèmes temps réel et des scénarios complexes mettant en ouvre peu d'objets.[44]

Une interaction modélise un comportement dynamique entre objets. Elle se traduit par l'envoi de message entre objets. Un diagramme de séquence représente une interaction entre objet, en insistant sur la chronologie des envois de message.[44]

Un objet est matérialisé par un rectangle et une barre verticale, appelée ligne de vie des objets [44], cette barre verticale représente l'écoulement de temps (de haut en bas). Les objets communiquent en échangeant des messages représentés au moyen de flèches horizontales, orientées de l'émetteur du message vers le destinataire. L'ordre d'envoi des messages est ainsi donné par la position de ces messages sur les lignes de vie des objets.

**Scénario 1 : « Interrogation » :**

→ L'utilisateur entrer les informations qu'il cherche.
→ Le système fait transférer les informations cherchées au Middleware.
→ Le Middleware localise les informations cherchées.
→ Le Middleware génère les requêtes dont les résultats sont les informations cherchées.
→ Le Middleware envoie pour chaque poste les requêtes associées.
→ Chaque poste exécute ses requêtes et envoi les résultats vers le système.
→ Le système rassemble tous les résultats.
→ Le système affiche les résultats.

La figure suivante illustre le diagramme de séquence de l'interrogation :

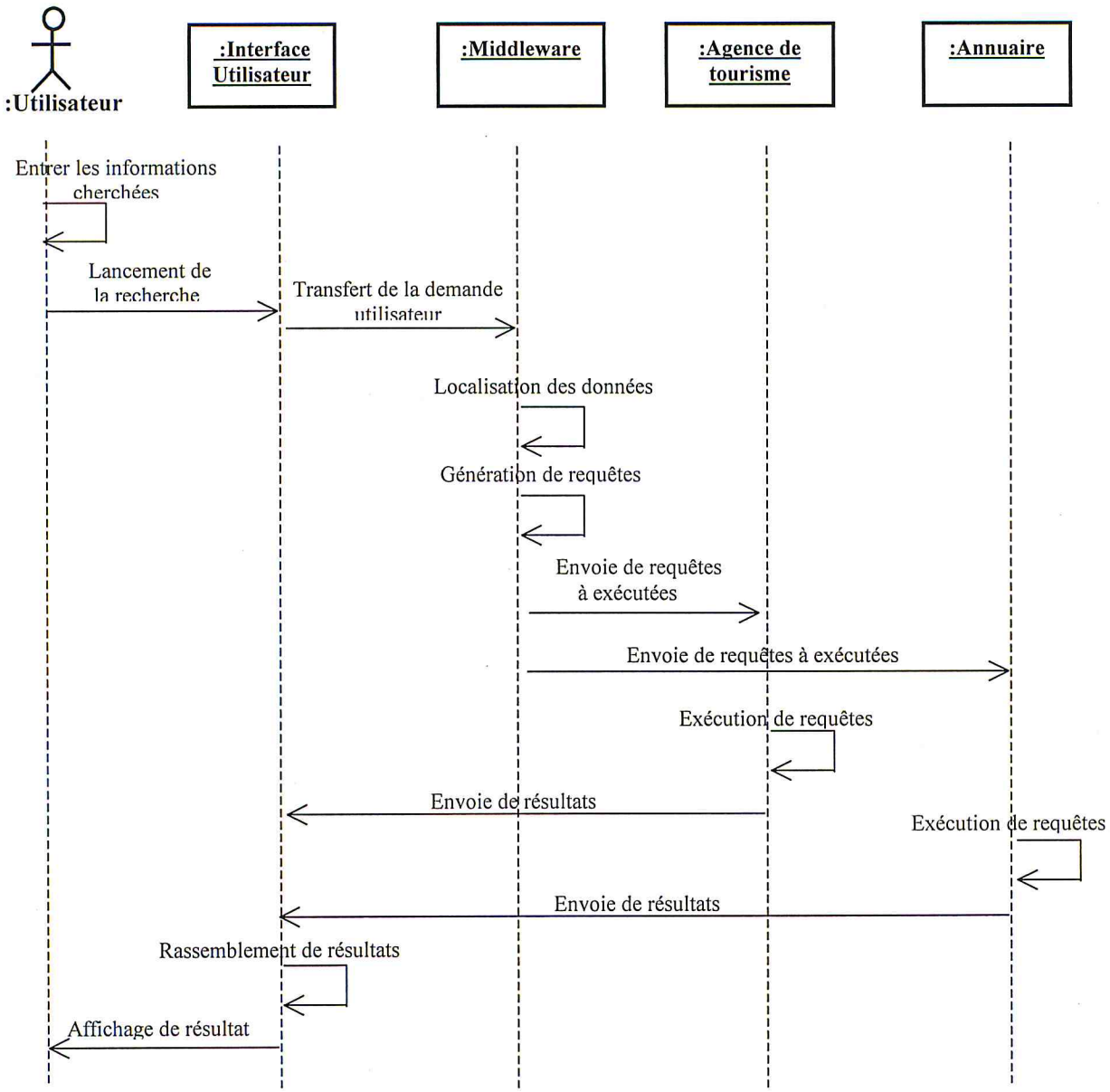



**Figure V-7:Diagramme de séquence d'interrogation des bases de données.**

**Scénario 2 : « Réservation »:**

→ L'utilisateur entrer les informations nécessaires pour une réservation.
→ Le système envoi les informations de l'utilisateur vers le Middleware.
→ Le Middleware localise l'hôtel qui l'utilisateur veut réserver une de ses chambres.
→ Le Middleware envoi les informations de l'utilisateur vers la banque.
→ La banque identifie l'utilisateur.
→ La banque vérifie si la somme d'argent de l'utilisateur suffit pour la réservation de la durée demandée.
→ La banque envoi les informations de l'utilisateur vers l'hôtel.
→ L'hôtel cherche une chambre vide.
→ L'hôtel réserve une chambre vide.
→ L'hôtel envoi un message vers la banque dont le contenu est chambre réservée.
→ La banque fait le transfert d'argent de compte de l'utilisateur vers le compte de l'hôtel.

→ La banque envoi un message au système dont le contenu est réservation complète.
→ Le système envoi un message à l'utilisateur dont le contenu est que la réservation est complétée avec succès.

La Figure suivante montre le diagramme de séquence de la réservation :

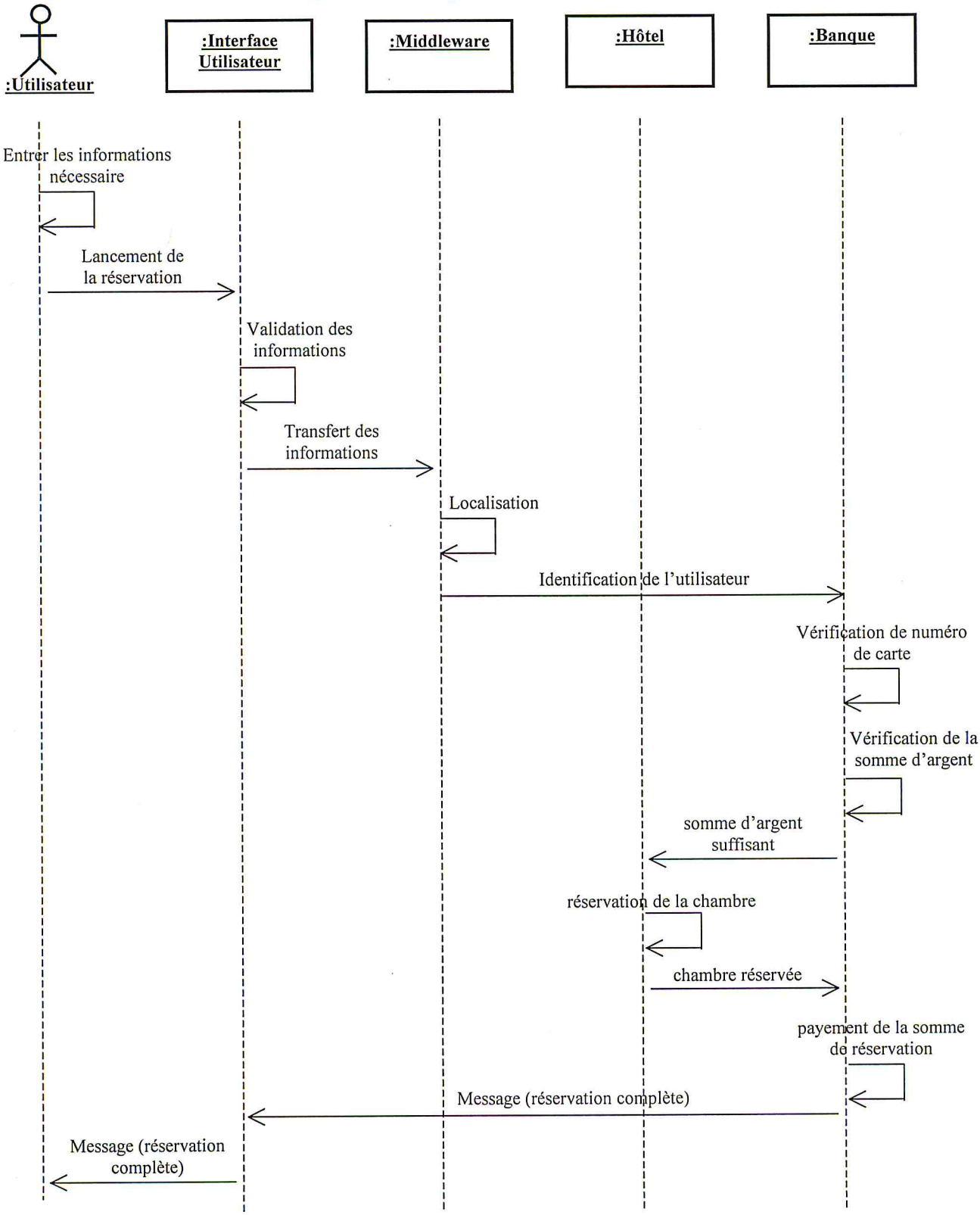



**Figure V-8: Diagramme de séquence de réservation d'une chambre vide.**

### 4.2. Diagrammes de collaboration :

Les digrammes de collaboration (tout comme les diagrammes de séquence) représentent une vue dynamique de système. Ils montrent également les interactions entre objets à travers la représentation d'envois de messages. L'ajout d'une dimension temporelle requiert la définition de numéros de séquence pour les messages

Une collaboration définit les éléments qui sont utiles pour l'obtention d'un objectif particulier en spécifiant le rôle de ces éléments dans un contexte de la collaboration[44].

Le premier message de l'interaction est envoyé par l'acteur, représenté par le symbole graphique des acteurs du modèle des cas d'utilisation. Un message se présente par une flèche placée à proximité d'un lien et dirigée vers l'objet destinataire du message au sein de l'interaction. En général, une séquence dans un diagramme de collaboration commence par le chiffre 1. Les numéros de séquence suivants sont incrémentés (2, 3…).

Le diagramme de collaboration suivant représente l'interrogation des bases de données :

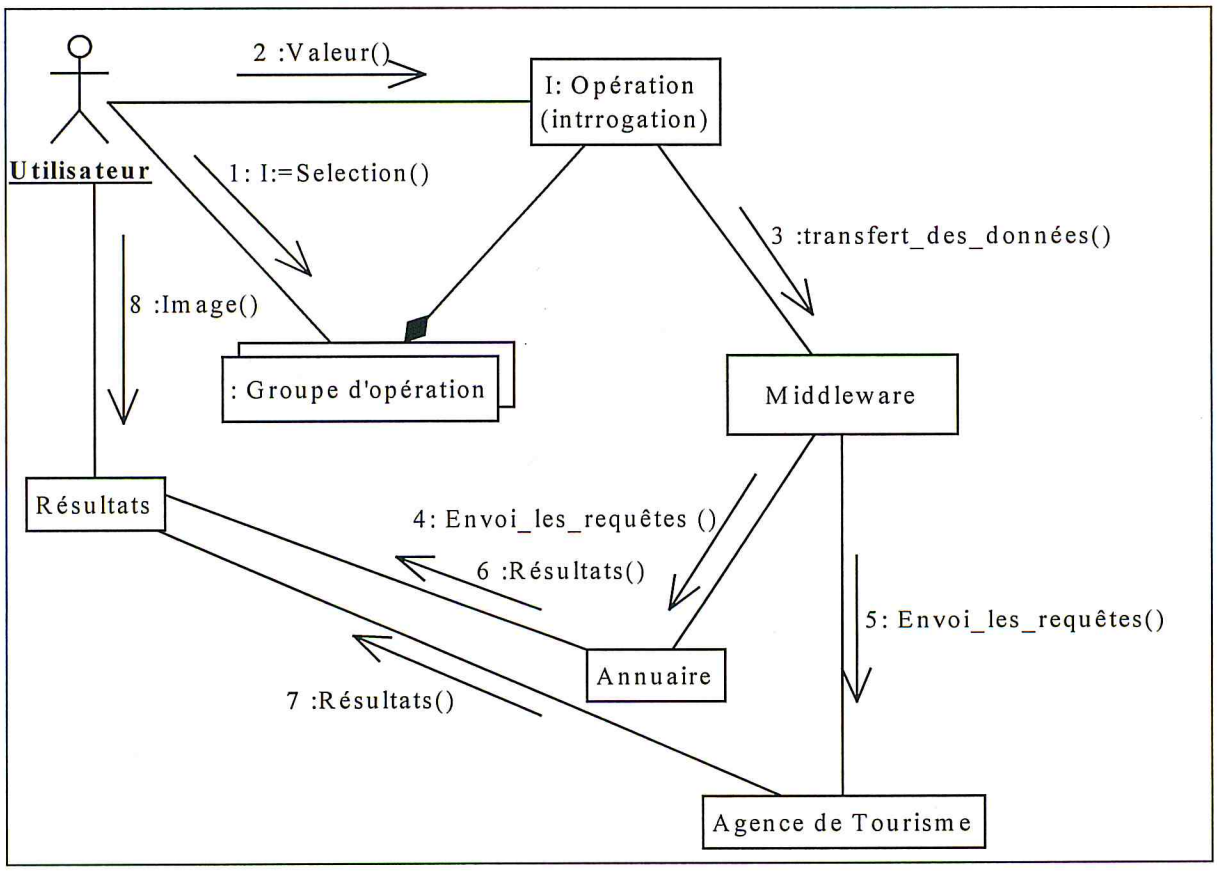



**Figure V- 9: Diagramme de collaboration de l'interrogation des bases de données.**

Les conventions utilisées ici sont :

• Les opérations **`Image()`** et **`Valeur()`** prennent en charge la lecture et l'écriture de l'état d'un objet. L'opération **`Valeur()`** permet à un objet d'interface de transférer des informations en provenance d'un utilisateur vers un objet de domaine.

Inversement, l'opération **`Image()`** permet à un objet d'interface d'extraire les informations contenues dans un objet du domaine afin de les montrer aux utilisateurs.

• La sélection d'un objet parmi un groupe d'objet se fait par l'intermédiaire d'une valeur retournée par l'opération de sélection. La représentation séparée du groupe d'objet et de l'objet sélectionné dans ce groupe est réalisé par une composition entre les deux éléments.

• L'opération **`résultat()`** affecte les résultats de l'interrogation (dans le cas de l'interrogation) des bases de données au interface utilisateur chargée d'afficher les résultats de l'interrogation au utilisateur. Dans le deuxième cas (la réservation d'une chambre vide dans un hôtel) cette opération affecte le résultat de la réservation (acceptée ou non) a l'interface utilisateur chargée d'afficher les résultats de réservation a l'utilisateur.

La figure suivante représente le diagramme de classe préliminaire qui correspond au diagramme de collaboration précédent.

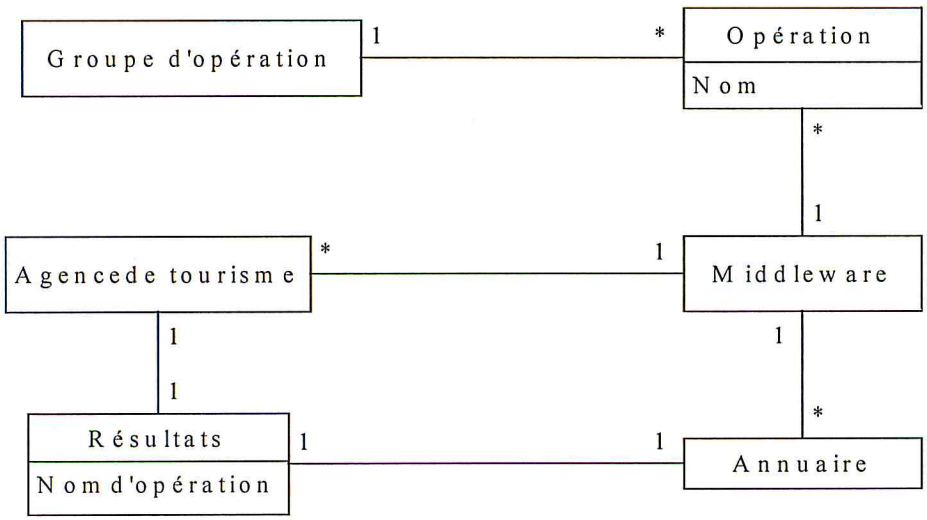



**Figure V-10: Ébauche de diagramme de classe.**

La réservation d'une chambre vide dans un hôtel est représentée par la collaboration suivante :

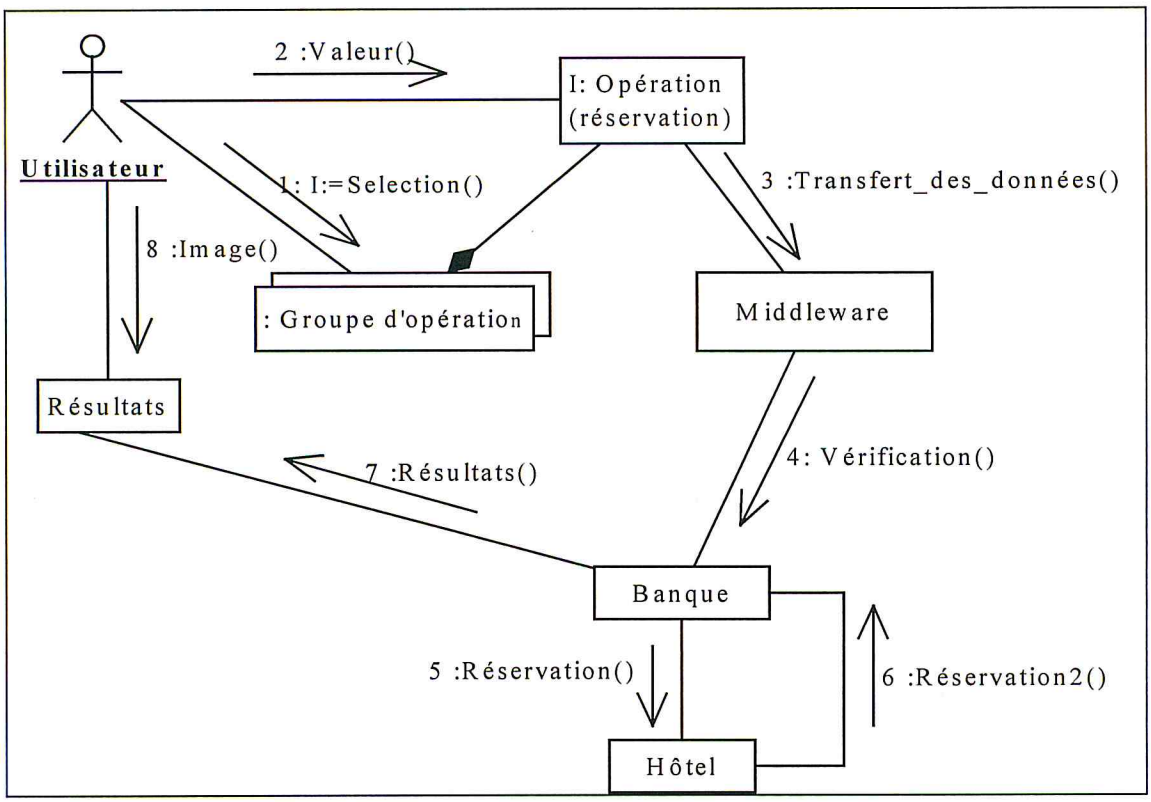



**Figure V-11: Diagramme de collaboration de réservation d'une chambre vide.**

• L'opération **`Vérification()`** est chargée de la vérification du numéro de la carte de crédit de l'utilisateur et la somme d'argent si elle est suffisante pour effectuer la réservation d'une chambre vide pour la durée demandée.

• L'opération **`Réservation1()`** prenne en chargée la réservation d'une chambre vide dans l'hôtel souhaité, et **`Réseravtion2()`** prenne en charge le transfert d'argent de l'utilisateur vers le compte bancaire de l'hôtel.

L'ébauche de diagramme de classe est ainsi :

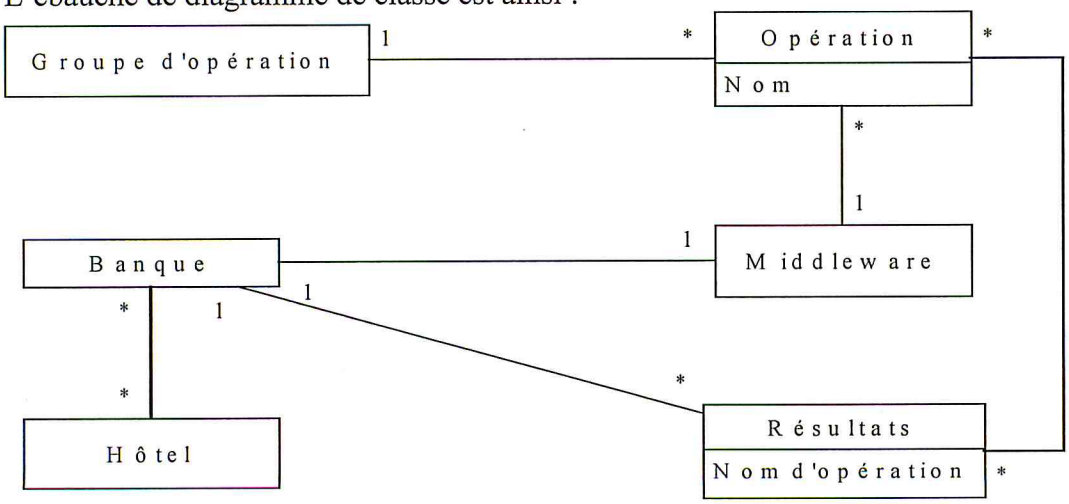



**Figure V-12: Ébauche de diagramme de classe.**

Le diagramme de collaboration du système est représenté par la figure suivante :

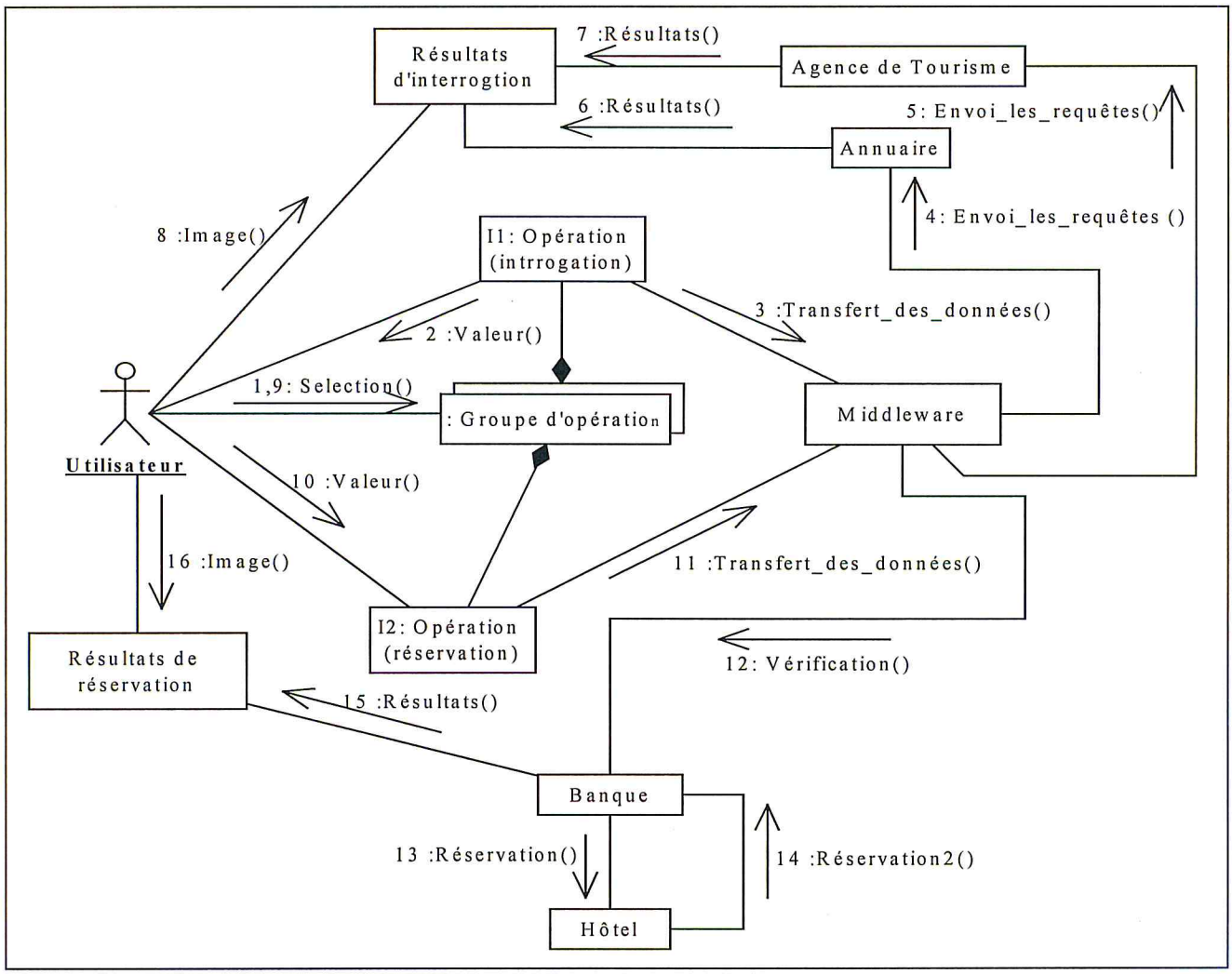



**Figure V-13: Diagramme de collaboration du système.**

### 4.3. Diagramme de classe :

Le diagramme de classe exprime de manière générale la structure statique d'un système, en terme de classe et de relations entre ces classes. Le diagramme de classes suivant représente les éléments qui doivent être implémentés dans notre système :

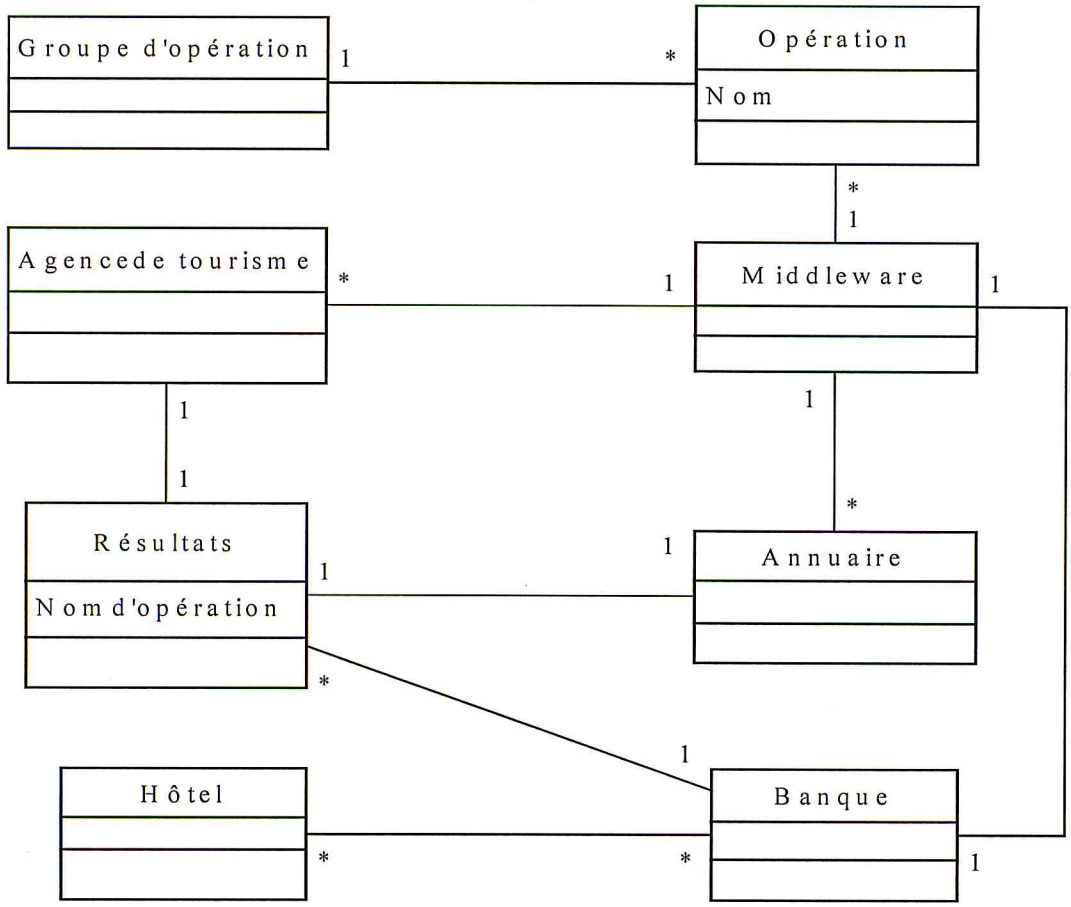



**Figure V- 14: Diagramme de classe du système.**

## 5. Conception de système RIAM :

Dans cette section, on va entrer dans les détails de la conception du système RIAM dans et on décrit comment les agents statiques et les agents mobiles vont coopèrent entre eux afin de réaliser le but commun.

### 5.1. L'interrogation :

L'utilisateur lance cette opération pour chercher des informations sur les hôtels, ces informations sont fixées selon les données entrées par l'utilisateur à travers une interface graphique. Pour lancer l'opération d'interrogation il faut que le poste de l'utilisateur contienne l'outil Agence, cette dernière est en effet un serveur agent. Le serveur agent est un environnement d'exécution d'agent. Après le lancement de l'opération d'interrogation, le serveur agent crée un agent mobile dont l'objectif est de chercher les informations demandées par l'utilisateur dans le réseau. Au début, l'itinéraire de l'agent mobile ne contient que le poste de Middleware, donc m'agent mobile déplace vers ce poste. A l'arrivée, l'agent mobile envoi au Middleware un message dont le contenu est les informations cherchées par l'utilisateur pour qu'il puisse générer un itinéraire et décrit des sous requêtes à l'intérêt de l'agent mobile afin que ce dernier satisfait la demande utilisateur en suivant l'itinéraire généré par le Middleware. Chaque poste appartienne à l'itinéraire de l'agent mobile doit possèder une agence constituée d'un serveur agent et un agent statique. Le serveur agent pour accueillir et envoyer l'agent mobile et l'agent statique pour exécuter les sous requêtes associées à ce poste et envoyer

les résultats vers l'agent mobile. Après que l'agent mobile visite tous les postes appartenant à son itinéraire, il revient vers le site qui contient le Middleware ensuite le retour vers son site natal. Lorsque l'agent mobile revient à son site natal il sera détruit après qu'il envoi un message vers le serveur agent dont le contenu est les résultats de l'interrogation afin que le serveur agent affiche ces résultats.

### 5.1.1. Diagramme de séquence :

Scénario :

→ L'utilisateur entre les informations qu'il cherche.
→ Le serveur agent crée un agent mobile.
→ L'agent mobile se déplace vers le site de Middleware portant avec lui les données demandées par l'utilisateur.
→ L'agent mobile envoi au Middleware un message dont le contenu est les données demandées par l'utilisateur.
→ Le Middleware génère un itinéraire pour l'agent mobile.
→ Le Middleware génère les sous requêtes.
→ L'agent visite les sites appartenant à son itinéraire.
→ L'agent mobile revient au site qui contient le Middleware.
→ L'agent mobile fait des mises à jours dans le module de méta-donnée.
→ L'agent mobile revient à son site natal.
→ Le serveur agent extrait le résultat et détruit l'agent mobile.
→ Le serveur agent affiche ce résultat.

La figure suivante illustre le diagramme détaillé de l'interrogation :

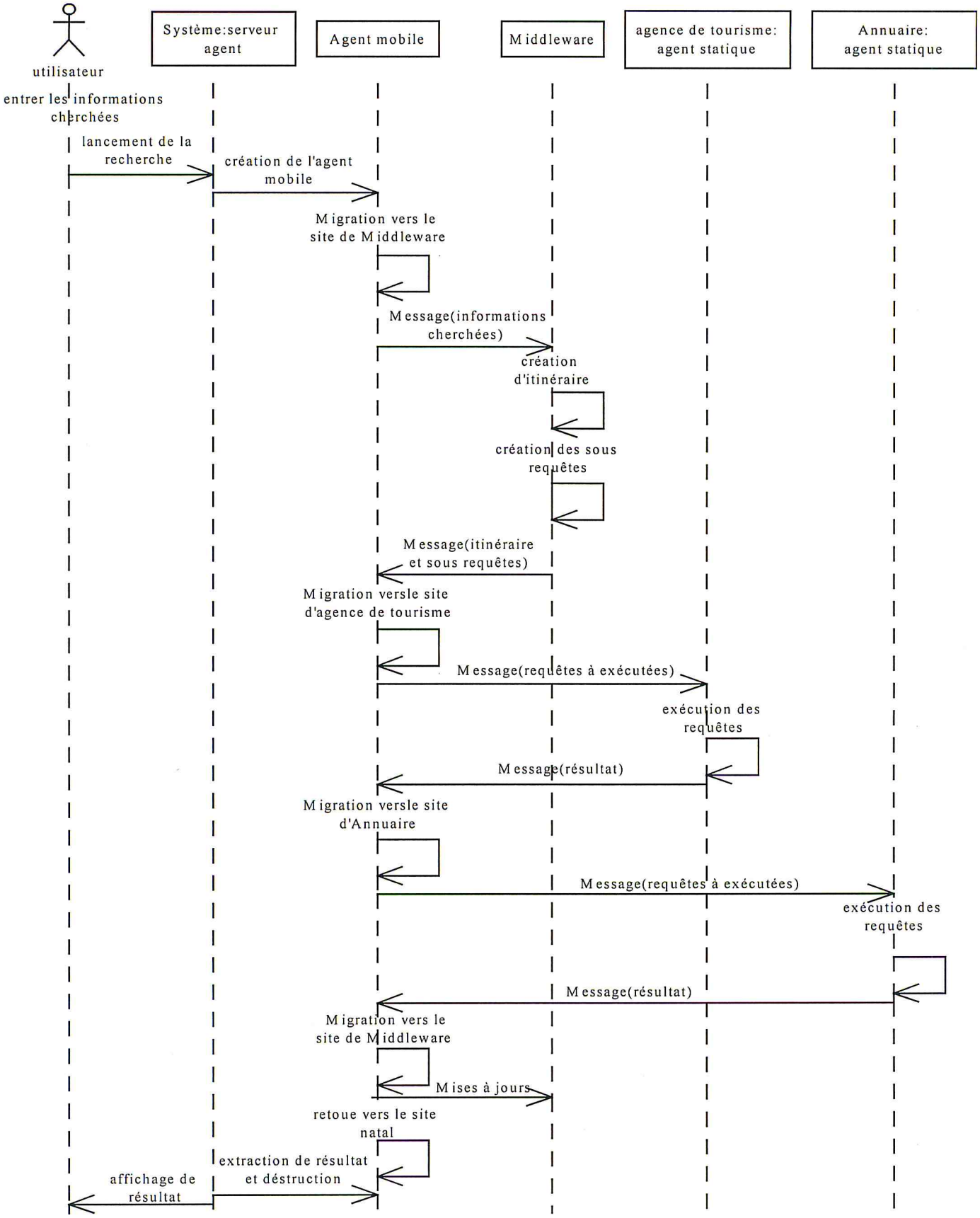



**Figure V-15: diagramme de séquence détaillée de l'interrogation.**

### 5.1.2. Diagramme de classe :

La figure suivante présente le diagramme de classe de la réservation :

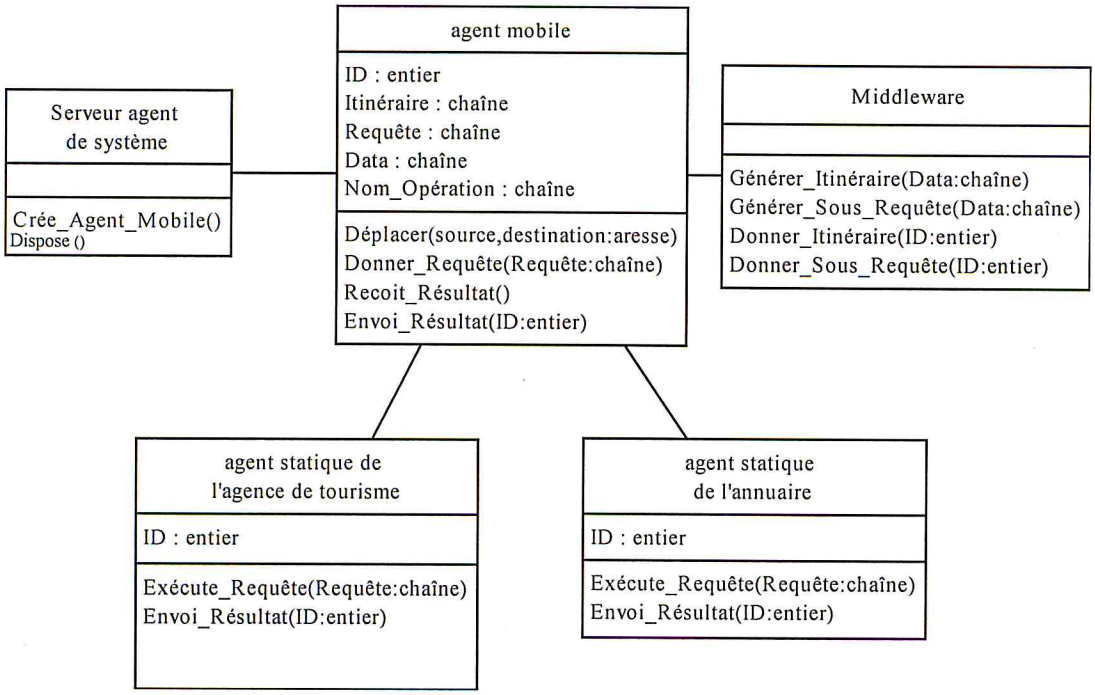



**Figure V-16: Diagramme de classe de l'interrogation.**

## 5.2. La réservation :

L'utilisateur effectue cette opération pour réserver une chambre vide dans un hôtel, pour réaliser cette opération l'utilisateur doit remplir ses informations personnelles à travers un formulaire. Lorsque l'utilisateur valide ses informations personnelles, le déroulement de l'opération commence par la création d'un agent par le serveur agent dont l'objectif est d'achever la réservation. L'itinéraire de l'agent mobile crée ne contient au début que le site qui contient le Middleware donc il va déplacer vers ce site et à l'arrivée l'agent mobile envoi un message au Middleware pour qu'il puisse localiser l'hôtel destiné. Dans ce cas le Middleware génère un itinéraire et des sous requêtes et en voyer l'itinéraire et les sous requêtes vers l'agent mobile. L'agent mobile commence son excursion en suivant l'itinéraire généré par le Middleware, l'itinéraire dans le cas de réservation est les suivant : au début, le site de banque pour identifier l'utilisateur et vérifier son numéro de carte de crédit, ensuite le site de l'hôtel destiné pour vérifier s'il y à une chambre vide, si le cas l'agent statique de ce poste va réserver cette chambre à l'intérêt de l'utilisateur et retourne vers le site de la banque pour faire le transfert d'argent de compte de l'utilisateur vers le compte de l'hôtel ensuite le retour vers le site de Middleware et enfin le retour ver le site natal.

### 5.2.1. Diagramme de séquence :

scénario

→ L'utilisateur entre ses informations personnelles.

→ Le serveur agent crée un agent mobile.
→ L'agent mobile se déplace vers le site de Middleware portant avec lui les information personnelles de l'utilisateur.
→ L'agent mobile envoie au Middleware un message dont le contenu est les informations personnelles de l'utilisateur.
→ Le Middleware génère un itinéraire pour l'agent mobile.
→ Le Middleware génère les sous requêtes.
→ L'agent visite les sites appartenant à son itinéraire.
→ L'agent mobile revient au site qui contient le Middleware.
→ L'agent mobile revient à son site natal.
→ Le serveur agent extrait le résultat et détruit l'agent mobile.
→ Le serveur agent affiche ce résultat.

La figure suivante illustre le diagramme détaillé de la réservation :

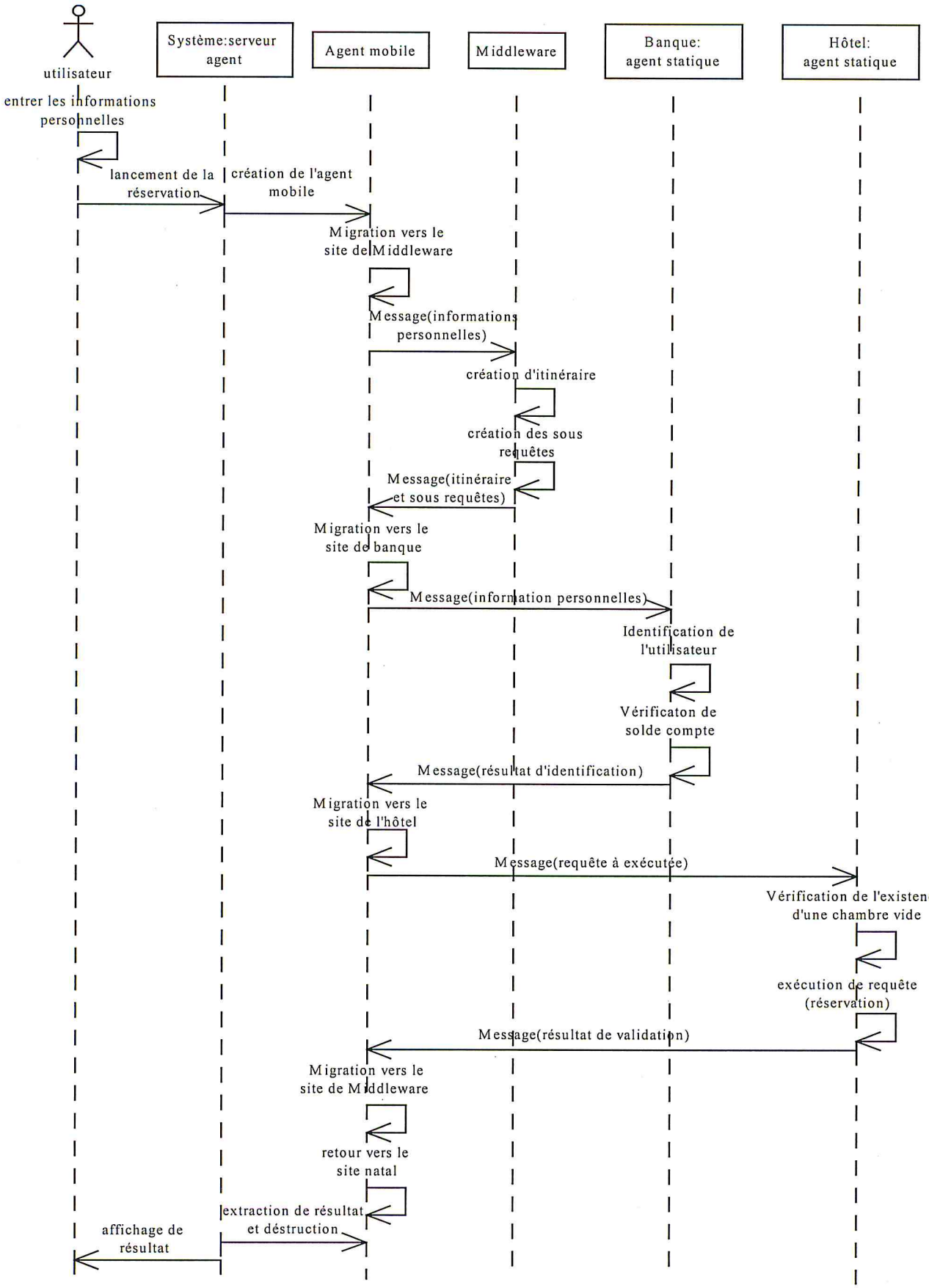



**Figure V-17: diagramme de séquence détaillée de la réservation.**

### 5.2.2. Diagramme de classe :

La figure suivante présente le diagramme de classe de la réservation :

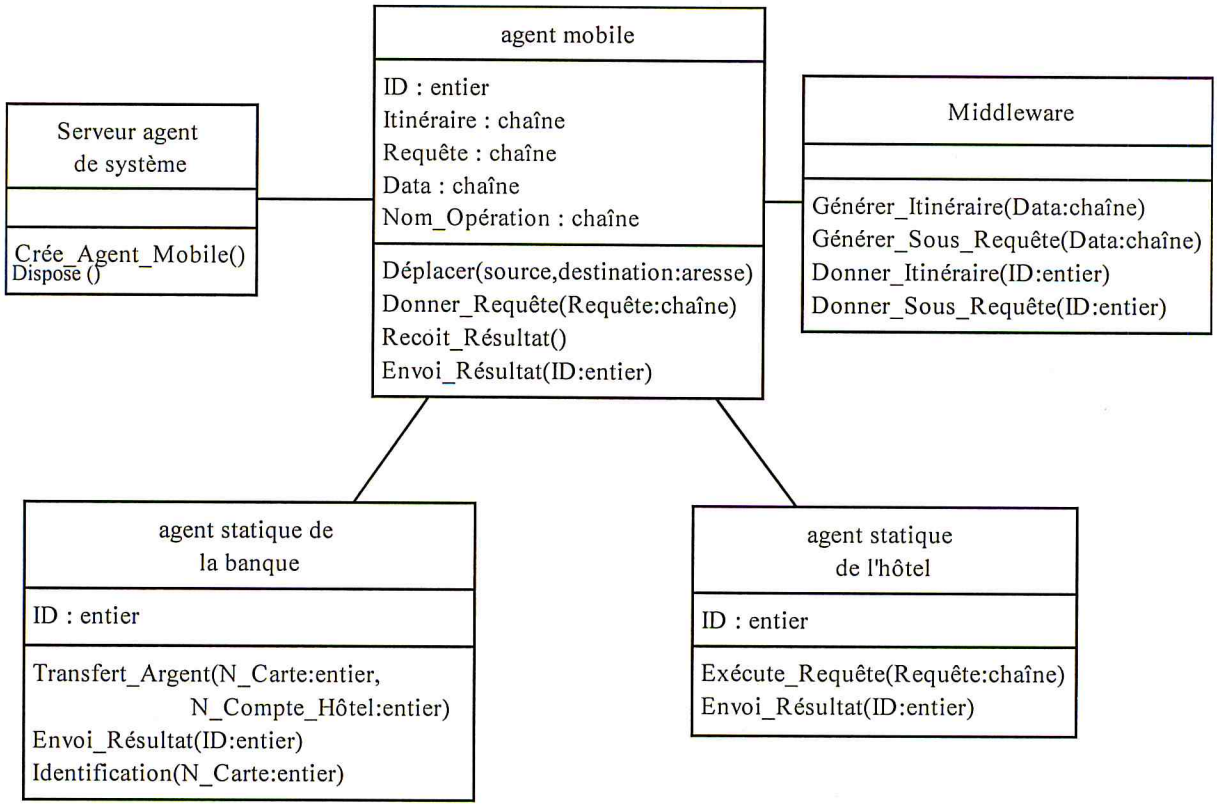



**Figure V-18: Diagramme de classe de l'interrogation.**

### 5.3. Diagramme de classe détaillé :

La figure suivante illustre le système RIAM globalement à travers un diagramme de classe détaillée :

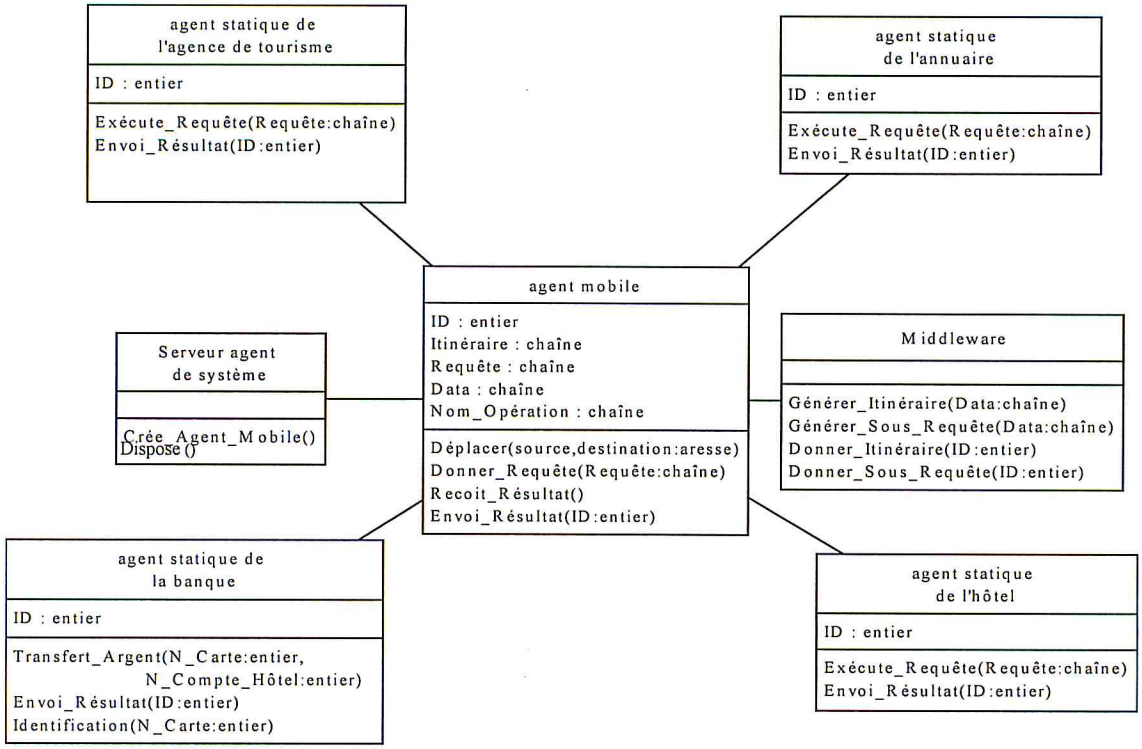



**Figure V-19: Diagramme de classe global.**

## 6. Conclusion :

Dans ce chapitre, nous avons présenté une conception de l'application et une conception du système RIAM posé dans le chapitre précédent.

De plus, par l'intermédiaire de plusieurs diagrammes, nous avons essayé de bien expliquer la solution proposée tout en illustrant l'ensemble de ses composants et leur manière d'agir, après la modélisation de la solution proposée, on passe maintenant à la réalisation, ce qui est proposé dans le chapitre suivant.

# CHAPITRE VI.
# Implémentation de système RIAM

## 1. Introduction :

Dans ce chapitre, nous commençons par une présentation d'une proposition d'une mise en œuvre de notre architecture définie dans le chapitre précédent. Cette architecture est constituée des agents statiques et mobiles plus d'un Middleware: Les agents statiques sont principalement les agents de ressource. Les agents mobiles sont les agents délégués par les utilisateurs. Et le Middleware est utilisé pour que l'agent mobile trouvé son itinéraire et ne pas visité des sites inutilement dans un ordre bien spécifié selon les données entrées par l'utilisateur.

En suite nous décrivons l'implémentation de notre système **RIAM** (**R**echerche d'**I**nformation avec les **A**gent **M**obile). En passant par une architecture globale de notre système, puis nous détaillons les différents modules. En fin nous terminons par exposé des résultats expérimentaux de tests de notre système dans diverses conditions d'utilisation.

## 2. Modèle à agents mobiles :

Plusieurs systèmes à agent mobile ont été développés ces dernières années. Parmi eux les systèmes agentTcl[1], Telescript[2], Aglets[3]… Etc. Une grande partie de ces systèmes sont construits sur l'environnement java, principalement en raison de sa forte diffusion dans le monde, mais aussi de ses avantages techniques : masquage de l'hétérogénéité des machines et typage fort pour la sécurité. Nous rappelons tout d'abord les fonctions générales de systèmes à agents mobiles.[22]

### 2.1. Fonctions des agents mobiles :

Un agent est un processus pouvant se déplacer de machine en machine afin de réaliser une tache. En générale, la mobilité est fournie par le biais d'une primitive

*dispatch* (machine) qui permet de se déplacer vers la machine désignée par le paramètre.

Un agent est composé d'un code correspondant à un algorithme[22], ainsi que d'un contexte incluant des données. Ce contexte peut évoluer en cours d'exécution, par exemple en collectant des données lorsqu'un agent réalise une recherche d'information sur un ensemble de serveurs. Le code et le contexte de l'agent sont déplacés avec l'agent lorsque celui-ci visite différente site.

Lorsqu'un agent se déplace vers un agence , il doit poursuivre son exécution sur le site destination. La plupart des systèmes à agent mobile implantent une migration faible, c'est à dire une fonction de migration où l'agent redémarre son exécution depuis le début.

Un système à agent fournit en général des primitives de communication permettant aux agents d'interagir entre eux, mais aussi d'interagir avec les systèmes qu'ils visitent. Ces primitives de communication prennent la forme d'envoi de messages ou d'appels de procédures. [5]

## 3. Choix du langage de programmation:

Lorsque la technologie des agents sera bien établie, on pourra s'attendre à voir une multitude d'outils logiciels pour la conception des systèmes basés sur les agents. Plusieurs langages destinés à la conception d'agent sont en train d'être développés, mais beaucoup de ces langages sont surtout destinées à mettre en évidence certains principes et contraintes, et ne sont pas encore destinées à devenir des standards ( voir chapitre 1). Java est considéré comme étant le langage le plus approprie pour la création des agents parce qu'il dispose d'une machine virtuelle qui procure l'indépendance du système d'exploitation et disponible sur la plupart des systèmes actuels. [4]

Java est un langage de programmation Orienté-Objets qui permet de développer des programmes que l'on peut exécuter sur la machine virtuelle java (JVM). Ce langage offre des mécanismes qui permet la mobilité des agents, les principaux mécanismes sont les suivants [5] :

### 3.1. Mobilité et portabilité du code :

Une des propriétés essentielle de java est que le code généré par le compilateur java est mobile. Ceci signifie que le code peut être déplacé entre différentes machines. La mobilité du code nécessite que le code soit portable. La portabilité du code produit par java provient du fait que le code (appelé *byte code*) est interprété par la JVM.

### 3.2. Liaison dynamique :

Un aspect crucial pour l'utilisation de code mobile est la liaison dynamique. Cette dernière signifie ici la possibilité de ne pas déterminer que lors de l'exécution le code qui sera exécuté pour un appel de méthode. Etant donné que java permet le chargement dynamique des classes, une variable d'un type donné (une interface en

Java) peut contenir une référence Java pointant sur une instance dont la classe à été chargée dynamiquement. Java retarde la liaison du code à partir de cette variable jusqu'à l'appel effectif, permettant ainsi l'exécution de code chargé dynamiquement.

### 3.3. Sérialisation :

Java fournit un mécanisme de sérialisation d'objets permettant d'échanger les instances entre des JVM différentes. Ce mécanisme permet de traduire un graphe d'objets Java en un flot d'octets qui peut être redirigé vers un fichier ou vers une autre machine à travers le réseau.

Le langage Java étant le langage que nous avons adopté pour l'implémentation de notre architecture ainsi que comme langage de description du code de nos agents. Et pour implémenter le comportement spécifique des agents, nous avons augmenté l'API (Application Program Interface) java standard avec l'API (Aglets) de manipulation des agents et les APIs de manipulation de base de donnée.

Une API (Aglets voir annexe) qui consiste en un ensemble de classes java, permet l'instantiation, une fois pour toute, des agents de ressources pendant la phase d'initialisation du système et une création plus dynamique des agents mobile au cours du fonctionnement du système.

Nous utilisons des APIs de manipulation les déférents types de base de donnée (MySQL, Oracle, SQLServer, Access …etc.).

## 4. Représentation générale du système :

L'infrastructure que nous proposons consiste en un environnement d'exécution d'agent destiné à supporter le code des agents mobiles, des agents de ressources (statiques) crées en cours du fonctionnement du système. Et les bases de données. La figure V-I représenté une représentation générale de notre système.

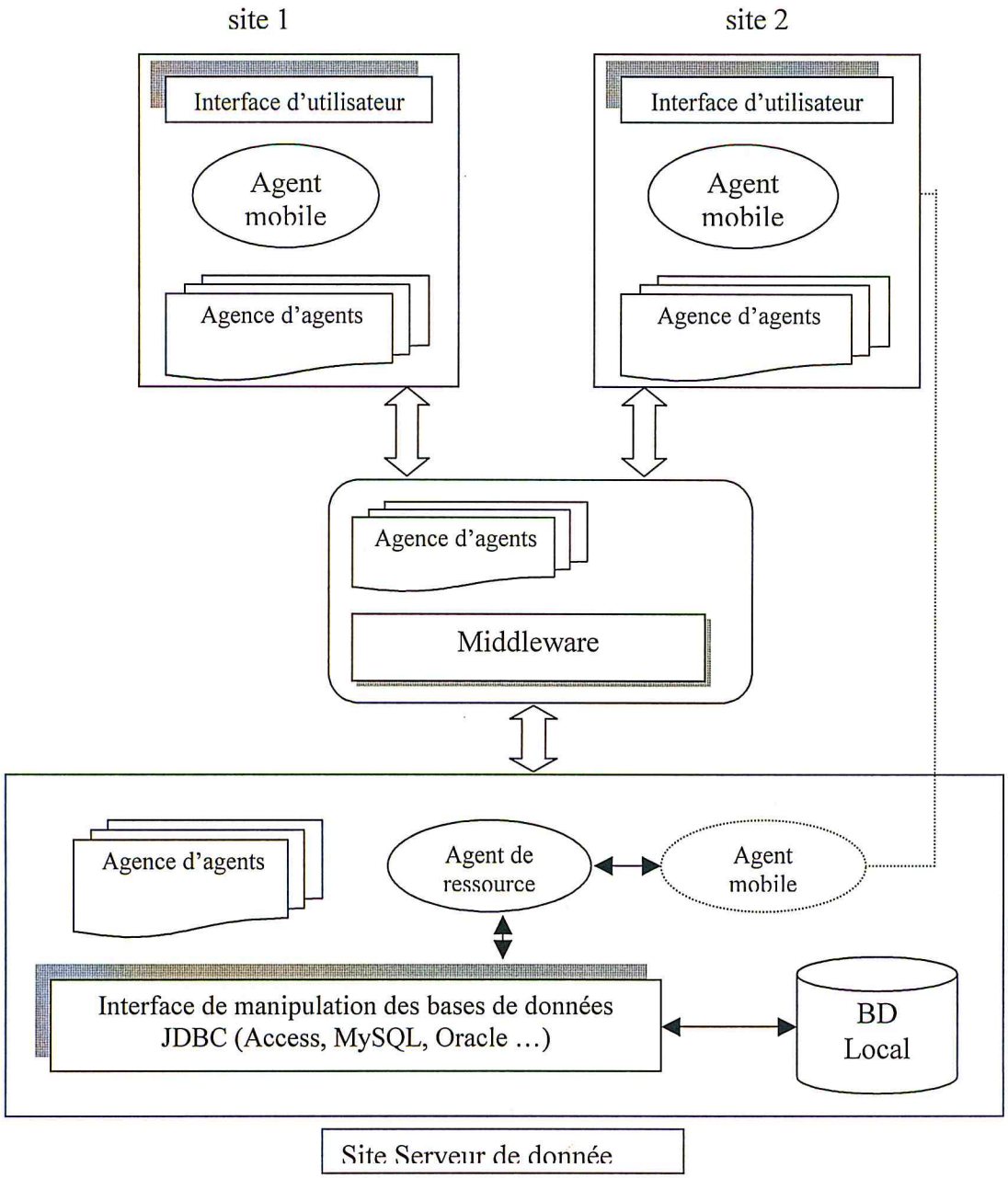



**Figure VI-1: Architecture générale du notre système RIAM.**

## 5. Architecture d'un agent :

Dans cette section, nous présentons l'architecture de base des agents dans le système. L'agent mobile et l'agent de ressource (agent statique) sont basés sur les modules constitutifs suivants : le module de communication, le module de gestion des événements, le module de contrôle, le module de gestion des interactions, la base de connaissance d'agent et une bibliothèque des actions primitives. La figure VI-2 montre les différents composants d'un agent.

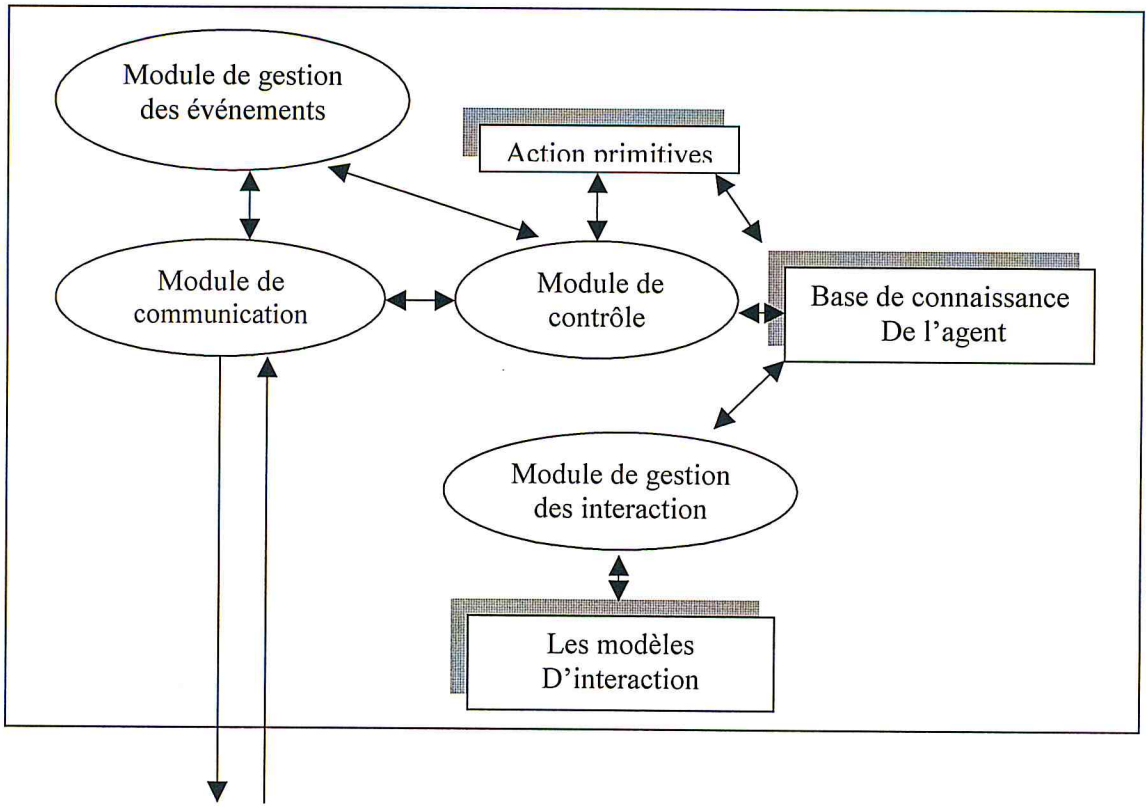



**Figure VI-2: Architecture d'un agent.**

### 5.1. Le module de communication :

C'est le responsable de la gestion de tous les messages entrants et sortants. Un événement est causé par l'arrivée d'un message. Le module de gestion d'événements identifie les événements et les transmettre au module de contrôle. Un signale l'erreur est envoyé en cas d'événement inconnu.

### 5.2. Le module de contrôle :

C'est le cœur d'un agent. Il est en fait un moteur d'inférence responsable du contrôle de l'agent. Quand un événement arrive, le module de contrôle déclenche l'action appropriée. Les actions sont surtout prédéfinies au moment de la conception de l'agent et maintenues sous forme d'une bibliothèque d'actions primitives. Dans le contexte de notre système, une action peut être un envoi d'un message ou une mise à jour d'une information.

### 5.3. Le module de gestion d'interactions :

C'est le responsable de garder la trace de toutes les interactions en cours de l'exécution d'un agent. Le module de gestion d'interactions utilise des modèles d'interactions prédéfinis quand il s'engage dans un dialogue avec un autre agent.

### 5.4. La base de connaissance de l'agent :

Elle représente le support d'information pour un agent. Cette dernière contient des connaissances sur les capacités de l'agent et sur d'autre agent avec lesquels il interagit.

### 5.5. La mobilité des agents :

Lorsqu'un agent est déplacé, un message contenant le code de l'agent ainsi que le contexte de l'agent est construit. Le contexte de l'agent est constitué d'un ensemble d'objets Java. Le mécanisme de sérialisation de Java nous permet de transformer l'ensemble des objets gérés par l'agent en un flot d'octets, de déplacer ces données vers le site destination, puis de reconstruire le contexte. Le mécanisme de chargeur de classe de Java nous permet de transformer le code de l'agent reçu dans le message en classes Java sur le site destination. Etant donné que des programmeurs différents peuvent utiliser un même nom de classe, un chargeur différent est associé à chaque agent sur un serveur d'agent. Ce chargeur est créé à l'arrivée de l'agent sur le serveur. Notons que le chargeur d'un agent ne récupère pas les classes de l'agent à la demande; le code de l'agent est déplacé avec lui en une seule fois et le chargeur est initialisé avec ce code à l'arrivée de l'agent.

Pour résumer l'implantation de notre système, le déplacement d'un agent se compose des étapes suivantes :

- sérialisation du contexte de l'agent et production d'un message incluant le contexte sérialisé de l'agent et son code.
- Envoi du contexte et du code de l'agent au serveur destination
- Réception du message sur le site destination.
- Destruction de l'agent sur le site origine.
- Création d'un nouveau processus léger pour l'exécution de l'agent.
- Création d'un chargeur pour cet agent, le chargeur étant initialisé avec le code de l'agent.
- Dé-sérialisation du contexte de l'agent.
- Démarrage de l'agent en appelant le point d'entré.

### 5.6. Communication entre les agents :

La communication entre les agents mobiles et les agents de ressources se fait via des échanges des messages. Un agent peut envoyer ou recevoir des messages qui peuvent être :

- Des messages contenant une requête.
- Des messages contenant une réponse d'une requête.
- Des messages d'information.

## 6. Architecture de Middleware:

Le Middleware est un mécanisme qui assure pour chaque agent mobile un itinéraire (Les sites où se trouvent les demandes de l'utilisateur), et pour chaque site

une requête et un ensemble d'actions. Le Middleware est composé de trois modules (figureVI-3) : Meta donnée, bibliographie de requêtes et directeur de l'agent.

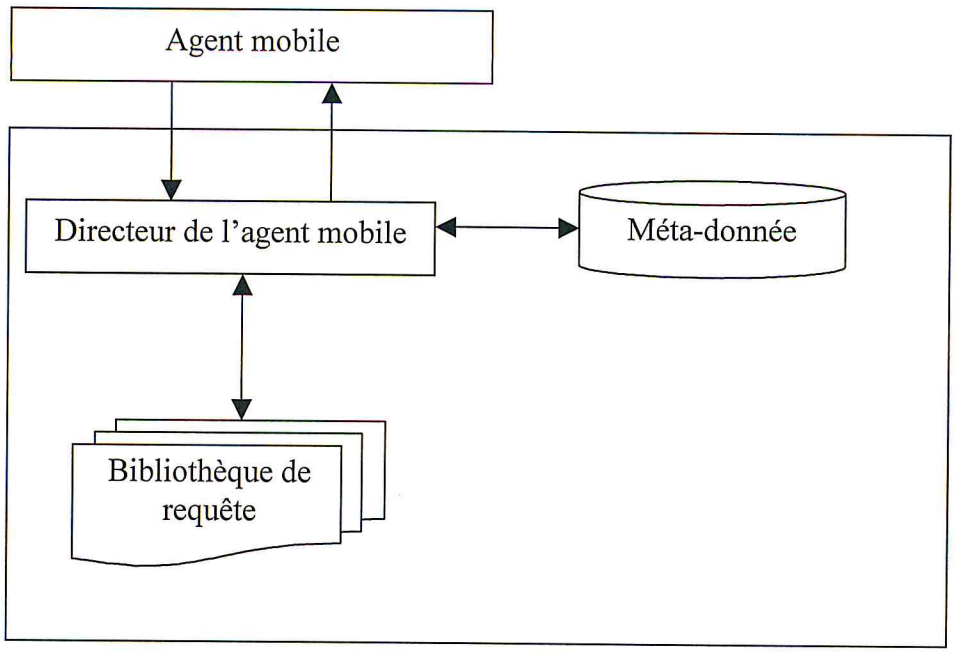



**Figure VI- 3: Structure interne de Middleware.**

**a)Bibliothèque de requête :** Contient l'ensemble des requêtes écrites en SQL standard qui peuvent utiliser pour accéder au donnée ( Par exemple : Select * from [nom de la table ] where [ x = x ]).

**b)Méta-donnée :** Ce module contient des informations sur les bases de données, leur emplacement et la taille des données.

**c)Directeur de l'agent mobile** : C'est le module principal qui échange les informations avec l'agent mobile. Il permet de :

- Détecter le but de l'agent mobile (par exemple : consultation ou réservation).
- Analyser les données entrées par l'utilisateur.
- Construire un itinéraire selon les informations qui sont dans le module de méta-donnée.
- Composer des sous requêtes pour chaque agent mobile à partir de Bibliothèque de requête et les demandes d'utilisateur.

## 7. Implémentation :

Dans ce qui suit nous allons exposer les aspects importants de l'implémentation dans le cadre de développement du logiciel, il s'agit d'implémenter le Système RIAM établie dans la partie conception.

### 7.1 Implémentation de l'agent Mobile :

Un agent mobile est une instance d'une classe java définie par le développeur de l'agent. Cette classe java héritée de la classe générale Aglet de l'API Aglets. Et elle doit définir une méthode run() qui représente le point d'entrer de l'agent et d'autre pour instantiation des variables utilisées par l'agent. La classe agent a la forme suivante (plus d'information voir Annexe):

```
Public class Agent extends Aglet{

Les variables utilisées (public, private, protected)

Public void onCreation () {
// Code à exécuté par l'agent mobile dans le site de création.
.....
....
        }

Public void run () {
// Code à exécuté par l'agent mobile dans chaque environnement
visité.
.....
.....
        }
}
```

**Figure VI- 4: Code présentant le code de base d'un agent.**

La classe agent fournit des méthodes permettant à l'agent mobile de se déplacer, Activé, désactivé, destruction de l'agent et d'autre. Par exemple le code associe à la méthode déplacement :

```
try{
                dispatch (URL destinations);

                }catch (Exception exp ){

                        }
```

**Figure VI- 5: Code présentant la primitive (dispatch).**

La communication entre les agents mobiles et les agents de ressources se fait à travers l'envois des messages, qui sont de deux types : Message Synchrone et asynchrone. Le code suivant illustre un envoi de message par un agent mobile :

```
try {
        Message msg = new Message("mobile agent");

        // pour les messages synchrones
        Object reply = proxy.sendMessage(msg);

        //  pour les messages asynchrones
        Object reply = proxy. sendAsyncMessage(msg);

        Vector Answer_msg = (Vector)reply;

        } catch (Exception e2) {
           System.out.println("Message non Envoye!");
                                }
```

**Figure VI-6:Code présentant le code de l'envoi d'un message.**

Le code suivant permet à un agent de recevoir un message à partir d'un autre agent :

```
public boolean handleMessage(Message msg) {

if (msg.sameKind ("mobile agent") ) {

 try {
        msg.sendReply( Reponses );
        return true;

       } else

         return false;

   }
```

**Figure VI-7: Code présentant un code de réception d'un message.**

### 7.2. Implémentation de l'agent de ressource :

Un agent de ressource est un agent stationnaire qui s'exécute sur l'environnement de création et ne se déplace pas. La classe de cet agent est aussi héritée (figure VI-8) de la classe générale Aglet et le développeur n'implémente pas la méthode de déplacement.

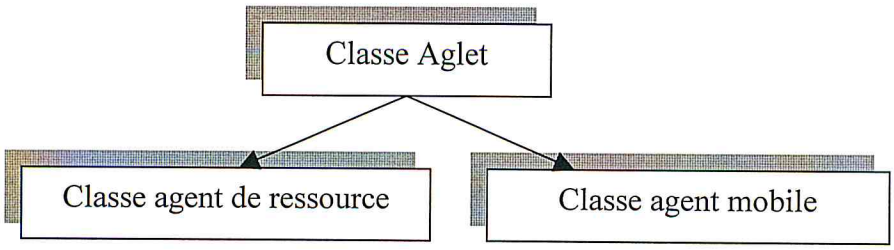



**Figure VI- 8: La classe Aglet de base et les dérivées.**

Une fois que l'agent de ressource est crée, il établit une connexion à une base de donnée selon le type de ce dernier. Pour implémenter une connexion à la base de données avec le langage JAVA il faut utiliser l'API JDBC, Le code suivant présent l'accès à une base de données MySQL avec l'API JDBC.

```
        try {
Class.forName("org.gjt.mm.mysql.Driver").newInstance();
            }
                catch (Exception E) {
         System.err.println("problème de chargement du pilot.");
                    E.printStackTrace();
            }
```

**Figure VI- 9: Code présentant une connexion vers une BD.**

### 7.3. Implémentation de Off-Line :

Dans le cas où l'utilisateur éteint sa machine avant que l'agent mobile arrive, ce dernier réside dans la dernière agence visitée jusqu'au démarrage de l'interface utilisateur une autre fois. Puis il retourne à son initiateur (utilisateur).

### 7.4. La multiThread :

Afin d'améliorer la performance de notre système. Nous utilisons le parallélisme des threades. Nous déclarons un seuil (n) pour les agents mobile en attend au niveau du Middleware et des agences ( tourisme, annuaire, hôtel et banque). Quand le nombre des agents mobiles est égal au seuil (n) nous déclenchons n thread chacun est associé à un agent pour traiter ces besoins.

## 8. Réalisation de l'application :

L'application réalisée est une interface graphique manipulant notre système multiagent, ce dernier est intégré dans l'application et fonctionne en arrière plan. Cela veut dire que lorsqu'un utilisateur demande un service parmi ceux qui sont offerts par

notre application, le système multiagent garantir ce service pour l'utilisateur mais sans qu'il sache qu'un système multiagent offre ce service pour lui.

Les services fournis par notre application sont : l'interrogation de la base de données qui permet de chercher des informations associées à un ou plusieurs hôtels et la réservation d'une chambre vide dans un hôtel. La figure VI-10 montre l'interface graphique, à travers cette interface on peut choisir un service comme il sera montrer ultérieurement.

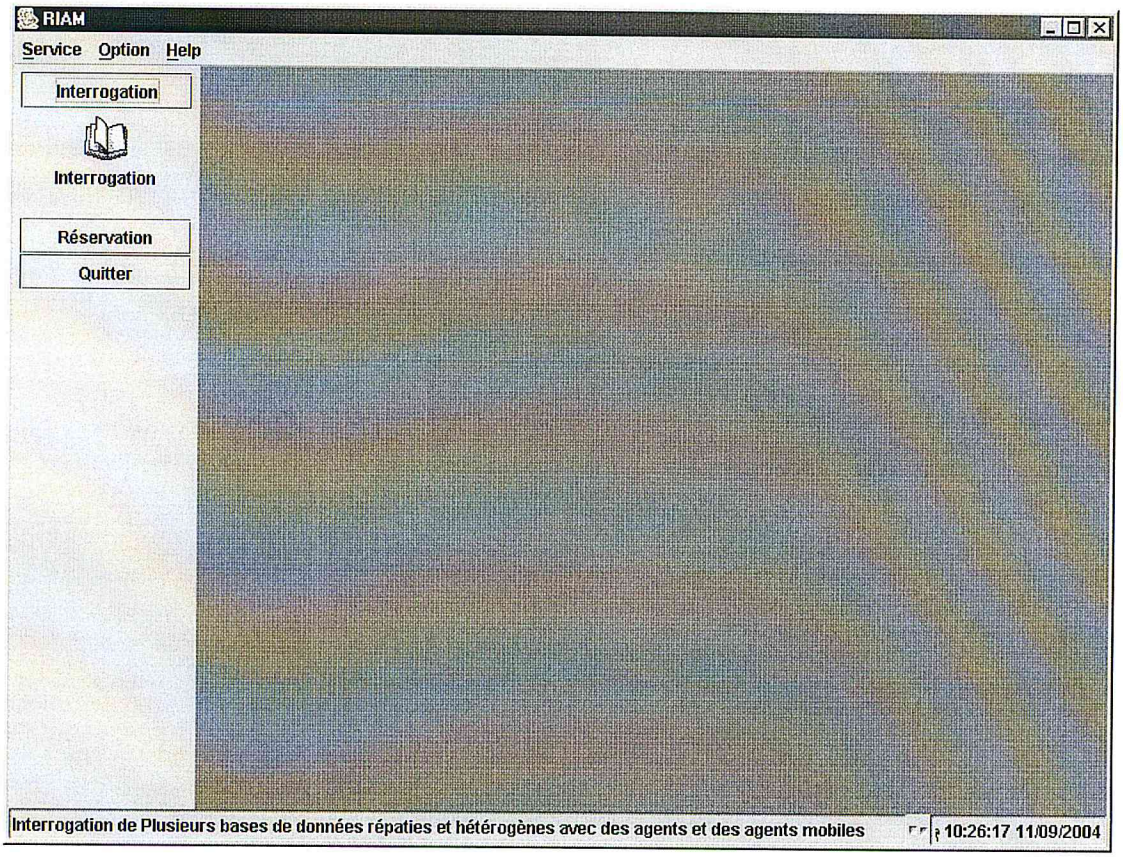



**Figure VI- 10: L'interface principale de l'application.**

## 8.1 L'interrogation :

Ce service permet aux utilisateurs de chercher des informations sur un ou plusieurs hôtels, le genre des informations cherchées par l'utilisateur varié selon les données entrées par cet utilisateur. Les données à entrer sont les suivants : nom de l'hôtel, classe de l'hôtel, prix, ville et pays. Par exemple si l'utilisateur entre une classe de l'hôtel = 5 étoiles et une ville = alger c'est à dire il veut voir la liste des hôtels de 5 étoiles qui se trouvent dans la ville nommée alger. L'utilisateur peut entrer n'importe quelle combinaison de données d'entrer décrites précédemment. L'utilisation de ce service est à travers l'interface principale en cliquant sur le menu **`Service`** puis sur **`Interrogation`**, ou bien sur le bouton **`Interrogation`** qui se trouve à gauche.

La figure VI-11 montre l'interface graphique qui permet aux utilisateurs de saisir leurs données d'entrée. L'interface d'interrogation est un formulaire dont les champs sont les données d'entrée.

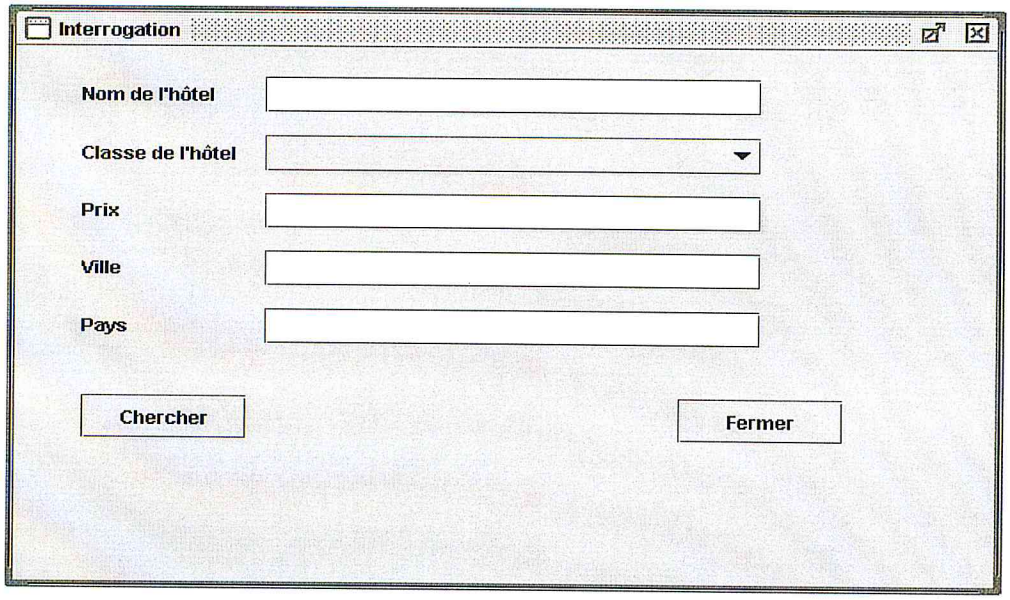



**Figure VI- 11: Interface de l'interrogation.**

## 8.2. La réservation :

La figure VI-12 montre l'interface de réservation, l'utilisation de ce service est à travers l'interface principale en cliquant sur le menu **`Service`** puis sur **`Réservation`**, ou bien sur le bouton **`Réservation`** qui se trouve à gauche.

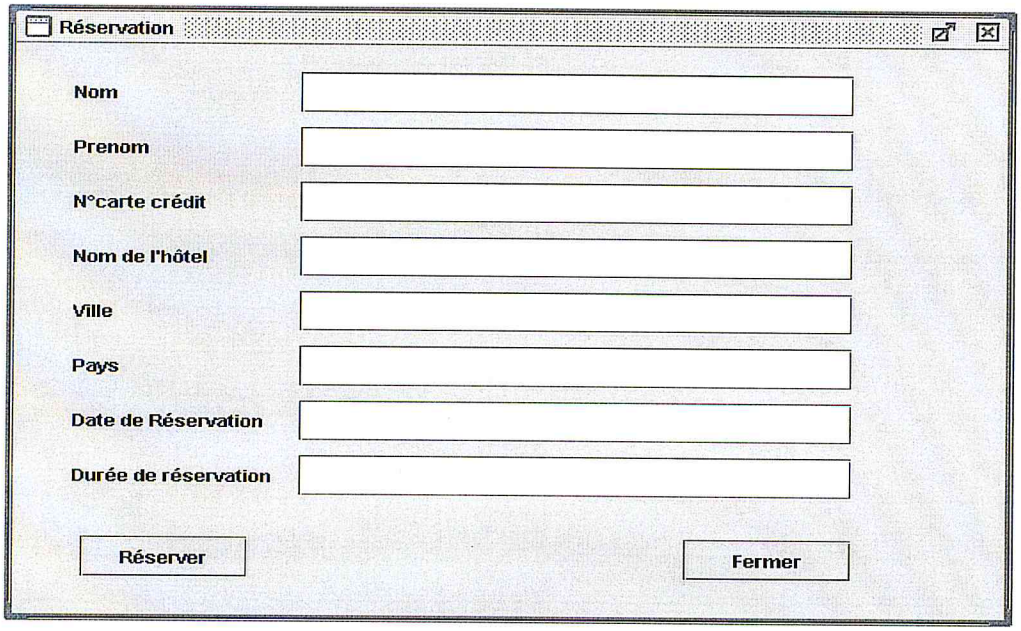



**Figure VI-12: interface de réservation**

La réservation permet aux utilisateurs de réserver une chambre vide dans un hôtel, cette opération est complétée lorsque l'utilisateur remplir ses informations privées et clique sur le bouton de **`Réserver`**, l'utilisateur doit remplir tous les champs qui appartient à l'interface de réservation. Cette dernière, comme il est montré par la figureVI-12, est un formulaire dont les champs sont les données d'entrées nécessaires pour compléter la réservation. Ces informations sont : nom (de

l'utilisateur), prenom (de l'utilisateur), n° de la carte crédit (de l'utilisateur), nom de l'hôtel, ville, pays, date de réservation, durée de réservation.

## 9. Test :

Cette partie permet de tester la qualité du système. Ce test est effectué sur une machine disposant l'application RIAM reliée avec un réseau dont les extrémités sont des machines contenant les bases de données à interroger et les agences, et un post contient le Middleware. Les machines contenant les bases de données doivent contenir aussi des serveurs autant qu'il existe des bases de données.

Le déroulement de ce test commence par le lancement de l'application RIAM, l'agence (le serveur agent) sera lancée par défaut lors de lancement de l'application, ensuite le lancement de l'interface de l'interrogation, le remplissage des données (par exemple en remplissant le champ de ville par *Alger*), les informations cherchées sont la liste des hôtels qui se trouvent dans la ville nommée Alger, après qu'on clique sur le bouton **chercher**, un agent mobile sera crée, lorsqu'il migrera vers le site de Middleware, il portera avec lui l'état de l'interface d'interrogation pour que le Middleware sache ce que l'utilisateur veut savoir et bien spécifier l'itinéraire de l'agent et les sous requêtes à exécuter. Le Middleware doit être en cours de travail lorsque l'agent mobile commence son excursion, car le premier site dont l'agent doit visiter est celui qui contient le Middleware afin que ce dernier défini à l'agent son itinéraire et ces sous requêtes. La figure VI-13 montre la manière de travail de Middlewere.

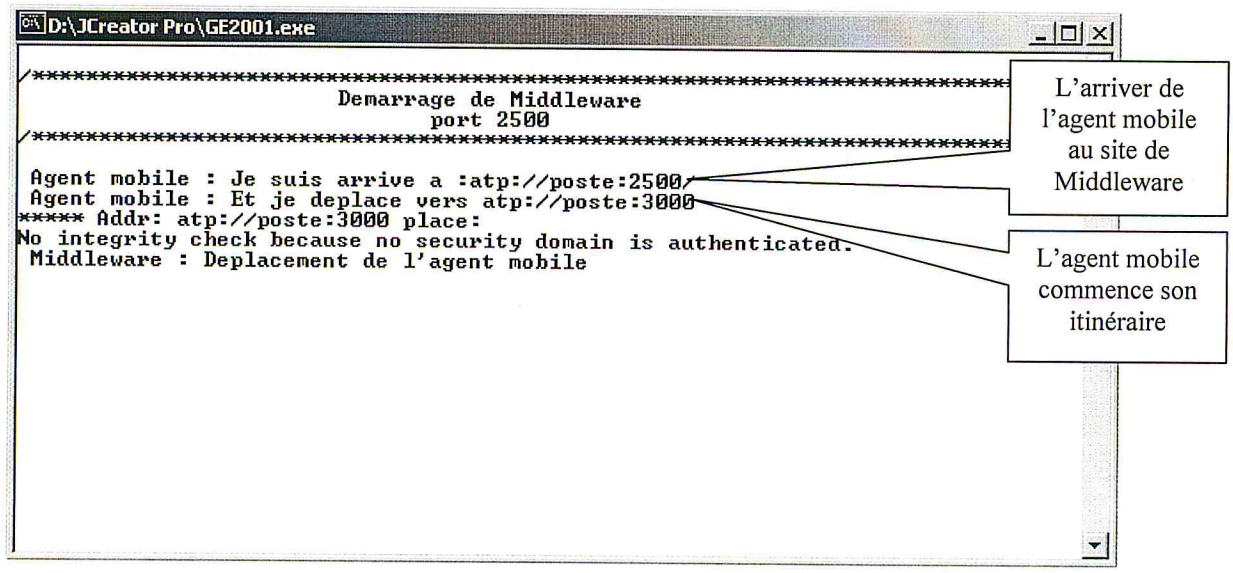



**Figure VI-13: Le Middleware.**

Après que le Middleware produit l'itinéraire et les sous requêtes de l'agent mobile, ce dernier commence son itinéraire afin de récupérer les informations cherchées par l'utilisateur. Dans ce cas (la liste des hôtels qui se trouvent dans la ville nommée Alger) le premier site à visiter est celui qui contient la base de donnée de l'agence de tourisme et le deuxième contient la base de donnée de la compagnie de téléphone. Les figures VI-14 etVI-15 montrent respectivement la visite de l'agent mobile au premier site et la visite de l'agent mobile au deuxième site.

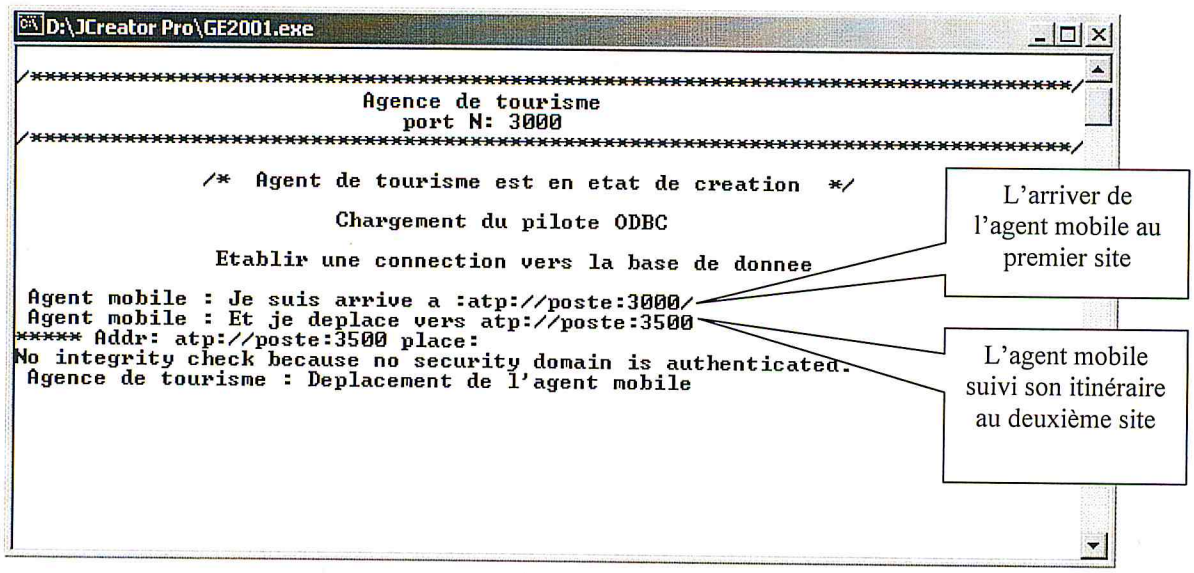



**Figure VI- 14: Visite de l'agent mobile au premier site.**

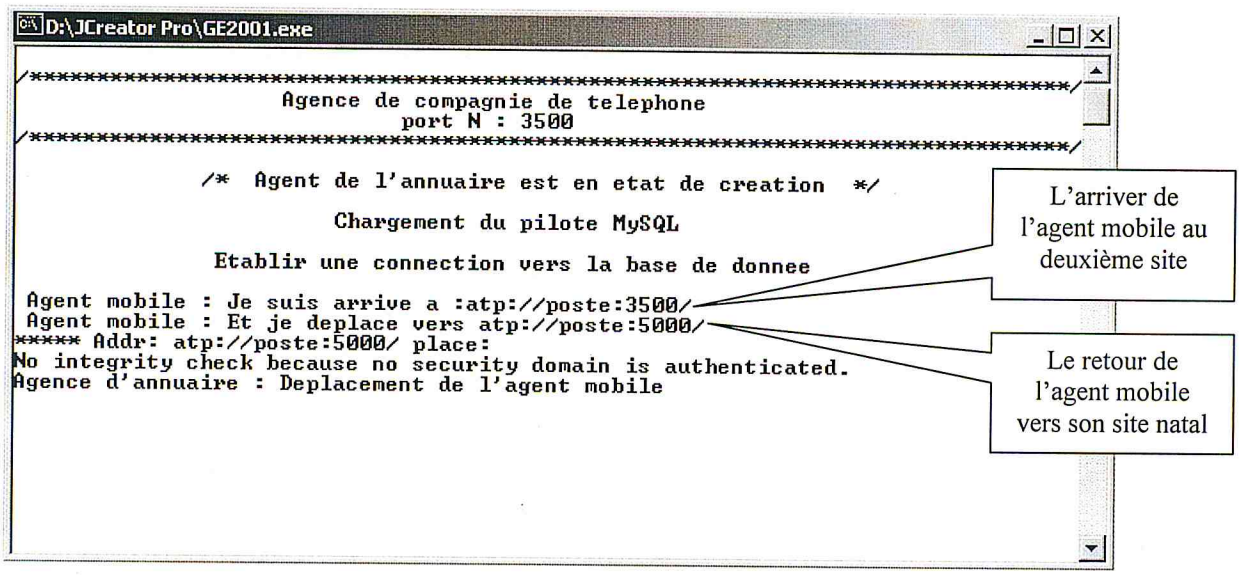



**Figure VI-15: Visite de l'agent mobile au deuxième site.**

Après le dernier site dans son itinéraire, l'agent mobile migre vers son site natal portant avec lui les informations cherchées par l'utilisateur et lorsqu'il arrive, il affiche ce résultat pour qu'il soit lisible par l'utilisateur. La figure VI-16 montre le résultat de cet exemple qui sera affiché.

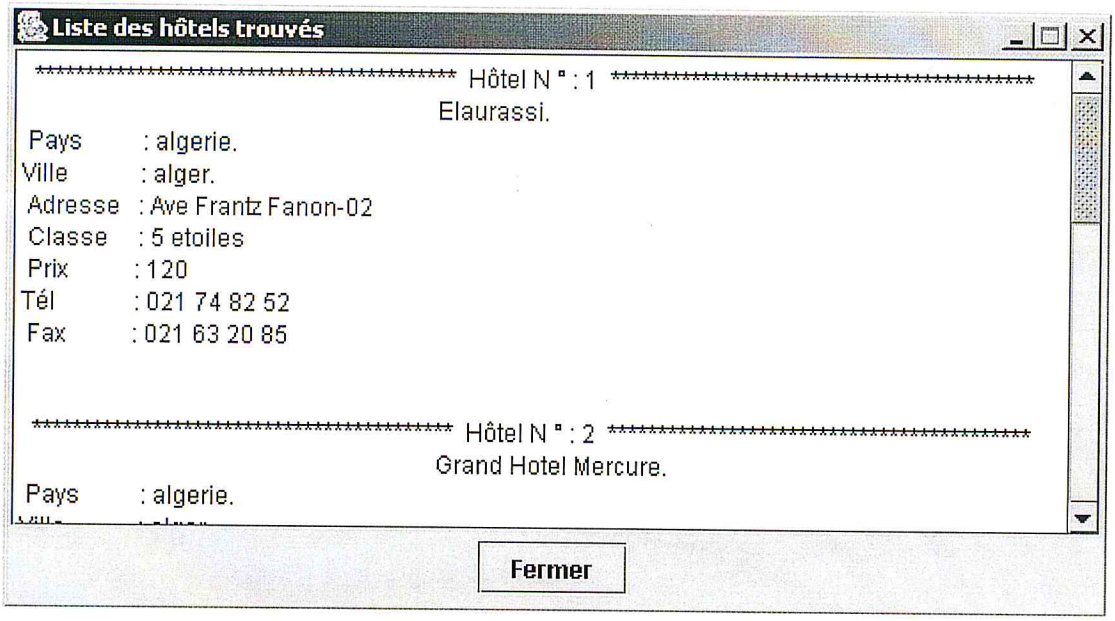



**Figure VI-16: Le résultat de l'exemple.**

Maintenant, l'utilisateur a toutes les informations nécessaires sur l'hôtel qu'il veut réserver une de ses chambres. Donc, il peut faire une réservation en cliquant sur le menu **`Service`** puis sur **`Resrvation`**, l'interface de réservation est apparaître. En cliquant sur le bouton **`Réserver`** après avoir remplir toute les informations nécessaires pour compléter la réservation, un message est apparaît dont le contenu est soit : « réservation complétée avec succès » ou « réservation échouée ». Les figures VI-17 et VI-18 montrent respectivement les deux messages qui parmi eux, un doit être affiché comme résultat de réservation.


**Figure VI- 17: Le message apparaît lorsque la réservation réussite.**


**Figure VI- 18: Le message apparaît lorsque la réservation est échouée.**

## 10. Conclusion :

Dans ce Chapitre, nous avons présenté. Une proposition d'une mise en œuvre possible de notre architecture décrit dans le chapitre précédent, à base d'un mélange d'agent statique et mobile. Le coté conversationnel a été possible grâce à l'utilisation de l'API de manipulation des agents Aglets pour formaliser les échanges entre agents.

Notre système a donné des résultats satisfaisants, mais des améliorations restent encore à faire, notamment en ce qui peut survenir pour l'agent mobile des problèmes en cours de leur déplacement.

# *Conclusion générale*

L'objectif primordial du présent travail consistait à contribuer à l'intégration du paradigme agents logiciels et Système multiagent dans le cadre de base de donnée répartie. La motivation qui sous-tend ce travail réside principalement dans le fait que les agents offrent une souplesse dans le coté de l'intégration des données et une meilleure performance dans le cadre de la recherche d'information par rapport à l'architecture client-serveur [20].

Au cours de notre projet nous avons développé un Système dans le but est d'interrogé de base de donnée répartie. En masquant tous les obstacles liés aux l'hétérogénéités des données. Et de rendre autonome l'exécution d'une requête répartie. Pour atteindre notre objectif nous nous profitons d'utilisé la caractéristique de la mobilité de l'agent mobile afin de déplacer la requête vers la base de données concernée. Et de remplacé les interactions à distances par les interactions locales.

L'approche agents mobiles constitue un paradigme de programmation puissant et convenable surtout pour les applications réparties. Particulièrement dans la recherche d'information, Elle s'avère excellente pour des bases de données réparties.

**Les objectifs :**

Nous allons maintenant revoir l'essentiel de ce que nous avons accompli comme travail à la lumière des objectifs que nous nous sommes fixés.

**Objectif 1 : Etudier les mécanismes et le fonctionnement des agents.**

Nous avons présenté tous ce qui concerne les agents, leur définition, caractéristique, les interactions entre les agents, …etc. Et le même pour les agents mobiles.

**Objectif 2 : Présenter le domaine de base de donnée répartie.**

Nous avons présenté le domaine de base de donnée répartie, les SGBD répartis et ses objectifs. Puis les approches de conception BD répartis. Nous avons présenté aussi les niveaux des hétérogénéités des données (Structure : modèle de donnée, SGBD (constructeur)…etc.).

**Objectif 3 : Proposer une architecture après l'étude des systèmes existants.**

Nous avons présenté des systèmes existant basé sur la technologie d'agent mobile. En suite nous avons proposé une architecture applicable appelée RIAM, ainsi les différents composants de cette architecture, les agents mobiles, ressources et le Middleware. Nous avons aussi conçu notre système avec UML.

**Objectif 4 : Une mise en œuvre possible de l'architecture RIAM.**

Nous avons donné une proposition d'une mise en œuvre possible d'une application multiagents basé sur l'architecture RIAM. Cette application est implémentée par le langage de programmation Java augmenté par des APIs Java non supporté par API Java Standard. Ces APIs sont API Aglets pour les agents et les APIs JDBC pour les Bases de données.

**Perspectives :**

En perspective, nous proposons quelques extension du système, qui se résument dans ce qui suit :

- Définition d'une politique qui permet la persistante de l'agent mobile dans notre système RIAM. Pour les problèmes liés au réseau ou l'indisponibilité des services ou niveau des agences.

- Définition d'un mécanisme d'authentification des agents pour gérer les droits qu'ils auront sur les environnements d'exécutions.

- Introduire au niveau de Middleware un mécanisme qui permet de construire d'un itinéraire afin d'optimiser le volume de données transférées entre les sites, en se basent sur les statistiques stockées dans les catalogues système relatifs aux données de différents base de données.

# Bibliographie

*Nos meilleures idées viennent des autres.*
*Ralph Waldo Emerson*

[1] : J.Ferber « Les systèmes multiagents. Vers une intelligence collective ».interEdition, 1995.

[2] : Nadine Richard « Description de comportements d'agents autonomes évoluant dans des mondes virtuels » Thèse de doctorat, Ecole Nationale Supérieure des Télécommunications, 2001.

[3] :Kane Omar. EUROMASTER « Nouvelles Technologies de l'information et de la communication et multimédia » MédiaRoanne et CITCOM France Télecom-Janvier 2001.

[4] :Y.Labrou et T.Fenin « A Proposal for a new KQML Specification », rapport technique CS-97-03, Université de Maryland Baltimore County, février 1997.

[5]: Deneubourg J. L., Gross S., Beckers R. et Sandini G., « Collectively self-solving problems, in Self organization, emergent properties and learning» A. Babloyantz editors, Plenum,1991.

[6]: Imed Jarras et Brahim Chaib-draa « Aperçu sur les systèmes multiagents » 2002

[7] : Marc-André Labrie « Langage de communication agent basé sur les engagements par l'entremise des jeux de dialogue » Janvier 2004

[8]: Wooldridge, M. et Jennings, N. R. (1995). « Intelligent agents: Theory and practice » Knowledge Engineering Review, 10(2).

[9]: Russell, S. J. et Norvig, P. (1995). « Artificial Intelligence : A Modern Approach » Prentice Hall.

[10]: D.Gaïti/G.Pujolle, « L'intelligence dans les Réseaux ». EYROLLES, octobre19 92.

[11]: Filippo Peverelli ,Laurent Cottier , Raphaël Mellid ,Tibor Moinat, « L'intelligence artificielle » juin 2002

[12] : Adina Magda Florea Professeur à l'Université "Politehnica" de Bucarest

« Agents et systèmes Multi-agents » http:// turing. cs. pub. ro/ auf2/

[13]: Alper caglayan, Colin Harrison. « Les agents: Applications bureautique, Internet et intranet ». InterEditions 1998.

[14]: Philippe SELLEM. « Accès à des données réparties à travers le Web :Méthodes de recherche (agents intelligents) » 2000.

[15]: Romaric CHARTON . « Des agents intelligents dans un environnement de communication multimédia : vers la conception de services adaptatifs », 2003.

[16] : Alper Caglayan – Colin Harrison « Les agents Application bureautique Internet et intranet » InterEdition 1998.

[17] : Sébastien LERICHE —Jean-Paul ARCANGELI « Une architecture pour les agents mobiles adaptables » IRIT – UPS 118 route de Narbonne, 31062 Toulouse Cedex 4, {leriche, arcangel}@irit.fr

[18]: Nicolas Esposito « Les agents mobiles – une introduction »
DASSULT SYSTÈMES – Recherche et nouvelle technologies

[19]: BETTAHAR Aoued « Les Aglets d'IBM »
Université de montréal cours IFT6802 – H2003

[20]: Daniel Hagimont – Inira Rhône – Alpes
« Système à agents mobiles » Ecole « construction d'applications réparties » DH-1999
*http://sirac.inrialpes.fr/~hagimont*

[21] : THESE *Présentée par* Stéphane PERRET,*Pour obtenir le titre de Docteur de l'Université Joseph Fourier - Grenoble I* « Agents mobiles pour l'accès nomade à l'information répartie dans les réseaux de grande envergure » Date de soutenance: 19 novembre 1997

[22]: Daniel Hagimont – Leila Ismail « Agents Mobiles et client/serveur : évaluation de performance et comparaison » INRIA Rhône – Alpes Institut National Polytechnique de Grenoble.2000

[23]: Francis Van Aeken « Les systèmes multiagents Minimaux » Institut National Polytechnique de Grenoble, 1999.

[24]: James E.White, « Telescript Technology:An Introduction to the Language » General Magic White Paper GM-M-TSWP3-0495-V1, General Magic, Inc., 420 North Mary Avenue, Sunnyvale, CA 94086, 1995.

[25]: William R. Cockayne, Michael Zyda, "*Mobile Agents*", Manning Assoc, 1997.

[26]: Daniel Genovese, « *Les agents mobiles pour le World Wide Web : étude comparative*, rapport du probatoire au Conservatoire National des Arts et Métiers », juin 1997.

[27]: Joseph Kiniry, Daniel Zimmerman, « A Hands-On Look at Java Mobile Agents », *IEEE Internet Computing,* juillet - août 1997.

[28]: Keith Kotay, David Kotz, « Transportable Agents », *Proceedings of the CIKM Workshop on Intelligent Information Agents, 3 rd International Conference on Information and Knowledge Management, CIKM 9*4, Gaithersbury, Maryland, décembre 1994.

[29]: David Kotz, Robert Gray, Saurab Nog, Daniela Rus, Sumit Chawla, George Cybenko, « AGENT TCL: Targeting the Needs of Mobile Computers », *IEEE Internet Computing,* volume 1, numéro 4, juillet - août 1997, pp. 58-67.

[30]: Robert S. Gray, « Agent Tcl: A transportable agent system », James Mayfield and Tim Finin, eds., *Proceedings of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management (CIKM 95*), Baltimore, Maryland, décembre 1995.

[31]: Danny B. Lange, Mitsuru Oshima, IBM Research, « *Programming and Deploying Java Mobile Agents with Aglets* », Addison-Wesley, 1998.

[32]: Danny B. Lange, « Present and Future Trends of Mobile Agent Technology », *Proceedings of the Second International Workshop on Mobile Agents 98 (MA'98*), Stuttgart, Allemagne, 9-11 septembre, 1998.

[33]: David Wong, Noemi Paciorek, Tom Walsh, Joe DiCelie, Mike Young, Bill Peet, « Concordia: An Infrastructure for Collaborating Mobile Agents », *Proceedings of the First International Workshop on Mobile Agents, MA'97*, Berlin, Allemagne, 7-8 avril, 1997.

[34]: Tom Walsh, Noemi Paciorek, David Wong, « Security and Reliability in Concordia », *Proceedings of the 31 st Annual Hawaii International Conference on System Sciences (HICSS31*), Kona, Hawaii, January 6-9, 1998.

[35]: James E. White, « Telescript Technology: Foundation for the Electronic Marketplace », white paper, General Magic, Sunnyvale, California, 1996. Une autre version de l'article se trouve dans Bradshaw, Jeffrey (ed.) *Software Agent*s, Menlo Park, California, AAAI Press / The MIT Press, 1996.

[36]: James E. White, C. S. Helgeson, and D. A. Steedman, « System and method for distributed computation based upon the movement, execution, and interaction of processes in a network », US Patent 5,603,031, filed 8 filed 1993, issued 11 February 1997.

[37]: Dr. I-Ling Yen « A Data Protocol for Mobile Agent Based Internet Data Access System »,Department of Computer Science University of Texas at Dallas August 2002.

[38]: Daniel Hagimont – Leila Ismail - Jacques Mossière « Spécialisation de serveurs par des agents mobiles » Projet SIRAC INRIA Rhône – Alpes Institut National Polytechnique de Grenoble.2000

[39]: www.uml.free.fr

[40]: G .Gardarin, P.Valduriez « SGBD Avancés Base de données Objets, déductives, réparties » Errolles, 1989.

[41] : Georges Gardarin, Olivier Gadarin, « le client-serveur »,1999.

[42]: Rim Moussa , « Systèmes de Gestion de Bases de Données Réparties & Mécanismes de Répartition avec Oracle ». 2003-2004.

[43]: Eddy.Meylan, « Bases de données réparties »,2003.

[44]: Pierre-Alain Muller, modélisation objet avec UML, EYROLLES, 2001.

[45]: J.bres, ateliers de genie logiciel, masson, 1993.

[46]: Le développeur Java™2 «Antoine Mirecourt », OEM (Osman Eyrolles Multimedia), Edition 1999

[47] : « Programmer en Java CPR informatique », Septembre 2002.

[48] :Bettahar Aoued, Les Aglets d'IBM , université de Montréal ,Cours ITF 6802 – H2003.

[49] : Danny B. Lange et Mitsuru Oshima.“ Mobile Agents with Java: The Aglet API”.

[50] : AKLOUF Y. & DRIAS H., 'VEMMA : Une Architecture de place de marché à base d'agents mobiles et intelligents', in proc of ISPS'03, Alger, Mai 2003.

[51] : Morvan,hameur, 'Apport des agent à l'optimisation de requêtes réparties à grande échelle'.Institut de recherche en informatique de toulouse.

[52]: Bob Tarr.Danko Nebesh.Sterling Foster.''Introduction To Mobile Agent Systems and Applications''. Tools USA 2000.

[53]: Daniela Barreiro Claro, João Bosco Mangueira Sobral. ''TOWARDS THE INFORMATION RETRIEVAL WITH MOBILE AGENT''. Department of Computer Science and Statistics Computer Science Pos-Graduate Course Federal University of Santa CatarinaFlorianópolis - Santa Catarina – Brazil.

[54]: Griselda Chevalier , J. Alfredo , Lourdes Fernández ,Sandra Edith.''Viajerus: A Framework Based on Mobile Agents for Distributed Information Retrieval''.

[55]: Yan Wang, Jian Ren,“Building Internet Marketplaces on the Basis of Mobile Agents for Parallel Processing”, Proc. of 3rd International Conference on Mobile Data Management (MDM2002), IEEE Computer Society Press,Jan. 8-11 2002, Singapore, pp 61-68.

# Annexe A. Aglets

## 1. Introduction :

L'API Aglets est une plate-forme basée sur Java proposée pour développer des agents mobiles dans un environnement hétérogène comme Internet. Elle a été développer dans le laboratoire d'IBM à Tokyo par un équipe de chercheur au début de 1995. [48]

## 2. Définition :

M. BETTAHAR Aoued de l'université de Montréal défini les Aglets d'IBM comme suit : « Les aglets (**Ag**ents App**lets)** sont des objets Java mobiles qui peuvent se déplacer d'une machine à une autre. Ainsi, un aglet qui s'exécute sur un hôte peut stopper son exécution, se déporter vers un hôte distant et continuer cette exécution dans son nouvel environnement ». [48]

Une autre définition écrite par *Danny B. Lange* et *Mitsuru Oshima* qui sont des membres de l'équipe de chercheurs qui à développer l'API Aglets : « L'établi Aglets d'IBM reflète le modèle d'applet dans Java. Le but est d'apporter la saveur de la mobilité à l'applet. Le terme aglet est en effet la composition de deux mots ***agent*** et ***applet.*** On a essayer de fabriquer un usage de l'Aglets dans le "*clean design*", et nous espérons que les nombreuse manières dans lequel le modèle Aglet reflète le modèle applet appréciera les programmeurs des applets ».

## 3. Les attributs d'un Aglets :

Un Aglet a 5 attributs [48]:

**Etat :** a pur but de reprendre l'exécution après la migration.

- Exécution state : Instruction en cour + pile (pas toujours nécessaire car les valeurs des variables peuvent aider à déterminer l'état d'exécution après migration) .

- Object state : Valeur des variables d'instance.

**Code :** le code doit être exécutable et sécurisé (JAVA est meilleur choix ).

**Interface :** peut être :

- Un ensemble de signatures de méthodes.
- Un échange de messages : par exemple pont à point ou broadcast mode synchrone ou asynchrone.

**Identificateur :** par exemple un nom et un numéro de série.

**Principale :** auteur de l'aglet, processeur de l 'Aglet.

Notons ici que les attributs d'un Aglet migrent

**4. Les éléments d'un Aglet :**

Les éléments d'un Aglet sont les suivants [49]:

**Aglets :** Un Aglet est objet mobile de Java qui visite les serveurs où les agents sont autorisés, dans un réseau informatique. Un aglet est autonome puisqu'il peut reprendre son exécution dès son arrivée à destination et réactif car il peut répondre (réagir) à des événements de son environnement.

**Proxy :** Un proxy est un représentant d'un aglet. Il sert de bouclier à l'aglet contre l'accès direct à ses méthodes publiques. Le proxy fournit également la transparence à l'emplacement pour l'aglet. C'est-à-dire qu'il peut cacher le vrai emplacement de l'aglet.

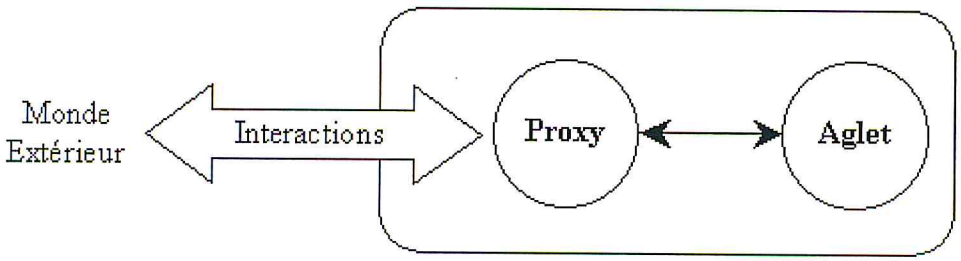



**Figure 3 : relation entre un aglet et son proxy[3]**

**Contexte :** Le contexte est l'environnement d'exécution de l'aglet via un proxy. Il fournit des moyens pour mettre à jour et contrôler des aglets dans un environnement d'exécution uniforme et sécurisé contre les aglets malveillants. Un nœud dans un réseau informatique peut exécuter plusieurs serveurs Aglets et chacun de ces serveurs peut accueillir plusieurs contextes. Un nœud capable d'héberger plusieurs contextes est appelé *Hôte*. Les contextes sont nommés et peuvent être localisés par la combinaison de l'adresse de leur serveur et leur nom.

**Message :** Un message est objet échangé entre Aglets. Il permet de transmettre des messages synchrones ainsi qu'asynchrones entre Aglets. Ces messages transmettent peut utilisés par des Aglets pour collaborer et échanger des informations.

**Future réponse :** La future réponse est utilisée dans l'émission asynchrone d'un message c'est un *handler* pour recevez la réponse après synchroniquement.

**Identificateur :** L'identificateur est lié à chaque Aglet. Cet identificateur est globalement unique et immuable durant toute la vie d'un Aglet.

## 5. Cycle de vie d'un Aglet :

Les principales opérations affectant la vie d'un aglet sont :

**Création :** se fait dans un contexte, une seule identifiant est attribuée. La création peut se faire de deux manières possibles :

1. Instantiation : attribution d'un contexte puis initialisation.
   Début d'exécution dès que l'initialisation est achevée
   Création par un autre Aglet ou un serveur d'Aglet
2. Clonage : faire une copie d'un Aglet existant (même contexte et identifiant défirent).

**Déportation** (Migration)**:** transfert d'un Aglet d'un contexte vers un autre.
**Récupération :** l'Aglet déporter est récupéré (tiré) dans son contexte d'origine.
**Désactivation :** interruption temporaire de son exécution et stockage de son état d'exécution.
**Activation :** reprendre l'exécution d'un Aglet désactivé.
**Libération ou destruction :** fin de vie d'un Aglet et son retrait du contexte.

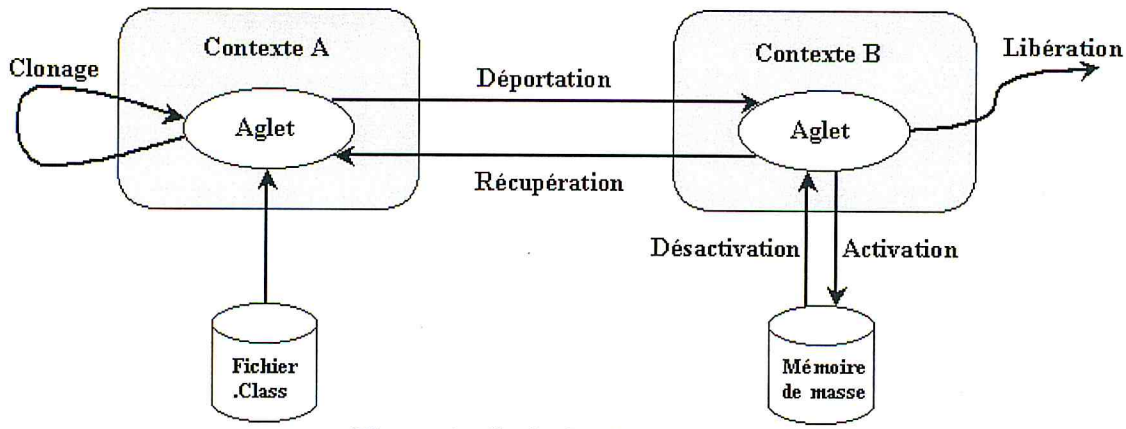



**Figure 4 : Cycle de vie d'un Aglet**

## 6. Mobilité d'un Aglet :

Afin de pouvoir transférer un Aglet (agent mobile), Aglets utilise un mécanisme de sérialisation de Java. Le système Aglets permet de transférer le code et les données des objets de l'agent mais ne permet pas le transfert de l'état d'exécution.

Le système Aglets don la possibilité au Aglet de violer l'autonomie d'un autre Aglet (déclencher la migration d'un autre Aglet). Les systèmes utilisant Java ne proposent aucune solution pour la migration des *threads.* [21]

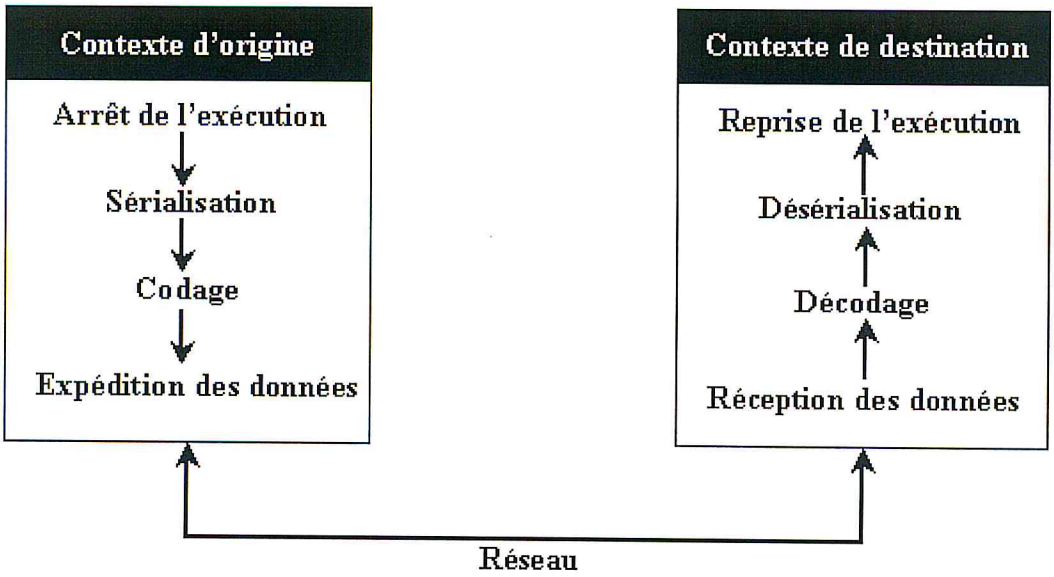



**Figure 5 : Transfert d'un Aglet.**

## 7. La Création de l'Aglet :

La création d'un aglet se fait par l'appel de la méthode **createAglet()** du contexte et retourne le proxy associé à l'aglet nouvellement crée. Le code associe à cette méthode :

```
Public abstract AgletProxy createAglet ( URL codeBase, String code, Object init )
```

Le paramètre **init** de type **Object** est transmis à l'aglet nouvellement crée. Ce dernier le récupère dans sa méthode **onCreation(Object init)**.Le code de base d'un aglet :

```
Public class Agent extends Aglet{

Les variable utilisées (public, private, protected)

Public void onCreation () {
// Code à exécuté par l'agent mobile dans le site de création.
.....
....
        }

Public void run () {
// Code à exécuté par l'agent mobile dans chaque environnement
visité.
.....
.....
        }
}
```

Code présente un code de base d'un aglet

## 8. Le modèle événementiel de l'aglet :

La programmation des aglets est basée sur la gestion des événements. Des notifications alertent l'aglet sur les actions influant sur son cycle de vie. Le programmeur à la possibilité d'implémenter des écouteurs (*listeners*) pour faire réagir l'aglet.

Il existe trois catégories de notifications:

### Notifications de clonage :

Permettent la capture des événements liés au clonage de l'aglet. Cette fonctionnalité est obtenue par implémentation de l'écouteur (interface) **CloneListener**.

### Notifications de mobilité :

Permettent la capture des événements liés au mobilité de l'aglet. Cette fonctionnalité est obtenue par implémentation de l'écouteur (interface) **MobilityListener.**

### Notifications de Persistance :

Permettent la capture des événements liés à la persistance de l'aglet. Cette fonctionnalité est obtenue par implémentation de l'écouteur (interface) **PersistencyListener.**

### L'interface Context :

Un aglet peut accéder aux informations relatives à son contexte d'exécution et communiquer avec son environnement grâce à l'interface **AgletContext**. Le contexte est crée par un serveur d'aglets qui n'est rien d'autre qu'une application permettant de gérer des aglets.

## 9. Le Modèle de Communication des Aglets :

La communication est basée sur l'échange d'objets de la classe **Message**. Ce modèle de communication est indépendant de la localisation de l'aglet et peut être asynchrone ou synchrone.

Un aglet possède un gestionnaire de messagerie **MessageManager** qui lui permet de les traiter un par un et dans l'ordre de leur arrivées respectives. Toutefois, cet ordre peut être changé par l'aglet en modifiant les priorités des messages dans la file d'attente.

### Les messages :

Chaque message est caractérisé par la catégorie (**kind**) à laquelle il appartient. La création d'une instance message nécessite l'affectation d'une valeur à son paramètre **kind**. Le code suivant illustre une création d'une message :

```
Message msg = new Message ("création".Objet);
```

**Les type des messages:**

***Messages synchrones :***

La méthode **sendMessage(Message msg)** permet l'envoi de messages synchrones. Elle permet d'envoi un message et attente la réponse.

Le destinataire du message doit implémenter un gestionnaire des messages reçus. Cette fonctionnalité est rendue possible par la surcharge de la méthode **Aglet.handleMessage(Message msg)**.Et la méthode **Message.sendReply** permet de retourner une réponse a l'envoyeur.

***Messages asynchrones :***

La messagerie asynchrone est implémentée grâce à la notion de ***futur objet.*** L'envoi d'un message asynchrone retourne un lien vers la réponse même si cette dernière n'existe pas encore. Ce lien permet de tester si la réponse est arrivée ou pas. Cette technique permet à l'aglet d'envoyer un message sans être obligé d'interrompre son exécution en attente de la réponse. Le code suivant illustre l'envoi d'un message asynchrone :

```
Public FututreReply AgletProxy.sendFututreMessage(Message msg);
```

## 10. Installation de l'Aglet et configuration (Windows 95/98/NT) :

L'installation de ASDK Aglet Software Development Kit version 2.0.2 nécessité au minimum une Java 2.il est testé et bien fonctionnée avec les versions suivantes :

- ✓ Java Software Development Kit 1.3.1.
- ✓ Java Software Development Kit 1.4.0.

**Configuration de variable d'environnement :**

Pour exécuter le ASDK, il faut que les variables suivante sont placé avec les variable d'environnement :

```
set java_home= X: \j2sdk1.4.0\bin
set aglet_home= X:\aglets-2.0.2
set path=%path%; X:\j2sdk1.4.0 \bin
set path=%path^%;%aglet_home%\bin;%path%
set aglet_path= X:\aglets-2.0.2\public
set aglet_export_path=%aglet_path%
set classpath=%classpath%; X:\aglets-2.0.2\public; X:\aglets-2.0.2\lib\.
```