

Marc Côté

NetSA, une architecture multiagent et son application aux services financiers.

Mémoire
présenté
à la Faculté des études supérieures
de l'Université Laval
pour l'obtention
du grade de maître ès sciences (M.Sc.)

Département d'informatique
FACULTÉ DES SCIENCES ET DE GÉNIE
UNIVERSITÉ LAVAL

Avril 1999



National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-41874-X

Canada

Résumé

Les agents logiciels et les systèmes multiagents sont de plus en plus présents en informatique et plus particulièrement dans les domaines de l'intelligence artificielle, du génie logiciel, de l'Internet et des jeux. Ce mémoire explique différents aspects des agents logiciels en s'attardant surtout sur les architectures des systèmes multiagents. Il montre également la conception d'un système multiagent baptisé NetSA (Networked Software Agent) et son application au domaine financier et plus particulièrement la vente d'hypothèques par le biais d'enchères.

NetSA est une architecture multiagent spécialisée dans la recherche d'informations dans des environnements riches en informations tels que l'internet. Cette architecture est composée de trois couches : la couche de communication avec l'utilisateur, la couche de traitement de l'information et la couche d'interrogation et d'extraction d'informations.

Nous avons contribué à cette architecture en spécifiant, développant et validant l'agent superviseur, c'est-à-dire l'agent central qui est en charge de la coordination et de la planification dans NetSA. Cet agent reçoit les requêtes des utilisateurs (via l'agent utilisateur) et engage une coopération entre tous les agents utiles à la réalisation de la requête. Lorsque l'information nécessaire à l'accomplissement de la requête est déficiente, il fait appel aux agents ressources pour obtenir les informations manquantes.

Nous avons ensuite développé des algorithmes sur la base des enchères, et ce en vue de valider l'architecture NetSA. Ces enchères ont pour but d'optimiser le profit de l'acheteur ou du vendeur selon les conditions de la vente. Dans le cadre de l'application qu'on a visé, (à voir les prêts hypothécaires) seules ont été implantées des versions de l'enchère anglaise et du meilleur prix car ce sont celles qui fonctionnent le mieux avec ce genre d'application.

Les résultats obtenus montrent clairement que l'architecture NetSA s'applique très bien aux domaines riches en informations dans la mesure où elle aide les utilisateurs à faire des gains de temps et d'argent.

Brahim Chaib-draa
Directeur de recherche

Marc Côté
Étudiant

Table des matières

Table des matières	i
Liste des tableaux	v
Table des figures	vi
Résumé	viii
Avant-propos	1
1 Introduction	2
1.1 Problématique	2
1.2 Objectifs	3
1.3 Méthodologie	4
1.4 Contenu du mémoire	4
2 Agents et systèmes multiagent	5
2.1 Introduction	5
2.2 Agents intelligents	5
2.2.1 Architecture d'agents	6
2.2.2 Agents et objets	7
2.2.3 Agents et systèmes experts	8
2.3 Systèmes multiagents	10
2.4 Architectures d'agents cognitifs	11
2.4.1 Les types d'architectures d'agents	11
2.4.2 Exemple d'architecture hybride : InteRRaP	13
2.4.3 Conclusion sur les architectures du type InteRRaP	21
2.5 Architectures multiagents dédiées aux environnements riches en information.	22
2.5.1 RETSINA	22
2.5.2 Carnot	24
2.5.3 InfoSleuth	26
2.5.4 UMDL	30
2.5.5 Conclusion	33

3 Communication	34
3.1 JATLite	34
3.1.1 Utilité de JATLite	34
3.1.2 Organisation de JATLite	36
3.1.3 Construction d'un agent avec JATLite	37
3.2 KQML	40
3.2.1 Les paramètres réservés	40
3.2.2 Les 35 performatives du discours	42
3.2.3 Utilisation de KQML dans la communication pour la recherche d'informations	46
3.3 Conclusion	48
4 Architecture NetSA	49
4.1 Caractéristiques de NetSA	49
4.1.1 Réutilisabilité et portabilité	49
4.1.2 Communication entre agents basée sur JATLite	50
4.1.3 HTML, formulaire et JavaScript	51
4.1.4 Les agents composants NetSA	51
4.1.5 Propriétés des agents composant NetSA	53
4.2 Agent Utilisateur dans NetSA	55
4.2.1 Description	55
4.2.2 Architecture interne	55
4.3 Agent Ressource dans NetSA	56
4.3.1 Description	57
4.3.2 Architecture interne	57
4.3.3 Communication	58
4.4 Agent Intermédiaire dans NetSA	59
4.4.1 Description	59
4.4.2 Architecture Interne	61
4.5 Agent Superviseur dans NetSA	62
4.5.1 Description	63
4.5.2 Architecture interne	63
4.6 Conclusion	66
5 Enchères	67
5.1 Enchères anglaises	67
5.1.1 Avantages et inconvénients	68
5.1.2 Algorithme	68
5.2 Enchères hollandaises	70
5.2.1 Avantages et inconvénients	70
5.2.2 Algorithme	72
5.3 Enchères du meilleur prix	72
5.3.1 Avantages et inconvénients	74
5.3.2 Algorithme	74
5.4 Enchères de Vickrey	74

5.4.1	Avantages et inconvénients	76
5.4.2	Algorithme	76
5.5	Enchères doubles	76
5.5.1	Avantages et inconvénients	77
5.5.2	Algorithme	77
5.6	Conclusion	78
6	Hypothèque	79
6.1	Période d'amortissement	79
6.2	Terme	79
6.3	Types de prêts hypothécaires	80
6.4	Taux d'intérêt	80
6.5	Fréquence de paiement	80
6.6	Privilège de pré-paiement sans pénalités	81
6.7	Assurances vie/invalidité hypothécaire	82
6.8	Ratio d'endettement	82
6.9	Calcul d'hypothèque	83
7	Application de NetSA au domaine financier	84
8	Conclusion	94
8.1	Résultats de NetSA	95
8.2	Développements à venir	95
8.2.1	Serveur d'ontologie	97
8.2.2	Agent d'extraction de connaissances	97
Bibliographie		98
A	Programmation Orientée Plan pour les Agents (POPA)	102
A.1	Variables	103
A.1.1	Type de données	103
A.1.2	Tableau	103
A.2	Fonctions prédéfinies	104
A.2.1	<code>NbReply% : send{Dest\$, Perf\$, Context\$, InReplyTo\$, In\$, Out\$}</code>	104
A.2.2	<code>TableSize% : size{VarName\$}</code>	104
A.2.3	<code>NewText\$ / NewNumber% : format\$/ format%{Text\$ / Nombre%, Precision%}</code>	105
A.2.4	<code>print{Text\$}</code>	105
A.3	Mots réservés	105
A.3.1	<code>While{Condition?} - EndWhile</code>	105
A.3.2	<code>If{Condition?} - Else - EndIf</code>	106
A.3.3	<code>StopPlan</code>	106
A.3.4	<code>BeginConcurrentPlans - EndConcurrentPlans</code>	106
A.4	Concurrence	107

A.5 Exemple	107
B Captures d'écran	109

Liste des tableaux

2.1	Les agents réactifs.	12
2.2	Les agents inductifs.	12
2.3	Les agents coopératifs.	13
2.4	Les agents hybrides.	13
3.1	Liste des paramètres réservés des performatives en KQML.	41
3.2	Les 7 performatives de régulation de conversation (E et R désignent respectivement l'émetteur et le récepteur).	43
3.3	Les 17 performatives de discours (E et R désignent respectivement l'émetteur et le récepteur).	44
3.4	Les 11 performatives d'assistance et de réseau (E et R désignent respectivement l'émetteur et le récepteur).	45
4.1	Classification des agents intermédiaires.	61
6.1	Tableau comparatif des frais d'intérêts selon les différents paiements .	81

Table des figures

2.1	Architecture d'agent.	7
2.2	Les stades du génie cognitif selon Waterman.	9
2.3	Modèle conceptuel d'InteRRaP.	14
2.4	Architecture d'InteRRaP.	16
2.5	Couche de contrôle d'InteRRaP.	18
2.6	Couche de planification local d'InteRRaP.	20
2.7	Couche de planification coopérative.	21
2.8	Architecture de Retsina.	24
2.9	Architecture de Carnot.	25
2.10	Architecture Haut-Niveau d'InfoSleuth.	27
2.11	Nuage d'agents d'InfoSleuth(d'après [BBB ⁺ 96]).	28
2.12	Architecture d'InfoSleuth.	31
2.13	Architecture UMDL [VD95].	32
3.1	Agent router de message de JATLite.	35
3.2	Architecture de JATLite	38
3.3	Exemple de structure d'agent modulaire.	38
3.4	Couches de KQML.	40
3.5	Modes de routage de l'information.	47
4.1	Les couches abstraites de NetSA.	52
4.2	Architecture de NetSA	54
4.3	Description de l'agent utilisateur.	56
4.4	Architecture interne de l'agent utilisateur.	57
4.5	Description de l'agent ressource.	58
4.6	Architecture interne de l'agent resource.	59
4.7	Exemple de communication pour l'agent ressource.	60
4.8	Description de l'agent intermédiaire.	61
4.9	Architecture interne de l'agent Intermédiaire.	62
4.10	Description de l'agent superviseur.	64
4.11	Architecture interne de l'agent superviseur.	64
5.1	Exemple du fonctionnement de l'enchère anglaise.	69
5.2	Exemple du fonctionnement de l'enchère hollandaise.	71
5.3	Exemple du fonctionnement de l'enchère du meilleur prix.	73

5.4 Exemple du fonctionnement de l'enchère Vickrey.	75
5.5 Exemple du fonctionnement de l'enchère double.	77
7.1 Page d'accueil de NetSA.	84
7.2 Informations personnelles concernant l'utilisateur.	85
7.3 Informations sur la maison.	86
7.4 Dépenses de l'utilisateur.	87
7.5 Revenus de l'utilisateur.	88
7.6 Choix du type de prêt hypothécaire.	89
7.7 Agent utilisateur.	90
7.8 Agent superviseur.	91
7.9 Agent ressource.	92
7.10 Résultats de l'enchère <i>meilleur prix</i>	93
8.1 Comparaison entre une exécution séquentielle et concurrente des plans.	96
B.1 Fenêtre principale de l'agent superviseur.	109
B.2 Fenêtre d'envoi manuel de messages de l'agent superviseur.	110
B.3 Fenêtre de configuration pour l'agent.	110
B.4 Fenêtre de configuration pour le router.	111
B.5 Fenêtre de configuration pour le registrar.	111
B.6 Fenêtre de configuration générale.	112

Avant-propos

À travers ces quelques lignes, j'aimerais remercier toutes les personnes qui ont collaboré à l'aboutissement de ce mémoire. Mon directeur de recherche pour ses précieux conseils, sa disponibilité, sa grande patience, et son soutien indéfectible. Je lui suis reconnaissant pour tous les enseignements qu'il m'a apportés et pour la droiture avec laquelle il a dirigé le projet NetSA. C'est une personne exigeante mais ces exigences nous poussent à nous surpasser. À mes parents, mon frère et ma soeur à qui je dis trop peu souvent que je les aime. Et je tiens particulièrement à remercier ma compagne Julie qui a su supporter mes longues absences tout au long de la réalisation de ma maîtrise.

Chapitre 1

Introduction

Ce chapitre introduit de façon générale le présent mémoire de maîtrise. Le sujet traité dans ce mémoire est le développement d'un système multiagent d'aide à la décision. La première section présente la problématique constituée par la complexité des nouveaux systèmes informatiques distribués. Ensuite, viennent les objectifs de ce mémoire suivis de la méthodologie utilisée. La dernière section présente le contenu de ce mémoire.

1.1 Problématique

La satisfaction d'une demande d'information est devenue à la fois plus facile et plus compliquée. Elle est devenue plus facile dans la mesure où grâce à l'émergence de nouvelles sources de données, comme le réseau mondial appelé Internet, chacun, en principe, peut avoir accès à une source d'informations inépuisable. Cependant, la masse énorme d'informations disponibles sur Internet, même sur un Intranet ou un entrepôt de données, qui, à première vue, semble être sa force majeure, est en même temps l'une de ses faiblesses. La quantité d'informations à la disposition de l'utilisateur, généralement un décideur, est trop grande : l'information recherchée est probablement disponible quelque part, mais il arrive souvent qu'une seule partie soit retrouvée, et parfois même rien du tout. Les méthodes de recherche d'information conventionnelles se sont avérées incapables de résoudre ces problèmes. Ces méthodes supposent que nous connaissons d'avance quelle information est valable et où exactement elle peut être trouvée. De telles méthodes sont utilisées de la manière suivante : les systèmes d'informations, comme les bases de données, sont approvisionnés avec des indices qui fournissent ces informations aux usagers. Grâce à ces indices, l'utilisateur peut, à tout moment, vérifier si certaines informations sont offertes par la base de données, si elles sont disponibles, et où il peut les trouver. Avec les nouvelles technologies notamment Internet, mais aussi Intranet/Extranet et entrepôt de données, ces stratégies ne sont plus applicables. Les raisons à cela sont les suivantes :

- La nature dynamique d'Internet : aucune supervision centrale ne s'applique quant au développement d'Internet. Toute personne qui désire l'utiliser ou offrir des informations ou des services est libre de le faire. Ceci a créé une situation où il

est devenu très difficile d'avoir une idée claire sur la taille réelle d'Internet ;

- La nature dynamique des informations : les informations qui ne sont pas disponibles aujourd'hui peuvent être disponibles demain et le contraire s'applique également ;
- L'information est hétérogène : l'information est offerte sous plusieurs formats et de plusieurs façons. Ceci complique la recherche automatique d'une information donnée, puisque chaque format et chaque service nécessitent une approche particulière.

Comme d'autres technologies, l'évolution d'Internet est continue. Le volume des données sera trop grand et trop varié de telle façon qu'il sera impossible pour l'être humain de suivre ce qui se passe. Le pire, c'est que prochainement les logiciels conventionnels ne seront plus capables de maîtriser la situation, par conséquent une nouvelle structure pour la recherche d'informations s'avère dès aujourd'hui nécessaire. Une telle structure facilitera la tâche et fera abstraction des différentes techniques. Ce type d'abstraction est comparable à celui avec lequel les langages de programmation de haut niveau ont débarrassé les programmeurs de tous les problèmes de bas niveau (registres et appareils).

Etant donné, cependant, que le processus de réflexion relatif à ces idées est très récent, aucun standard n'est établi. Une idée prometteuse a cependant émergée ces dernières années. Cette idée consiste en un ensemble d'agents intelligents qui communiquent entre eux en vue d'une résolution de problèmes coopérative. C'est ce que nous appelons les systèmes multiagents.

1.2 Objectifs

Comme nous l'avons précédemment souligné, l'approche multiagent semble très prometteuse pour des domaines riches en informations. Partant de là, nous avons développé une architecture appelée NetSA basée sur la coopération entre agents.

Ce mémoire présente, de façon générale, l'étude des architectures d'agents et des systèmes multiagents adaptés à la recherche d'informations. Il présente également les enchères comme processus de négociation entre les agents. Pour réaliser cette étude, nous comptons atteindre les objectifs suivants :

1. étudier les agents et les systèmes multiagents,
2. étudier les architectures de systèmes multiagents pour la recherche d'informations,
3. créer une architecture de système multiagent,
4. étudier le principe des prêts hypothécaires,
5. étudier les différents types d'enchères,
6. appliquer l'architecture multiagent au courtage de prêts hypothécaires en utilisant les enchères.

1.3 Méthodologie

Avant de commencer toute recherche, un état des lieux ou un état de l'art s'impose. Dans notre cas, un tel état de l'art permet de comprendre le concept des agents et des systèmes multiagents, leurs architectures ainsi que les principes entourant les ventes par enchères et les prêts hypothécaires. En faisant la synthèse des travaux existants, nous avons élaboré une architecture d'agent baptisée NetSA (Networked Software Agents) pouvant être utilisé à plusieurs fins mais spécialement pour la recherche d'informations. Cette architecture a ensuite été utilisée pour le domaine financier et plus précisément pour le courtage de prêts hypothécaires. Dans ce cadre, nous avons personnellement conçu et développé un agent superviseur en charge de la planification et de la supervision dans NetSA. Nous avons ensuite développé des algorithmes d'enchères que nous avons utilisés dans NetSA pour le courtage des hypothèques entre plusieurs banques.

1.4 Contenu du mémoire

Le chapitre 2 introduit les notions d'agents et de systèmes multiagents. Dans ce cadre, les architectures d'agents cognitifs y sont présentées ainsi que les architectures multiagents oeuvrant dans des environnements riches en informations. Le chapitre 3 présente les aspects de communication entre agents. Plus précisément JAT-Lite [CDR99], un outil de communication pour les agents, et KQML [FFMM94], un langage d'échange d'informations entre agents, y sont expliqués. Le chapitre 4 décrit l'architecture multiagent NetSA et ses différents agents. Y sont plus particulièrement décrits, l'agent utilisateur, l'agent intermédiaire, l'agent ressource et l'agent superviseur. Le chapitre 5 présente différents types d'enchères. Les avantages et les inconvénients de chaque type d'enchère sont expliqués et son adéquation quant à son application aux prêts hypothécaires évaluée. Le chapitre 6 détaille le calcul des prêts hypothécaires canadiens. Le chapitre 7 est en quelques sorte une validation des chapitres précédents dans la mesure où il présente la construction d'une application financière construite autour de NetSA, des enchères et des prêts hypothécaires. Finalement, le chapitre 8 montre les résultats fournis par l'application vue au chapitre 7.

Chapitre 2

Agents et systèmes multiagent

2.1 Introduction

Depuis le début de l'informatique, l'homme rêve du jour où il pourra jouir pleinement de la vie en laissant aux machines le soin d'exécuter les tâches qui l'exaspèrent. Dans les années 70, les chercheurs en intelligence artificielle avaient prédit que les hommes de l'an 2000 utiliseraient des entités intelligentes pour exécuter les tâches de tous les jours. Ils ont cependant été confrontés à la complexité du raisonnement humain qui est très difficile à modéliser. Le découragement engendré en a fait renoncer plusieurs, qui maintenant affirment que l'obstacle dû à la représentation du raisonnement humain ne sera peut-être jamais franchi.

Or, les chercheurs pugnaces en intelligence artificielle sont ceux qui croient en ce qu'ils font et en ce sur quoi ils travaillent. Avec beaucoup d'acharnement et de persévérance, ils ont développé des prototypes et des concepts de plus en plus affinés. Ils ont observé leur environnement. Ils se sont questionnés sur comment les humains réfléchissaient et sur comment ils interagissaient. De ces observations et de ces réflexions est né le concept d'agent intelligent et plus tard celui de systèmes multiagents.

Dans les pages qui vont suivre, nous verrons ce que sont ces concepts. Nous verrons plus particulièrement les architectures formant des systèmes multiagents. Avant de commencer, il serait intéressant de comprendre quelques concepts comme les systèmes experts, les agents et les systèmes multiagents.

2.2 Agents intelligents

Le mot agent en informatique est devenu très populaire sans qu'on sache vraiment le définir. Présentement, personne n'a de définition précise concernant ce mot dans le contexte informatique. Cela toutefois n'empêche pas l'évolution des agents et des systèmes multiagents.

Le dictionnaire Larousse définit un agent de la manière suivante : « Personne chargée de gérer, d'administrer pour le compte d'autrui ». Par exemple, un agent d'assurance est un agent dans le sens où il représente une ou plusieurs compagnies d'assurance pour le compte desquelles il fait souscrire des contrats. Dans le cas qui nous préoccupe,

nous pourrons dire qu'un agent est un programme qui aide les utilisateurs en diminuant leur charge de travail. Nous serions donc tentés de dire que pratiquement tous les logiciels disponibles sur le marché sont des agents puisqu'ils effectuent pour les utilisateurs au moins un travail ou en réduisent la durée. Nous pouvons, à l'extrême, extrapoler sur le fait que tous les outils informatiques, manuels, électriques, électroniques, et autres sont des agents. En fait, la notion d'agent est encore floue et les chercheurs ne s'entendent pas encore sur la définition du concept d'agent.

Pour Jennings, Sycara et Wooldridge, par exemple, un agent est un système informatique situé dans un quelconque environnement capable d'exécuter des actions flexibles¹ et autonomes² dans le but d'accomplir les objectifs pour lesquels il a été créé [JSW98]. Smith définit l'agent comme une entité logicielle persistante dédiée à une utilisation précise [SCS94]. Selker considère les agents comme étant des programmes qui simulent les relations entre humains en faisant quelque chose qu'une autre personne pourrait faire [Sel94]. Janca voit les agents comme des entités logicielles à qui nous pouvons déléguer des tâches [Jan95]. Finalement Nwana décrit les agents comme étant des composants logiciels ou matériels capables d'agir pour accomplir certaines tâches pour son utilisateur attitré [Nwa96].

Ces définitions, bien que générales, convergent toutes vers une idée commune. L'agent a été créé pour servir son utilisateur. Nous pouvons alors ressortir quelques caractéristiques qu'un agent devrait posséder. Ces caractéristiques sont :

- *L'autonomie* : Un agent devrait être capable de résoudre la majeure partie de ses problèmes sans l'intervention d'un humain.
- *L'esprit Social* : Un agent devrait être en mesure de pouvoir communiquer avec d'autres agents ou avec les humains pour demander de l'assistance pour résoudre ses problèmes ou pour aider d'autres agents.
- *La connaissance de l'environnement* : Un agent devrait connaître (et être «conscient» de) son environnement pour s'adapter aux changements survenus dans ce dernier. Par exemple, si un agent suit jour après jour le même parcours et qu'un obstacle vient obstruer son passage, il devrait être en mesure de savoir comment contourner cet obstacle pour pouvoir poursuivre sa route.
- *L'initiative* : Un agent ne devrait pas seulement se contenter de répondre aux changements survenus dans son environnement, il devrait également être capable de prendre des initiatives et être opportuniste lorsque c'est nécessaire. Par exemple, si un agent surveille l'entrée de virus informatiques sur un réseau, il devrait prendre l'initiative de réparer quand c'est nécessaire ou de mettre en quarantaine les fichiers infectés.

2.2.1 Architecture d'agents

La structure interne d'un agent logiciel n'est pas simple. D'une façon générale, les agents ont une architecture (Figure 2.1) qui leur permet d'être réactifs, cognitifs ou

¹ La flexibilité d'un agent signifie que l'agent a conscience de l'environnement dans lequel il évolue et est capable de changer son comportement si son environnement change également.

² L'autonomie d'un agent implique qu'il est capable d'agir sans l'intervention direct d'autres entités.

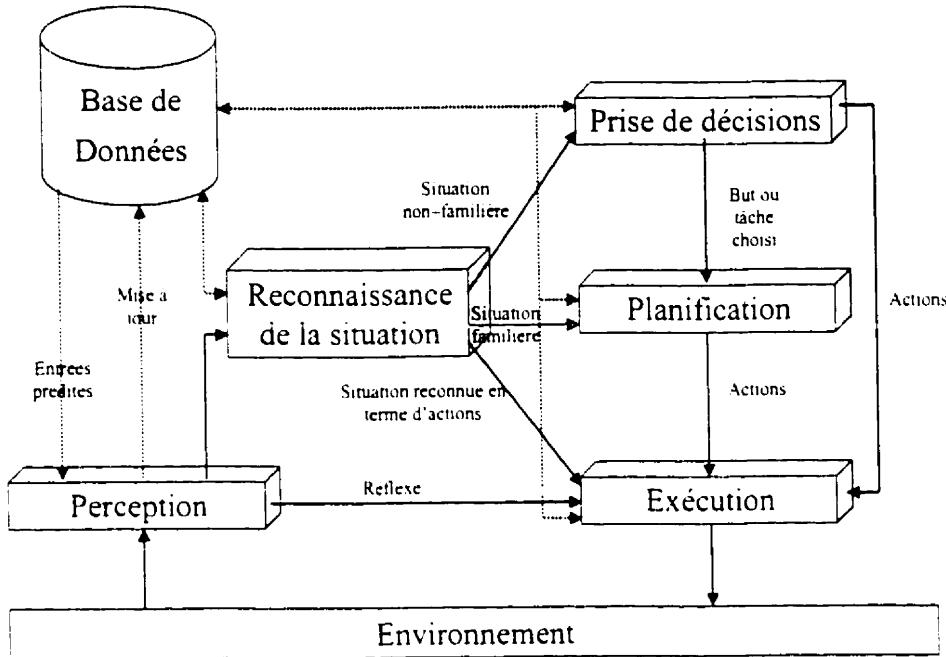


Figure 2.1: Architecture d'agent.

coopératifs.

Le raisonnement au niveau réactif est utilisé lorsque l'agent perçoit des situations qu'il connaît très bien. Nous pouvons comparer ce processus à celui du réflexe chez les êtres humains.

Le niveau cognitif est utilisé lorsque la situation perçue par l'agent lui est familière et suit un patron qu'il connaît. Par exemple, si l'agent sait comment se rendre d'un point A à un point B et sait comment se rendre d'un point B à un point C, si nous demandons à l'agent de se rendre au point C à partir du point A, il saura par un simple raisonnement comment réaliser cela.

Le niveau coopératif est la porte de secours de l'agent. Si un agent ne peut pas résoudre un problème (situation non-familière), il engage un processus de coopération pour demander de l'aide aux autres agents.

2.2.2 Agents et objets

Comme l'informatique va très vite, beaucoup de gens ont souvent la fâcheuse habitude de confondre les paradigmes. Par exemple : les objets et les agents. Souvent les chercheurs travaillant sur le concept «agent» se font demander la différence entre un objet et un agent. Un objet vient du paradigme de programmation orientée objet et un agent vient d'un paradigme formé de plusieurs disciplines comme l'intelligence artificielle, les systèmes distribués, le génie logiciel, etc. Un agent peut être constitué d'objets mais un objet n'est pas nécessairement un agent car un agent peut être programmé

avec plusieurs paradigmes comme la programmation fonctionnelle et la programmation logique.

Une des grandes différences entre les objets et les agents réside dans les demandes de services (ou appels de méthodes). Dans un objet, lors d'un appel de méthode (service), l'objet effectue la méthode et retourne une réponse. Pour un agent, la réalité est tout autre. Lorsque vous demandez à un agent d'exécuter une tâche, celui-ci, pour diverses raisons qui dépendent du contexte, peut refuser de faire une tâche. Il peut également exécuter une tâche sans avoir a priori une idée de la manière d'exécuter cette tâche. C'est en utilisant sa capacité sociale de demander de l'aide qu'il peut s'acquitter de telles demandes.

2.2.3 Agents et systèmes experts

La grande majorité des systèmes à base d'intelligence artificielle sont des systèmes à bases de connaissances et la majorité de ces systèmes sont des systèmes experts. Pour construire des systèmes experts, nous faisons appel au génie cognitif qui a pour but de déterminer les connaissances d'un expert dans un domaine précis afin de pouvoir les simuler. Le mot «connaissance» englobe à la fois des faits, des problèmes typiques et des heuristiques de résolution de problème. Au début des systèmes experts, le mot «connaissance» référait aux connaissances d'un expert acquises au terme d'une longue expérience. Ces connaissances sont souvent peu formalisées et inaccessibles au non-expert. En raison de cet intérêt porté aux connaissances spécialisées et floues, la méthodologie du génie cognitif repose souvent sur des recettes comme celles de Buchanan [HRWL83]. Pour construire un système expert, Waterman [Wat85] propose les étapes suivantes (ces étapes sont illustrées à la Figure 2.2) :

1. *L'identification* : il s'agit de définir les buts du projet, les tâches détaillées que le système devrait résoudre et identifier les contraintes techniques et pratiques (algorithme, machines, logiciels, personnel, temps, etc.)
2. *La conceptualisation et l'obtention des connaissances* : il faut identifier ou constituer un vocabulaire plus précis pour décrire une activité de résolution de problème. Les concepts, relations, heuristiques, etc. identifiés lors de l'étape précédente doivent être rendus explicites.
3. *La formalisation* : les concepts exprimés en langage naturel doivent être traduits dans un langage formel. Cela permettra d'exprimer un problème ainsi que les heuristiques utilisées pour le résoudre avec des expressions quasi formelles. Il faut ensuite rendre apparentes les interdépendances des données et des connaissances (flux d'information, identification de sous-problèmes, etc.). Dans le même temps, il faut choisir (ou construire) un logiciel approprié (un moteur d'inférence par exemple).
4. *L'implémentation* : lors de cette étape, il faut développer un système informatique en utilisant les connaissances finalisées. Dans bien des cas, il s'agit d'écrire des règles du type «Si ... Alors ...». Parfois, la construction d'un prototype peut s'avérer nécessaire pour tester le bon choix des outils et la pertinence de l'analyse formelle.

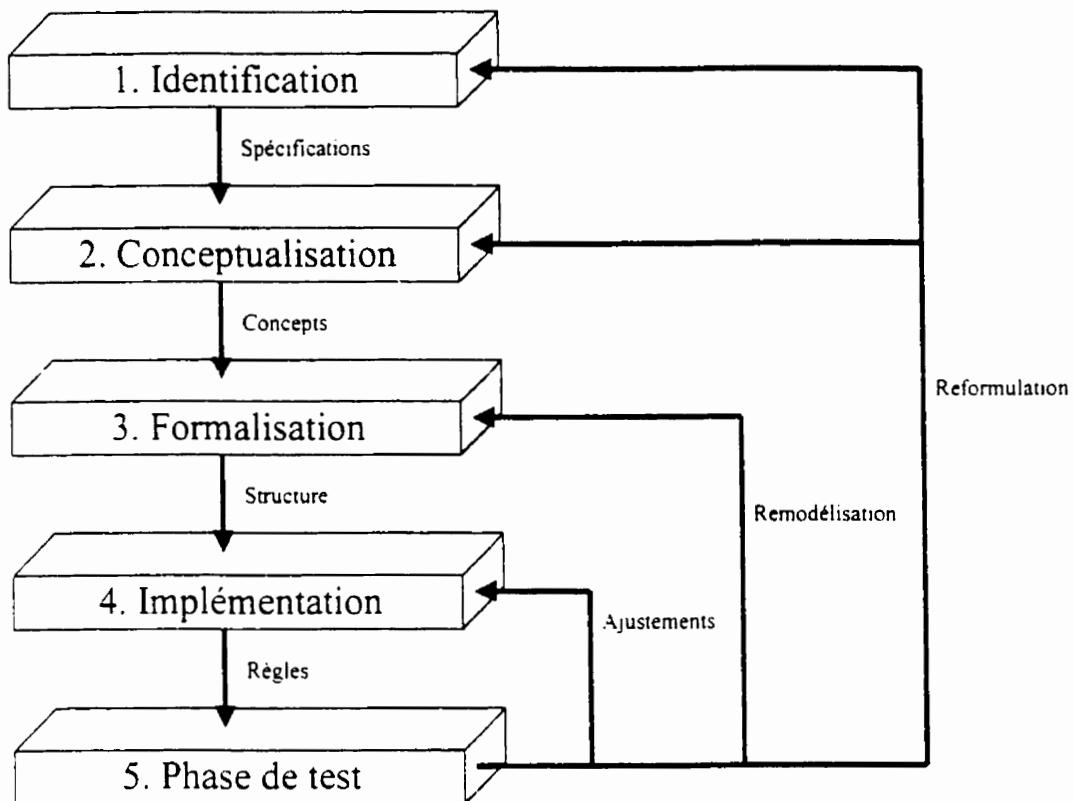


Figure 2.2: Les stades du génie cognitif selon Waterman.

5. *La phase test* : chaque version du système doit être testée extensivement. Cela peut conduire à de multiples ajustements et affinages du système informatique, à une remodélisation complète du système qui renvoie à l'étape 3 ou encore à une reconceptualisation des tâches qui renvoie aux étapes 1 et 2. Le savoir-faire est dynamique et ne peut être saisi que dans une simulation dynamique. Dans ce cas, les itérations paraissent inévitables.

Dans ces cinq phases du projet, l'ingénieur cogniticien doit travailler en coopération étroite avec l'expert. Une implication importante de l'expert est surtout inévitable pour les phases 2 (extraction du savoir) et 5 (test). Il est aussi conseillé de l'associer aux autres phases plus techniques, à condition de pouvoir l'intéresser à un langage plus formel.

Comme nous le voyons, la vision «classique» de l'ingénierie de la connaissance est très centrée sur le problème d'une bonne extraction des connaissances ainsi que de la modélisation. Le rôle clé dans ce dispositif appartient au cogniticien. C'est lui qui «gère» les trois niveaux du génie cognitif les plus importants :

1. *obtention de la connaissance* : il s'agit d'extraire les connaissances de l'expert ainsi que les connaissances provenant d'autres sources disponibles.

2. *analyse des connaissances* : il s'agit de structurer et d'organiser les connaissances qui sont normalement hautement non-structurées et de formes diverses. Au début du processus, il s'agit de voir d'abord quels sont les types de connaissance utilisés. Dans un deuxième temps, il faut préparer l'implémentation, en utilisant éventuellement une formalisation des connaissances.
3. *implémentation de la connaissance* : lors de cette étape, il faut traduire ces connaissances structurées dans un format informatique.

Nous utilisons souvent le terme «*acquisition des connaissances*» pour décrire l'extraction, la structuration, l'organisation et la traduction des connaissances.

Nous pouvons remarquer qu'un agent peut utiliser la technique du système expert pour concevoir des systèmes d'aide à la décision et résoudre des problèmes. Il existe cependant de grandes différences entre un système expert et un agent dans la mesure où le système expert est nécessairement bâti autour d'une expertise alors que l'agent fait généralement appel à des propriétés comme l'autonomie, la proactivité, la persistance, etc. Il est généralement conçu pour apporter de l'aide à l'utilisateur. Les agents augmentent leur autonomie en s'entraînant, ce qui n'est pas le cas des systèmes experts. Nous pouvons alors conclure que les systèmes experts ne sont pas des agents, mais un agent pourrait contenir une expertise et donc avoir le comportement d'un système expert.

2.3 Systèmes multiagents

Traditionnellement, les systèmes multiagents portaient la bannière de l'intelligence artificielle distribuée [MCD96]. Depuis peu, le terme système multiagent est utilisé pour décrire tout type de système composé de plusieurs composants logiciels plus ou moins autonomes. Précisément, les systèmes multiagents peuvent être définis comme étant un ensemble de plusieurs agents logiciels capables de résoudre certains problèmes et qui, en combinant leurs efforts, seront en mesure de résoudre des problèmes qu'un seul agent aurait été soit incapable de résoudre, soit capable de résoudre difficilement en y mettant le temps et les ressources nécessaires. Les agents dans un système multiagent sont autonomes et souvent de nature hétérogène.

Comme pour le cas des agents, un système multiagent comporte différentes caractéristiques. Les principales caractéristiques sont :

- chaque agent du système possède des informations et des compétences restreintes dans le cas de la résolution d'un problème donné ;
- il n'y a pas de système central de contrôle ;
- les données et les informations sont décentralisées et ;
- le fonctionnement du système multiagent est asynchrone.

Un des points forts des systèmes multiagents réside dans sa capacité à la tolérance aux erreurs. Si un agent dans le système tombe en panne, le système sera capable de continuer à fonctionner malgré tout, à la condition qu'un autre agent puisse prendre la relève. C'est pourquoi il n'y a pas de contrôle ou de quête d'informations centralisés.

Bien évidemment la perte de plusieurs agents peut compromettre les résultats, mais comme pour une voiture, la perte d'un boulon n'est pas critique, cependant la perte de plusieurs boulons peut être catastrophique.

Les systèmes multiagents bien qu'asynchrones n'oeuvrent pas pour autant dans un environnement anarchique. Une structure organisationnelle est souvent donnée aux systèmes multiagents. Ces structures sont appelées des architectures. Plusieurs architectures existent déjà. La section 2.5 nous dévoilera quelques unes des architectures existantes. De plus, le chapitre 4 est dédié entièrement à l'architecture nommée NetSA conçue dans le cadre du présent mémoire.

2.4 Architectures d'agents cognitifs

Le monde de l'informatique connaît depuis peu une certaine révolution dans le domaine de la recherche en intelligence artificielle. De nombreux chercheurs se ruent sur ce que l'on pourrait appeler le *Klondike* de l'informatique. Le monde des agents logiciels commence petit à petit à envahir les méthodes de résolutions de problèmes liées à l'intelligence artificielle distribuée. Ce phénomène en plein effervescence occupe déjà une bonne part des marchés mondiaux. Mais un tel engouement n'arrive jamais sans certains problèmes de compatibilité et de ré-utilisation du code. Dans ce cas, une standardisation au niveau de la conception et de l'architecture est nécessaire. Il faut donc créer des architectures génériques pour faciliter la conception d'agents.

Dans les lignes qui vont suivre, nous aborderons différents types d'architectures dans le but d'en faire une brève classification.

2.4.1 Les types d'architectures d'agents

Depuis une dizaine d'années, un grand nombre d'idées a survécu autour du concept «agent» et différents types d'architectures d'agents sont nés. Parmi ces derniers, on peut distinguer quatre types d'architecture représentant quatre types d'agents :

1. les agents réactifs,
2. les agents inductifs,
3. les agents coopératifs,
4. les agents hybrides.

Les agents réactifs

La représentation du monde en intelligence artificielle est parfois extrêmement difficile, ce qui cause énormément de problèmes puisque les agents réactifs se doivent de bien percevoir leur environnement. Ces problèmes ont amené les chercheurs à questionner la validité de tout le paradigme en vue de développer des architectures réactives. Dans ce cadre, ils ont dû définir une architecture réactive qui n'inclut aucune sorte de modèles symboliques centralisés et qui n'utilise pas de raisonnements symboliques complexes [WJ95]. En quelque sorte, l'agent réactif réagit aux stimuli venant du monde extérieur

en invoquant une réponse quasi-instantanée soit une réaction du type réflexe à partir d'une base de connaissances limitée et de règles d'inférence simples. Les travaux dans ce contexte ont donné lieu aux architectures suivantes :

	Applications	Créateurs
1)	Subsumption Architecture	Brooks [Bro89]
2)	Pengi	Agre & Chapman [AC87]
3)	Situated Automata	Kaebbling & Rosenschein [RK86]
4)	Dynamic Action Selection	Maes [Mae89]
5)	Behaviour Based System	Steels [Ste94]

Tableau 2.1: Les agents réactifs.

Les agents inductifs

Les agents inductifs sont définis comme un modèle symbolique explicitement représenté dans un certain monde et où les décisions sont faites via un raisonnement logique basé sur l'association de gabarit et la manipulation symbolique. L'idée d'avoir des agents inductifs purement basés sur le raisonnement logique est attrayante mais faut-il encore avoir la possibilité de traduire le problème sous une forme logique ce qui n'est pas toujours facile.

Par ailleurs, les problèmes énoncés semblent très difficiles à prouver avec la logique simple, et la complexité de manipulation des symboles en général n'est pas toujours décidable. Du moins, c'est hautement indécidable avec la logique du premier ordre. C'est pourquoi cette approche n'est pas l'approche optimale [WJ95]. À titre indicatif, mentionnons les exemples d'architecture suivants :

	Applications	Créateurs
1)	STRIPS	Fikes [FN71]
2)	IRMA	Bratman [BIP88]
3)	AGENT0	Shoham [Sho91]
4)	BDI Architecture	Rao & Georgeff [RG91]
5)	Phoenix	Cohen [HC92]
6)	ECO Models	Ferber [FJ91]
7)	AuRA Architecture	Arkin [Ark92]

Tableau 2.2: Les agents inductifs.

Les agents coopératifs

Ce type d'agent, fortement utilisé dans le cadre de l'intelligence artificielle distribuée, ne peut fonctionner seul. Il doit constamment communiquer avec d'autres agents afin de pouvoir résoudre un problème donné. La nécessité de communiquer provoque ici une dépendance au niveau de la rapidité et de l'efficacité des communications

ainsi qu'au niveau de la coordination et de la négociation. Voici quelques applications de ce type d'agent :

	Applications	Créateurs
1)	PGP	Decker [DL90]
2)	Negociation	Sycara & Conry [Syc87]
3)	Game Theory	Rosenschein [LR92]

Tableau 2.3: Les agents coopératifs.

Les agents hybrides

Comme dans la plupart des domaines de recherche, nous tentons de reprendre et d'agencer le meilleur des différentes recherches effectuées dans le même domaine. Plusieurs chercheurs ont suggéré qu'une approche mi-réactive et mi-inductive est souhaitable. Ils tentent par ce fait de marier les approches classiques et les approches alternatives [WJ95]. Mentionnons ici :

	Applications	Créateurs
1)	RaP	Firby [Fir95]
2)	Touring Machine	Ferguson [Fer92]
3)	GRATE	Jennings [Jen92]
4)	3T	Bonasso [BFG ⁺ 97]
5)	InteRRaP	DFKI [MP93]

Tableau 2.4: Les agents hybrides.

Durant les dernières années, la recherche sur les architectures multiagents s'est surtout focalisée sur les architectures cognitives dont InteRRaP est une excellente représentation. C'est pourquoi dans la section suivante nous donnons un aperçu de cette architecture.

2.4.2 Exemple d'architecture hybride : InteRRaP

InteRRaP est une architecture d'agent vouée à la mise en place de systèmes multiagents. Développée par le groupe de recherche sur les systèmes multiagents du DFKI en Allemagne, cette architecture comporte cinq composants principaux comme l'indique la Figure 2.4 [FMP95] :

1. l'interface avec le monde,
2. la base de connaissances,
3. la couche réactive,
4. la couche de planification locale,
5. la couche de planification coopérative.

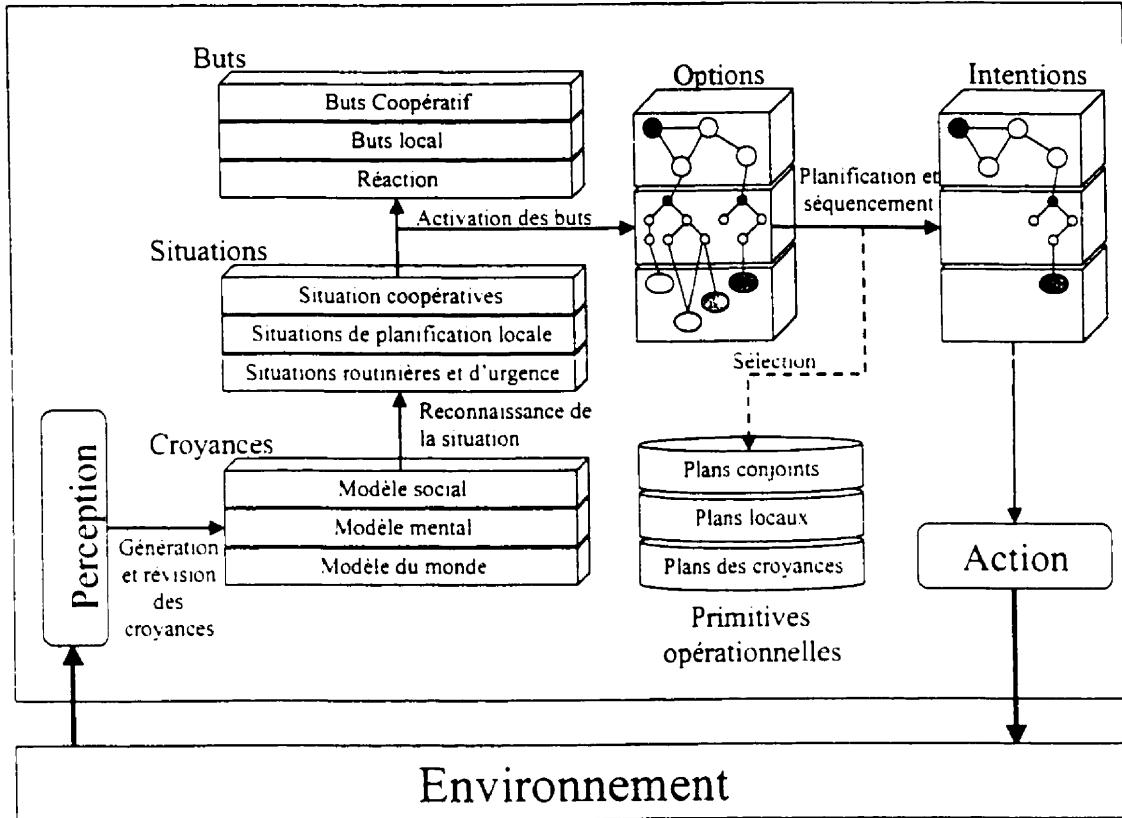


Figure 2.3: Modèle conceptuel d'InteRRaP.

Comme le montre la Figure 2.3, le fonctionnement de l'architecture est passablement simple. D'abord l'agent reçoit des informations venant de l'environnement. Avec ses informations et les croyances de l'agent, il tente de reconnaître le type de situation auquel il fait face ainsi que les buts visés par cette même situation. Il enclenche ensuite les buts qu'il pourrait atteindre. Parmi ces buts, il doit en choisir un seul et cela en filtrant les différentes options qui s'offrent à lui. Il cherche ensuite à élaborer un plan pour réaliser ce but et exécuter finalement le plan en question (le plan est vu comme une intention). Ceci se termine naturellement par l'exécution du plan sélectionné.

Pour arriver à structurer InteRRaP de cette manière, les chercheurs du DFKI ont mis de l'avant certaines exigences au niveau des agents autonomes et des systèmes multiagents, en particulier :

1. *réponse aux situations de réflexe* : l'agent doit pouvoir réagir aux stimuli provenant de l'extérieur de manière instinctive.
2. *réponse aux situations familières* : l'agent doit être en mesure de résoudre un problème par induction avec la base de connaissances qu'il possède.
3. *réponse aux situations nécessitant une coopération* : dans certaines circonstances, lorsqu'une assistance est requise, un agent doit être capable de coopérer avec d'autres agents pour obtenir ou fournir de l'information.

4. *réponse exacte* : parfois les contraintes de temps forcent les programmeurs à utiliser une approximation du résultat au lieu d'un calcul précis pour satisfaire les contraintes de temps. Il faut s'assurer que l'approximation soit le plus près possible de la réalité.

Dans une architecture d'agent autonome, la marche à suivre pour en arriver à la résolution d'un problème est la suivante :

1. *génération et révision des croyances* : l'agent doit d'abord agencer ses croyances avec sa perception du monde. Il doit en particulier générer ses propres croyances à partir de ce qu'il perçoit du monde, et les modifier de manière à tenir compte de la dynamique de l'environnement qui l'entoure.
2. *reconnaissance de la situation* : l'agent extrait les situations à partir de ses croyances.
3. *activation des buts* : l'agent choisit le but qui concorde le mieux avec la situation qu'il vient de reconnaître.
4. *planification* : l'agent élabore ensuite un plan pour atteindre le but poursuivi.
5. *coordination* : après avoir décidé quel type de réponse il doit utiliser, l'agent définit l'ordre d'exécution des différentes portions de plan. Il vérifie quelles portions de plan peuvent être exécutées en parallèle et quelles sont les ressources associées à telle ou telle portion de plan.
6. *exécution* : lorsque toutes les étapes précédentes ont pu être réalisées, l'agent peut exécuter les portions de plan selon les spécifications émises par les étapes de planification et de coordination.

Nous présentons maintenant les différentes éléments composant l'architecture d'InteRRaP comme cela est indiqué à la Figure 2.4. Nous verrons entre autre :

- l'interface avec le monde,
- la base de connaissances,
- la couche de contrôle,
- la couche de réflexe,
- la couche de planification locale,
- la couche de planification coopérative.

Interface (voir Figure 2.4)

L'interface avec le monde est un sous-système de l'agent qui fournit des méthodes de base pour percevoir son environnement, l'évolution des ses actions et la manipulation du côté communication avec les autres agents. L'interface se divise en trois sous-systèmes soit :

1. *le sous-système senseurs* : ce sous-système est composé de senseurs pouvant être calibrés, activés, désactivés et lus.

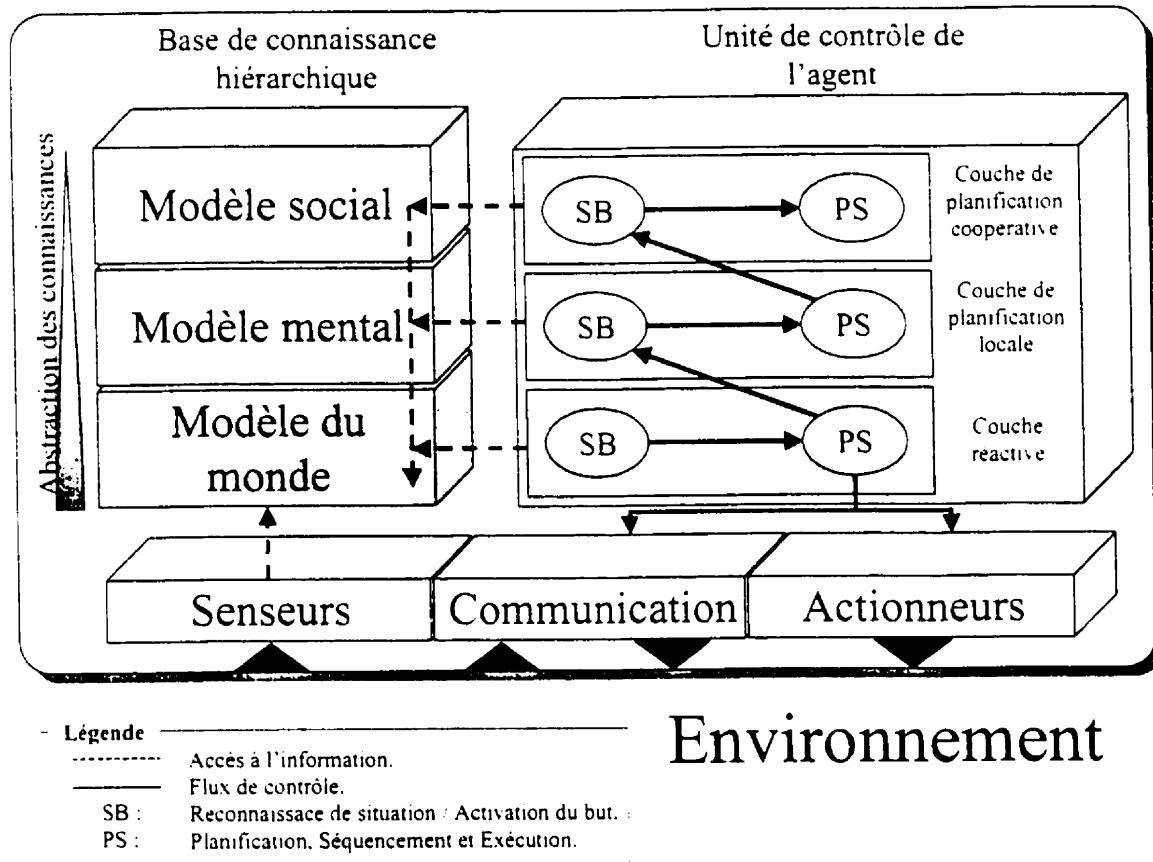


Figure 2.4: Architecture d'InteRRaP.

2. *le sous-système actionneurs* : ce sous-système est composé d'actionneurs pouvant contrôler les actions physiques que peut réaliser l'agent comme le contrôle des moteurs d'un robot par exemple.
3. *le sous-système de communication* : ce sous-système fournit les fonctionnalités nécessaires à l'expédition et la réception de messages venant d'autres agents.

Base de connaissances (voir Figure 2.4)

La base de connaissances est en quelque sorte une base de données où sont sauvegardées les différentes informations relatives à ce que l'agent connaît. Comme le montre la Figure 2.4, la base de connaissance d'un agent InteRRaP est partitionnée en trois couches.

1. *La couche du modèle du monde* est associée avec la couche réactive. Elle contient la représentation du monde dans lequel évolue l'agent.
2. *La couche du modèle mental* est associée avec la couche de planification locale. Elle emmagasine les connaissances et les croyances de l'agent. Elle est utilisée comme base de faits lors des processus d'induction.
3. *La couche du modèle social* est associée avec la couche de planification coopérative. On y retrouve les protocoles utilisés pour la communication entre les agents. Ces protocoles sont coopératifs ou compétitifs selon les besoins de l'agent.

Ces différentes couches de la base de connaissances montrent bien la séparation faite entre les différentes couches de l'architecture. La représentation des connaissances dans la base se fait à l'aide de AKB (Assertionnal Knowledge Base) [Wei95]. Ce type de représentation fournit trois types de services généraux :

1. *Services d'assertion* : Permettent de transformer des croyances en connaissances, de créer de nouveaux concepts et de nouvelles relations, de changer la valeur des concepts et des relations.
2. *Services de recherche* : Fournit un accès aux croyances déjà contenues dans la base de connaissances,
3. *Service d'information* : Offre la possibilité d'accéder à l'information sur demande. Au fur et à mesure qu'une information est changée dans la base de connaissances, un processus sentinelle envoie un avis de ce changement aux couches de contrôle qui en ont fait la demande.

Il y a un transfert constant d'informations entre l'interface et l'unité de contrôle. Par la perception de l'interface, la base de connaissances peut acquérir de nouvelles croyances. Par la suite, l'unité de contrôle explorant les nouvelles croyances, tentent de se définir de nouveaux buts. Étant donné la faiblesse du système d'inférence de AKB, l'architecture InteRRaP possède ses propres gabarits de croyances (PoBs : Patterns of Beliefs).

Couche de contrôle (voir Figure 2.4)

Toutes les couches d'InteRRaP ont une composition similaire et suivent les fonctions de base d'un agent : la reconnaissance de la situation, l'activation du but, la planification, la coordination et l'exécution.

La couche doit d'abord reconnaître la situation et construire un but. Ensuite, elle décide si elle peut ou pas compléter ce but. Si elle peut, elle débute la planification, coordonne les plans conçus et les exécute. Sinon, elle demande à une couche d'un niveau supérieur d'effectuer le but. Après l'exécution du but, le résultat est rétroactivement retourné pour une vérification de l'effet provoqué par le but. Ainsi, la couche peut s'assurer que le but a bien été atteint.

Résolution des buts : Le mécanisme de résolution d'un but se fait par une méthode ascendante. La couche la plus basse détecte un but et vérifie si elle a la capacité de le résoudre. Dans le cas où elle ne peut pas résoudre un but, elle demande à la couche supérieure immédiate de le faire. Il en va de même pour cette dernière qui utilise le même procédé de résolution. Bien sûr, la dernière couche (la couche de planification coopérative) ne peut pas transmettre à une couche supérieure son but. Alors, c'est à ce moment que les processus de coopération entrent en action comme nous le verrons plus loin.

Exécution de l'action : Pour réaliser un but donné, des actions doivent être posées. Pour cela, InteRRaP utilise une méthode descendante. La couche qui a pu résoudre le but retourne le résultat à la couche lui qui lui a fait la requête. Et ce processus boucle jusqu'à la couche de premier niveau qui effectuera les actions relatives au résultat obtenu.

Couche réactive (voir Figure 2.4)

La couche réactive est utilisée pour deux fonctions. Premièrement, elle inculque un comportement de type réflexe (réactif) à l'agent, lui permettant ainsi de réagir en temps réel à une série d'urgences. Deuxièmement, elle fournit la connaissance procédurale pour exécuter efficacement des tâches routinières. Cette couche interagit avec la couche de planification locale par l'appel de fonctions locales et pour lui demander de résoudre un but dans le cas où elle ne peut le faire elle-même. La partie *modèle du monde* de la base de connaissances est accessible par la couche réactive. Cette partie est composée d'une suite de situations/réactions (PoBs) pré-définies et d'un algorithme de recherche performant.

Fonctionnement : Lorsqu'une situation d'urgence est nécessaire, l'agent recherche dans la base de connaissances, à l'aide de l'algorithme, s'il n'y a pas une situation qu'il connaît déjà. S'il la connaît, il planifie, coordonne et envoie immédiatement les actions permettant de répondre à la situation d'urgence à l'acteur situé dans l'interface du monde. Ce dernier se chargera d'exécuter le plan d'action. Sinon, il passe le but à la couche de planification locale qui tentera à son tour de résoudre le problème et qui lui

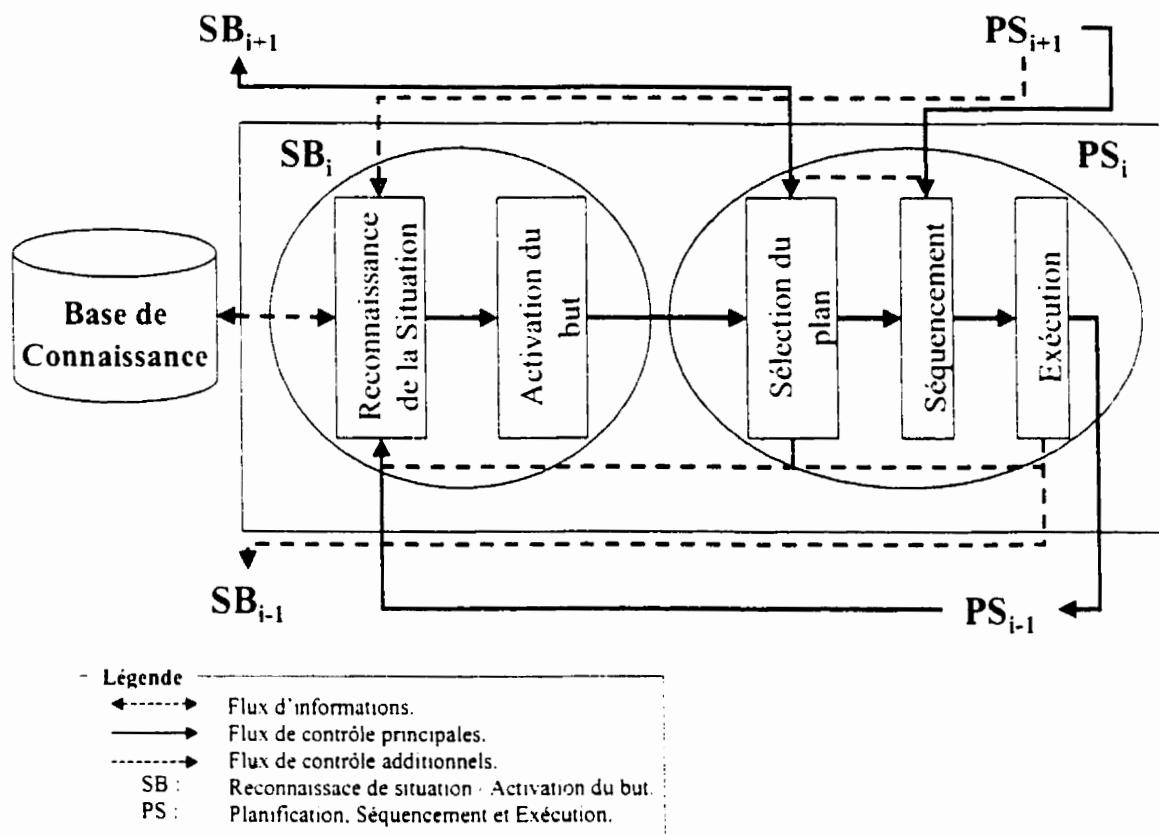


Figure 2.5: Couche de contrôle d'InteRRaP.

enverra le résultat trouvé. Ce résultat sera transmis par la suite à l'acteur pour son exécution.

Couche de planification locale

Le but de la couche de planification locale (voir Figure 2.6) est de donner à l'agent les aptitudes pour combiner des plans en vue d'atteindre un but local ou accomplir une tâche locale. Elle utilise le *modèle mental* de la base de connaissances pour effectuer un raisonnement sur le problème en cours.

Reconnaissance de la situation et Activation du but : Lorsque la couche réactive ne reconnaît pas une situation, elle fait une requête à la couche de planification locale. La couche réactive peut envoyer les messages suivants : (1) activer une requête, (2) annuler une requête antérieure, (3) acquitter l'exécution d'un message, (4) etc. Si la couche de planification ne peut elle-même atteindre ce but, elle demande à la couche de planification coopérative de l'aide pour la réalisation de ce but. Sinon, le processus de raisonnement est enclenché.

Sélection du plan : Une sélection d'un plan se fait à partir de la bibliothèque de plans. Si plus d'un plan est sélectionné, un seul doit être choisi pour des fins d'exécution. En principe, un choix aléatoire peut être fait car ce choix n'est pas critique. Mais dans les faits, on choisira le plan qui aura la meilleure utilité donc celui qui aura le coût moindre. Le plan sélectionné est par la suite déposé dans une pile de plans.

Interprétation de plan : La couche de planification locale doit interpréter le plan sélectionné, c'est-à-dire qu'elle doit déterminer quelle sera la prochaine action à être exécutée. Pour chaque plan situé dans la pile de plans, une interprétation doit être faite.

Coordination et exécution des plans : Bien que les plans aient été choisis et interprétés, il faut maintenant fixer un ordre d'exécution pour éviter les conflits ainsi qu'un « timing » d'exécution (quand faudra-t-il exécuter et pour combien de temps). Selon le domaine considéré, la complexité de la coordination varie énormément. Il est évident qu'un agent qui, par exemple, presse des oranges n'a pas la même complexité qu'un agent chargé de piloter un avion. Et pour terminer, il faudra exécuter les plans en respectant l'ordre émis par la coordination.

Couche de planification coopérative :

Dans un environnement multiagent, les agents doivent co-exister et coordonner ensemble leurs activités. Un bon outil de la coordination est la négociation et une pré-condition à la négociation est la communication. La couche de planification coopérative d'InteRRaP (voir Figure 2.7) possède un ensemble de mécanismes généraux de négociation lui permettant d'accepter des buts communs, d'allouer des tâches, résoudre des

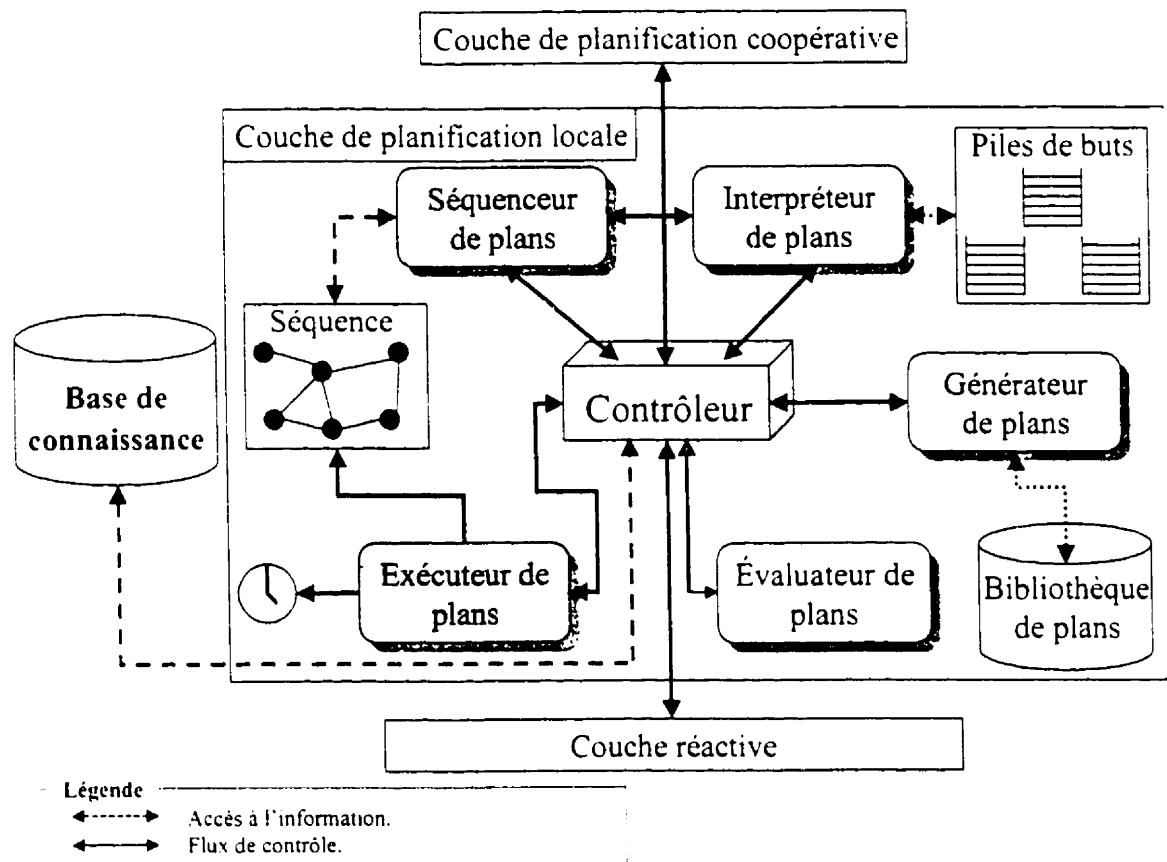


Figure 2.6: Couche de planification local d'InteRRaP.

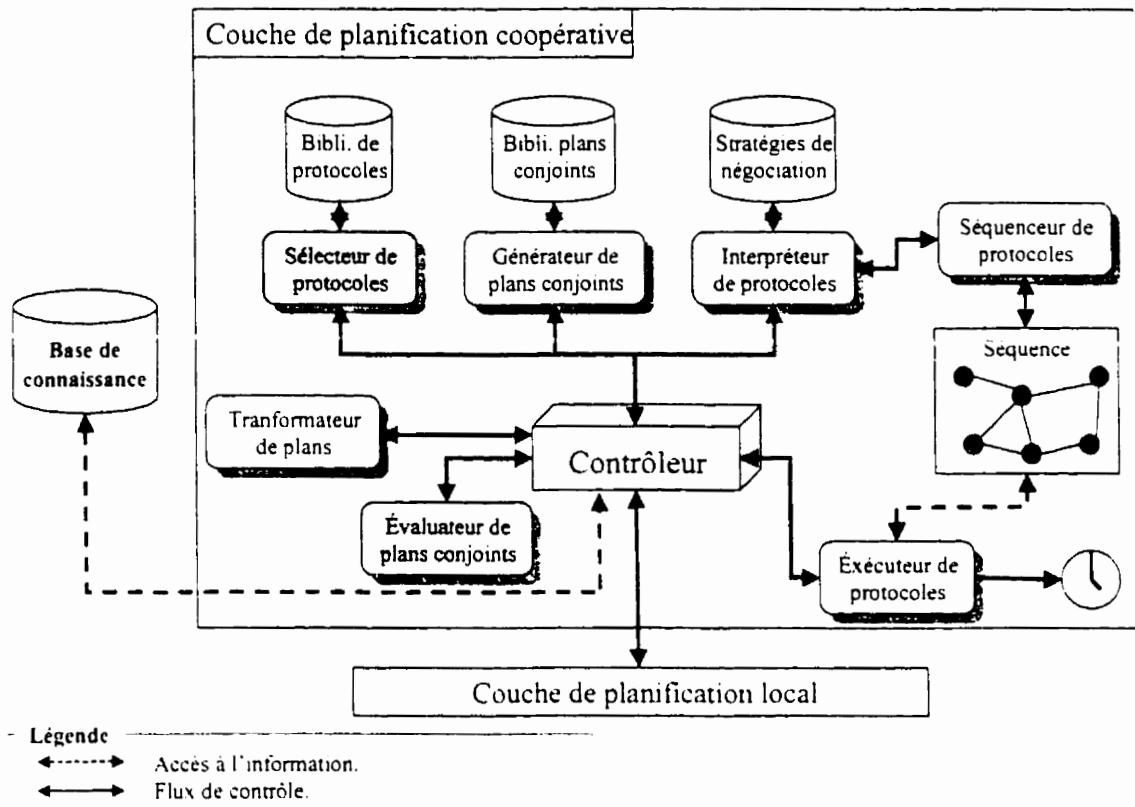


Figure 2.7: Couche de planification coopérative.

conflits et coordonner les plans locaux (qui sont situés dans la partie *modèle social* de la base de connaissances). Cette couche a pour fonction de reconnaître les situations d'interaction et de dériver des buts à partir de situation reconnue. Elle contient aussi des outils pour sélectionner des protocoles et des stratégies de négociation et pour accepter les requêtes des autres agents.

L'activation de la couche de planification coopérative se fait à la suite d'un échec de résolution de la part de toutes les autres couches. Donc les requêtes viennent de la couche de planification locale qui est la dernière à avoir tenté de résoudre le but.

Négociation : La négociation est un mécanisme permettant à deux agents de trouver une entente pour la réalisation d'un but (pour résoudre un conflit). Les auteurs d'InteRRaP ont adopté KQML [FFMM94] comme langage de communication. Toutes les requêtes sont faites dans ce langage. Il utilise également le *Contract Net* [Smi77] comme protocole de négociation.

2.4.3 Conclusion sur les architectures du type InteRRaP

La conception de telles architectures d'agents dites génériques est une bonne chose dans la mesure où il est facile de construire des agents réactifs, inductifs, coopératifs

ou hybrides.

Il faut cependant être prudent avec cette généralité. L'utilisation d'un tel moule comporte des inconvénients lorsque l'on veut innover. Ces moules exigent souvent de gros investissements et sont souvent hermétiques aux changements. La généralité apporte également une lourdeur inutile au système lorsque plusieurs parties et fonctionnalités du moule ne sont pas utilisées.

2.5 Architectures multiagents dédiées aux environnements riches en information.

Construire un système multiagent n'est pas une chose simple en soi. Le niveau de complexité qu'il engendre est parfois supérieur aux grands projets informatiques. De ce fait, le coût de production d'un système multiagent est énorme en personne/mois tant au niveau de la conception, que de la réalisation et de la vérification (déverminage). Il nous faut donc utiliser des moyens pour réduire la charge de travail et par le fait même le coût de production. Une des nombreuses solutions apportées est la réutilisation d'une architecture de système multiagent qui soit portable et universelle.

Ce chapitre présente quelques architectures multiagents existantes. Ces architectures sont le fruit de grands projets de recherche. Le chapitre 4 montre l'architecture développée au cours de ce projet de maîtrise au sein du groupe de recherche DAMAS de l'Université Laval.

2.5.1 RETSINA

RETSINA nous vient de l'Université Carnegie-Mellon. De nombreux chercheurs tel que Sycara, Decker, Pannu, Williamson et Zeng y ont travaillé, et certains y travaillent toujours. Les auteurs de RETSINA [SPW+96] ont développé un ensemble d'agents logiciels coopérants de manière asynchrone pour la quête d'information et pour l'intégration de prises de décisions variées, telles que l'aide à la décision dans les organisations, la gestion de porte-feuille d'action, etc.

Caractéristiques de Retsina

Les concepteurs de RETSINA avaient mis de l'avant huit caractéristiques, à savoir :

1. *Sources d'informations distribuées* : Les sources d'informations publiques étant hétérogènes et dynamiques, RETSINA devait être en mesure de s'adapter à la mobilité de l'information et à son évolution.
2. *Le partage des ressources* : Un agent de RETSINA peut être très en demande ; il doit donc pouvoir partager ses ressources avec plusieurs agents et cela de façon asynchrone.
3. *L'abstraction* : La résolution de problèmes par un système multiagent est assez complexe. Une telle complexité doit être transparente aux yeux des utilisateurs.

4. *Modularité et réutilisation* : Lorsque nous concevons une architecture multiagent, il serait utile de pouvoir réutiliser le système à d'autres fins en ne modifiant que certains modules du système.
5. *Flexibilité* : Les agents du système devraient pouvoir réagir à une nouvelle configuration sur demande.
6. *Fiabilité* : Le système doit être tolérant aux erreurs donc si un agent est mis hors tension, le système ne devrait pas être affecté.
7. *Qualité de l'information* : Il faut s'assurer de la validité de l'information recueillie.
8. *Cohérence des données* : Il faut vérifier que les données recueillies n'entreront pas en conflit avec les données existantes.

Types d'agent

Comme toute architecture multiagent, Retsina comporte divers agents (voir Figure 2.8) remplissant chacun une fonction spécifique.

Agents interfaces : Les agents interfaces interagissent avec les utilisateurs. Différentes tâches leur sont attribuées comme par exemple :

- recueillir l'information minimum nécessaire pour initier la résolution de problèmes ;
- présenter les résultats obtenus ;
- demander des informations supplémentaires si nécessaire durant la résolution de problèmes et ;
- demander à l'utilisateur certaines confirmations.

Agents-tâches : Les agents de ce type s'occupent de l'aide à la décision en formulant et exécutant des plans pour la résolution du problème visé. Ils ont une bonne connaissance d'un domaine particulier et peuvent aider les autres agents sur ce domaine. Un agent de tâches est apte à :

- recevoir les tâches déléguées par l'utilisateur à l'agent interface ;
- interpréter la tâche et en extraire le but ;
- construire un plan pour satisfaire ce but ;
- identifier les besoins en information des sous-but du plan construit ;
- décomposer et exécuter le plan construit.

Agents d'informations : L'agent d'informations fournit un accès intelligent aux données hétérogènes et réparties. Il est capable d'extraire l'information pertinente, résoudre les conflits et faire la fusion de certaines données. Cet agent est aussi capable de :

- fournir de l'information au sujet de la source qui lui est attribuée ;
- répondre à des requêtes périodiques et répétitives sans consultation de la source d'information ;

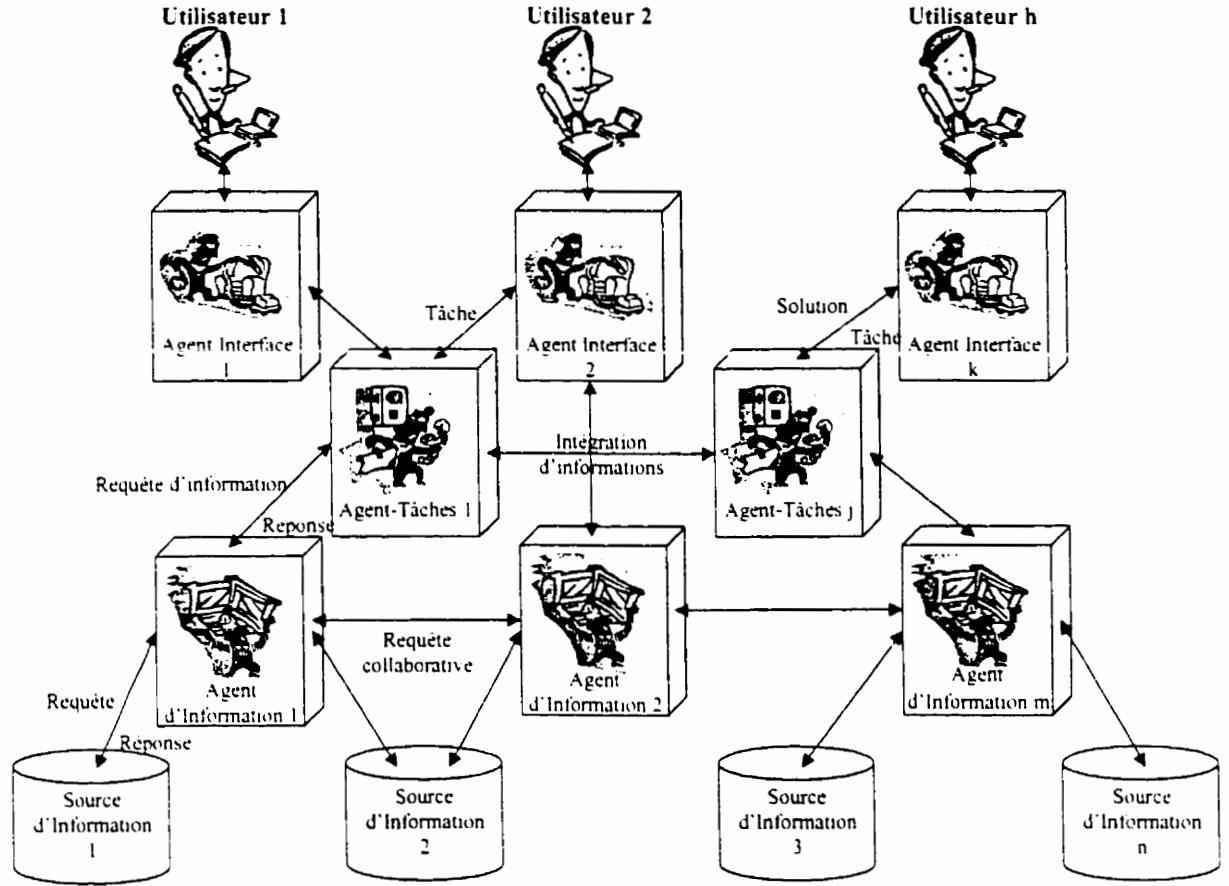


Figure 2.8: Architecture de Retsina.

- surveiller l'évolution d'une donnée pour ensuite avertir un agent requérant.

Retsina semble être une bonne architecture pour la recherche d'information. elle comporte cependant certaines lacunes au niveau de l'optimisation de communication. Par l'absence d'un agent intermédiaire pour diriger immédiatement les agents vers la bonne ressource, les agents de Retsina doivent instaurer à chaque requête un processus de négociation par *Contract Net* [Smi77] ce qui peut surcharger les réseaux. Cette absence d'agent intermédiaire rend également l'ouverture du système beaucoup plus complexe. Un nouvel agent désirant entrer sur Retsina doit aviser tous les agents inclus dans l'architecture de sa venue. Les auteurs ont également fait mention d'un problème de goulot d'étranglement au niveau des agents-tâches probablement dû à la complexité de l'architecture interne des agents.

2.5.2 Carnot

En 1990, la compagnie Microelectronics and Computer Technology Corporation (MCC) a eu pour mission de résoudre un problème d'intégration d'information dans

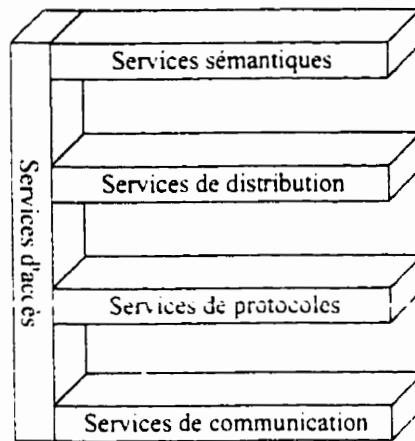


Figure 2.9: Architecture de Carnot.

des bases de données hétérogènes. Ce projet a reçu le nom de Carnot et s'est terminé en 1995 [WCH⁻93]. Carnot pouvait d'ores et déjà gérer le flux d'information, accéder aux bases de données hétérogènes et faire de la découverte d'information pour certaines entreprises.

Carnot a été développée autour d'un ensemble de couches génériques se concentrant sur le problème du management et de l'intégration de l'information de l'entreprise. Ces couches sont organisées suivant cinq ensembles de services comme le montre la Figure 2.9 :

- les services de communication,
- les services de protocoles,
- les services de distribution,
- les services sémantiques,
- les services d'accès.

Services de communication : Les services de communication fournissent à l'utilisateur une méthode constante d'interconnection hétérogène entre le matériel et les ressources. Ces services rendent la communication effective et intègrent plusieurs plate-formes pouvant être rencontrées dans une entreprise. C'est une couche de bas niveau pour la communication.

Services de protocoles : Les services de protocoles fournissent les protocoles de base issus des réseaux à grandes distances et les rendent accessibles aux autres niveaux de services. Ces services gèrent plusieurs normes comme X.500, X.400, ORB, SNMP, RDA, et plusieurs autres.

Services de distribution : Les services de distribution sont des moteurs de transactions permissifs (contrôle l'inconsistance de l'information). Ils acceptent et transmettent des requêtes selon un plan de résolution de problèmes pré-établi.

Services sémantiques : Les services sémantiques fournissent une vue globale de toutes les ressources intégrées dans le système Carnot. Leur but est de fournir une même ontologie pour un domaine donné dans le but d'éviter les inconsistances de langage pouvant nuire à la communication entre les services.

Services d'accès : Les services d'accès prévoient des mécanismes de manipulation des quatre autres services Carnot. Ces services sont une interface de contrôle des autres couches. La supervision du bon fonctionnement du système est gérée par ces services.

Carnot a constitué en fait un premier pas vers l'architecture multiagent appelée InfoSleuth.

2.5.3 InfoSleuth

La manque de structure dans la présentation de l'information sur Internet est un énorme défi du point de vue découverte, accès, mise à jour et analyse de l'information. Ceci a alors amené les chercheurs de MCC à chercher à améliorer Carnot en vue de lui permettre la recherche d'informations dans des sources géographiquement distantes et dynamiques comme dans le cas de Web. InfoSleuth [BBB⁺96] a donc pris la relève de Carnot en 1995. Pour développer InfoSleuth, les chercheurs de MCC ont utilisé des technologies standardisées comme :

- KQML (Knowledge Query and Manipulation Language) [FFMM94],
- KIF (Knowledge Interface Format) [Gea92],
- HTTP (Hyper Text Transfert Protocol) [BLFF96],
- Java [GM95].

Ceci procure à InfoSleuth un haut niveau d'ouverture du système, de flexibilité et d'extensibilité.

L'Architecture haut niveau d'InfoSleuth

L'architecture générale d'InfoSleuth est divisée en trois couches principales (Voir Figure 2.10) : (1) la couche agent, (2) la couche sémantique (ontologies) et (3) la couche application.

La couche agent : La couche inférieure de l'architecture est la couche agent. Cette couche est composée d'agents coopératifs où les règles d'induction sont implantées en LISP [AGST75], CLIPS [Ril91], LDL++ [Zan91] et Java. Ces agents annoncent d'abord leurs services et font par la suite des requêtes afin de trouver un agent pouvant répondre à un besoin spécifique. Un besoin peut être divisé en plusieurs sous-requêtes qui sont ensuite distribuées aux agents appropriés, les réponses provenant de ces derniers sont finalement réunies par l'agent demandeur pour l'obtention de la réponse à la requête originale.

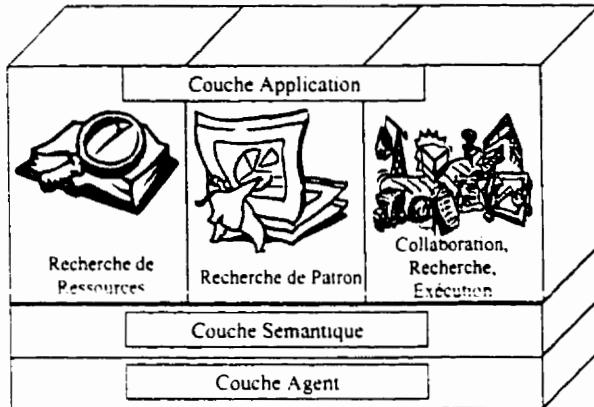


Figure 2.10: Architecture Haut-Niveau d'InfoSleuth.

La couche sémantique : Cette couche indique les agents coopératifs ayant le même vocabulaire et le même modèle sémantique qui sont capables d'interagir dans un domaine particulier. Par exemple, dans le cas où le travail se fait sur le domaine des finances, alors les agents aptes à communiquer entre eux sont seulement ceux utilisant des termes propres à la finance.

La couche application : Cette couche couvre trois régions principales soit :

- la recherche de ressources (par exemple trouver les agents et les bases de données relatives au domaine de l'ontologie),
- la recherche de gabarits et l'analyse (par exemple par data-mining),
- la collaboration, la recherche et l'exécution.

Ces régions s'interrelient et se chevauchent.

Architecture des agents

Le cœur d'InfoSleuth peut être vu comme un nuage d'agents (Voir Figure 2.11). L'utilisateur demande des requêtes et reçoit des réponses de ce nuage par l'intermédiaire d'un agent utilisateur. Ce nuage puise ces informations des agents ressource qui communiquent directement avec des bases de données. La communication entre agents se fait à l'aide du protocole KQML [FFMM94] en utilisant le format KIF [Gea92] dans son champ contenu. Le nuage comprend en fait différents agents que nous allons maintenant détailler en plus des agents utilisateur et ressource.

Agent utilisateur : L'agent utilisateur est la porte d'entrée des requêtes externes au système. Il fournit à l'utilisateur l'applet [GM95] lui permettant de faire facilement une requête. L'agent traduit ensuite le formulaire soumis dans le protocole KQML/KIF en

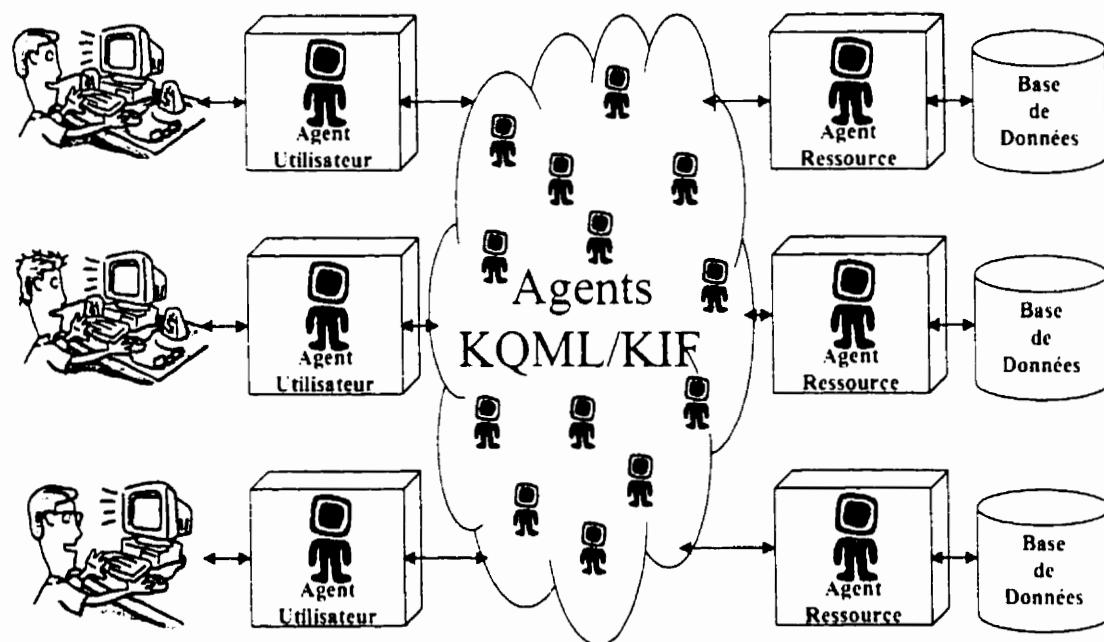


Figure 2.11: Nuage d'agents d'InfoSleuth(d'après [BBB⁺96]).

vue de son utilisation dans le système. L'agent utilisateur est persévérent et autonome dans le sens où il est capable de garder le contexte de l'utilisateur derrière une simple session de navigateur Internet, faire des requêtes à long terme et d'autres tâches qu'il peut effectuer en l'absence de l'utilisateur. Il est capable de stocker de l'information pour l'utilisateur, de conserver un profil de l'utilisateur et d'agir comme un agent ressource. Cet agent est une application Java [GM95].

Moniteur : Le moniteur est un serveur HTTP ayant deux rôles principaux :

1. surveiller les sites Internet visités par l'utilisateur et en aviser l'agent utilisateur afin qu'il puisse compléter sa base de connaissances et trouver des patrons de détection. Ces patrons permettent à l'agent utilisateur de détecter l'information pertinente contenue par exemple dans une page Web. En espionnant et prenant note des pages visitées par l'utilisateur, l'agent utilisateur peut connaître les intérêts de celui-ci et élaborer une stratégie sur la base de ces intérêts.
2. accepter des applets de l'agent utilisateur et les placer dans le répertoire approprié afin qu'ils soient accessibles à l'utilisateur étant donné les restrictions de sécurité des différents navigateurs Internet. Ceci n'est qu'un moyen de contourner la sécurité des navigateurs Internet qui empêche les applets d'écrire sur les disques locaux d'un ordinateur pour les protéger des applets malicieuses.

Cet agent est aussi une application Java.

Agent courtier : L'agent courtier associe aux différentes requêtes les agents qui sont capables d'y répondre. Les agents ressources peuvent s'y inscrire en envoyant un message en KQML qui avertira l'agent courtier du type de service fourni par le nouvel agent ressource. Lorsqu'un agent demande à l'agent courtier qui peut faire sa requête, celui-ci lui répond en lui donnant le nom du ou des agents aptes à faire la requête. Le courtier est une application Java.

Serveur d'ontologie : Le serveur d'ontologie a pour responsabilité de gérer la création, la mise à jour et la formation de requêtes d'une multitude d'ontologies. Le format KIF est utilisé aussi bien pour interroger l'ontologie que pour exprimer les résultats de l'interrogation. Les différents formats d'ontologies sont exprimés via un méta modèle d'ontologie. Le serveur est une application Java.

Agent d'exécution : L'agent d'exécution est responsable de l'exécution haut niveau des requêtes basées sur l'ontologie. Il décompose les requêtes en sous-requêtes et les distribue aux agents ressources pouvant y répondre. Il réunit ensuite les réponses de ces sous-requêtes pour former la réponse à la requête d'origine. L'agent d'exécution est programmé en Java avec des fonctions CLIPS [Ril91].

Agent ressource : L'agent ressource est aux données ce que l'agent utilisateur est à l'utilisateur. Cet agent reçoit des requêtes formulées en KQML et les transforme en requête SQL [HC96] ou LDL++ [Zan91] afin d'extraire l'information requise des

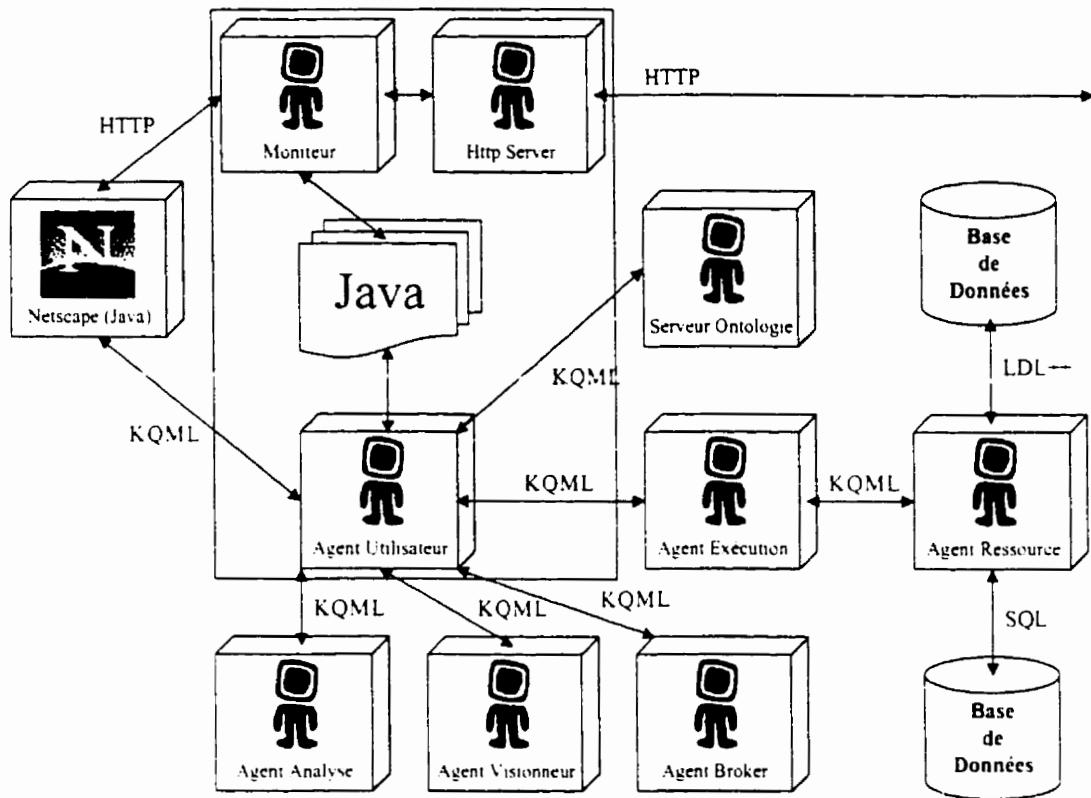


Figure 2.12: Architecture d'InfoSleuth.

bases de données. L'information trouvée est ensuite traduite en KQML en réponse à la requête du demandeur. Comme pour l'agent utilisateur, l'agent ressource garde un contexte des ressources permettant des recherches incrémentales. De plus, il peut obtenir, stocker et avertir un agent sur la méta-information au sujet de la ressource locale. L'agent ressource est constitué de Java et de LDL++ avec des versions ODBC et SQL. Actuellement une version SQL est implémentée et une version JDBC [HC96] est prévue à moyen terme.

Agent d'analyse des données : L'agent d'analyse des données accomplit une variété d'analyse, d'extraction de connaissances et de reconnaissance de patrons sur l'information renvoyée par les requêtes. Cet agent utilise différents langages de programmation comme Java, CLIPS, LDL++ et LISP [AGST75].

Serveur visionneur : Le serveur visionneur est un cas particulier d'agent ressource. Il gère et manipule toutes les applets qui ont trait à une ontologie particulière. C'est un courtier d'applets utilisé dans les applications de base de données, dans l'édition des requêtes et comme outil de visualisation pour la manipulation et l'affichage des requêtes. Ce serveur est une application Java.

2.5.4 UMDL

En 1994, une équipe multi-disciplinaire de l'université du Michigan propose de créer une bibliothèque numérique évolutive [VD95]. Le terme «bibliothèque numérique» est le nom générique pour décrire des structures qui fournissent aux êtres humains des accès intellectuels et physiques aux énormes réseaux mondiaux de l'information dans des formats numériques multimédias.

La mission fondamentale des bibliothèques a été de conserver et de fournir un accès intellectuel et physique aux écrits. Bien que cette mission fondamentale continue à être très importante, la représentation de l'information a changé radicalement (document électronique, matériel optique et magnétique, etc.). Ce changement massif de la représentation change la structure de classement, de recherche, d'utilisation et de réutilisation de l'information développée par les bibliothèques classiques. Ceci dit, une bibliothèque numérique est en mesure :

- de fournir de l'information peu importe l'endroit et le moment,
- de fournir l'accès à des collections d'information multimédia construites sur l'intégration de textes, d'images, de graphiques, de sons, de vidéos et d'autres médias.
- d'être assez convivial pour assister l'utilisateur dans sa recherche d'information.
- d'être le cœur d'une nouvelle technologie d'auto-apprentissage et de recherche abaissant les barrières géographiques et temporelles.

L'architecture d'UMDL est composée de 4 types d'agents logiciels comme le montre la Figure 2.13.

Agent utilisateur : L'agent utilisateur fournit les stratégies de recherche aux utilisateurs d'UMDL. Il guide l'utilisateur en lui donnant des astuces pour chercher la meilleure information désirée ainsi que son meilleur format (texte, image, son, vidéo, etc.).

Agent d'exécution de requêtes : Le développement d'un paradigme de requête permet à l'utilisateur d'extraire l'information désirée facilement par la construction de requêtes complexes dans un environnement distribué. L'agent d'exécution de requêtes garde une trace des requêtes de l'utilisateur pour améliorer ses performances lors des prochaines requêtes. Cette trace lui permet de connaître les préférences de l'usager.

Médiateur : Le médiateur a pour but de contrôler les interactions entre les agents utilisateur et les agents collecteurs. Souvent les médiateurs forment une architecture hiérarchique. Les interactions entre les médiateurs ressemblent fortement à la résolution de problèmes coopératifs.

Agent collecteur : Les agents collecteurs sont reliés aux ressources. Ils sont en charge de la liaison entre les informations (base de données, collection d'image, vidéothèque, catalogue, etc.) et le système multiagent.

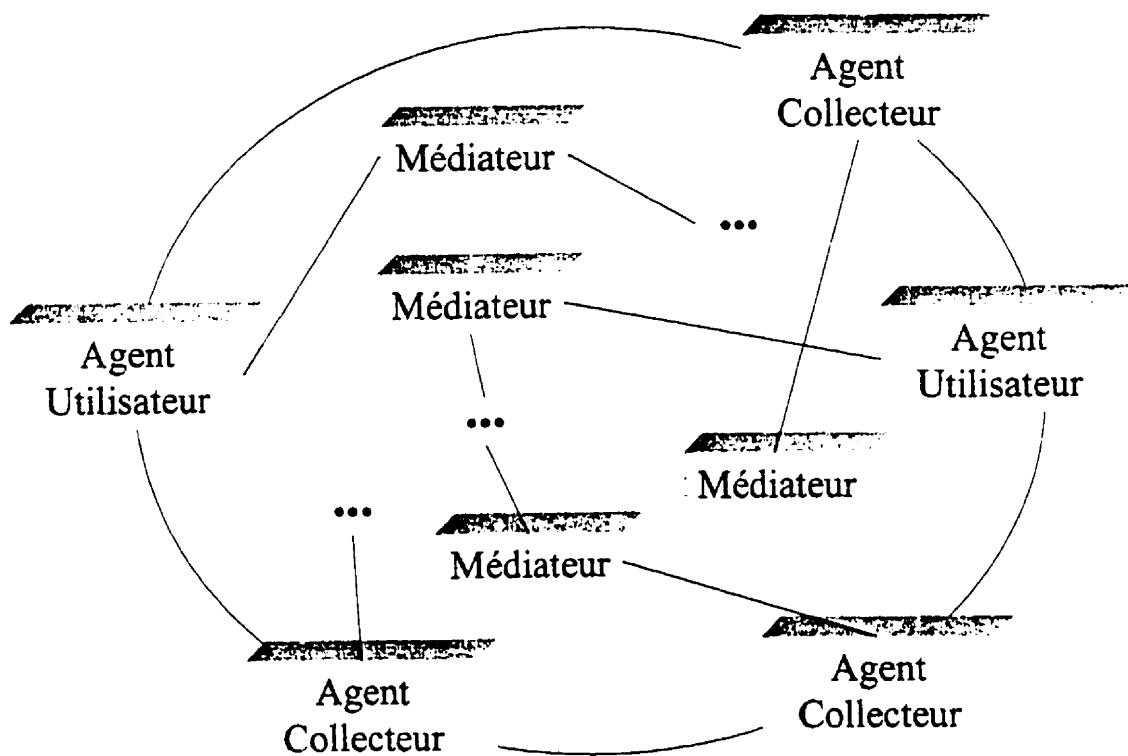


Figure 2.13: Architecture UMDL [VD95].

2.5.5 Conclusion

Ce chapitre a fait ressortir trois fonctionnalités pour les agents. Premièrement, il y a les agents utilisateur qui sont là pour l'assister et faire le lien avec le reste de l'architecture. Il y a aussi les agents de traitement qui se chargent de la planification et de la résolution de problèmes. Finalement, il y a les agents dédiés à la recherche d'informations.

Dans le cadre de ce mémoire, nous avons créé une architecture nommée NetSA qui englobe également ce principe à trois niveaux. Ce genre d'architecture s'avère fort commode dans la résolution de problèmes nécessitant l'accès à plusieurs sources de données hétérogènes. Le chapitre 4 donne plus de détails sur cette architecture.

Chapitre 3

Communication

Dans les systèmes multiagents, la communication joue un rôle prépondérant. Elle constitue le lien qui unifie les agents entre eux. Pour l'utiliser dans le cadre de notre architecture, nous avons utilisé tout d'abord, JATLite qui est un ensemble de classes Java pour la communication entre les agents. Ensuite, KQML qui est un langage de manipulation et de requête pour les connaissances dans un système multiagent.

3.1 JATLite

JATLite (Java Agent Template Lite [CDR99]) est un ensemble de classes écrites en Java permettant à des programmeurs de créer rapidement de nouveaux agents qui communiquent de manière robuste via l'Internet. JATLite fournit également une infrastructure de base, comme montré à la Figure 3.1, dans laquelle les agents s'enregistrent auprès de l'agent routeur de messages (AMR) en utilisant un nom et un mot de passe. Cette infrastructure permet aux agents enregistrés de se connecter et de se déconnecter d'Internet, de recevoir et d'envoyer de messages, de transférer des fichiers avec le protocole FTP et d'échanger d'une façon générale des informations avec d'autres agents situés dans des ordinateurs de tous genres.

Parmi les différents types d'agents vus au chapitre 2.5, JATLite est en fait spécialement utilisé pour le développement d'agents de type coopératif ou hybride et pour les systèmes multiagents comme décrit dans chapitre 2.5. De tels agents ont nécessairement besoin d'un lien communication fiable pour l'accomplissement de leur tâche. Ils doivent ultimement utiliser un protocole commun de messages, tel que KQML (voir la section 3.2), dans lequel la sémantique des messages est indépendante de l'application.

3.1.1 Utilité de JATLite

Les systèmes multiagents sont difficiles à construire et à déverminer et cela, d'une façon générale. De nos jours, leur programmation doit se faire avec Java afin qu'ils puissent fonctionner sur des plate-formes informatiques hétérogènes (Windows, Unix, Mac OS, etc.) ou pour qu'ils soient inclus dans de très petites applets pour un usage temporaire d'agent sur une page Web. Pour ce faire, JATLite fournit un ensemble de

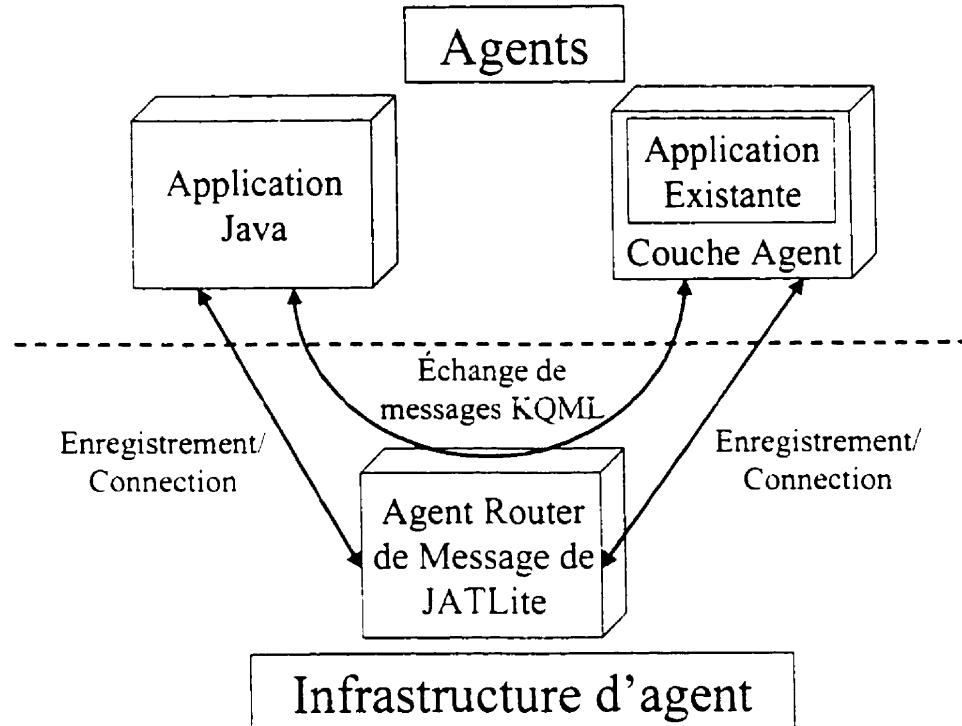


Figure 3.1: Agent router de message de JATLite.

classes et une infrastructure d'agent Java qui facilitent la construction de systèmes uniformes. De plus, il donne la possibilité aux programmeurs d'utiliser des langages de haut-niveau (par exemple KQML) et des protocoles communs (par exemple TCP, FTP et SMTP). Les classes fournies par JATLite sont destinées à une utilisation en couche, ce qui permet au développeur de décider quelles classes utiliser ou ignorer connaissant les couches qui lui seront utiles. Par exemple, si un programmeur ne veut pas utiliser KQML, il pourra dans ce cas omettre les classes de la couche KQML. Par contre, s'il utilise KQML, JATLite lui procure une couche KQML avec plusieurs outils dont un vérificateur de validité des messages KQML.

Ce qui rend JATLite vraiment unique, c'est son infrastructure. Traditionnellement, les systèmes multiagents utilisent un serveur de noms d'agents (ANS) pour faire la connection entre les agents. Un agent qui utilise un ANS doit demander à ce serveur l'adresse IP de l'agent avec qui il veut communiquer pour ensuite faire une connection TCP directe avec cet agent. Avec un tel ANS, si l'adresse IP des autres agents changent, la communication entre agents devient très difficile. Il faut donc incessamment demander et mettre à jour les adresses IP contenues dans l'ANS. Dans le même ordre d'idée, si un agent se déconnecte par accident il devient alors impossible de communiquer avec cet agent. C'est donc la responsabilité de chaque agent de vérifier que ses messages ont été bien transmis et de les retransmettre plus tard dans le cas d'un échec. Il est facile de voir que la programmation distribuée peut facilement être défaillante dans de telles conditions.

Avec l'infrastructure de JATLite, tous les agents sont en liaison directe avec l'agent

routeur de messages (AMR). L'AMR redirige tous les messages des agents selon leur destinataire à la dernière adresse IP connue. Ensuite, comme un système de messagerie électronique, l'AMR garde en mémoire et sauvegarde les messages reçus tant qu'il n'a pas reçu un accusé de réception demandant la suppression de ces messages. Pour avoir droit à ce service, un agent doit toujours initialiser la communication en envoyant une demande de connexion à AMR pour s'y enregistrer et préférablement terminer la communication par une demande de déconnexion. Ainsi l'AMR renforce la robustesse des systèmes distribués.

L'AMR de JATLite est une application spécialisée qui reçoit un message des agents enregistrés et qui les redirige au bon récepteur. Les messages reçus sont tous mis dans une file de messages dans le système de fichiers. L'utilisation de l'AMR apporte de nombreux avantages, que les ANS n'ont pas :

- les agents qui sont des applets peuvent communiquer avec d'autres agents malgré la sécurité imposée par les navigateurs Internet comme Netscape¹ et Internet Explorer².
- la communication asynchrone est possible même si un agent ne peut pas recevoir immédiatement un message qui lui est destiné.
- un agent n'a pas à se préoccuper des changements d'adresse possibles des autres agents ou des problèmes de transmission de messages.

Pour les agents mobiles qui changent très souvent d'adresse IP, JATLite est une solution très intéressante.

JATLite à lui seul ne constitue pas un agent. Autrement dit, JATLite seul ne donne aucun comportement intelligent à l'agent mais il procure cependant tout le nécessaire à la communication avec d'autres agents. Des aptitudes comme la recherche d'information et l'automatisation des tâches d'un utilisateur ne font pas partie des fonctionnalités de JATLite. Pour faire cela, le programmeur est libre d'utiliser la théorie ou la technique la mieux adaptée aux besoins de l'application visée. Un bon exemple est JESS. JESS est un système expert programmé en Java qui, associé à JATLite procure à l'agent une couche inductive pour des agents à base de règles.

3.1.2 Organisation de JATLite

L'architecture JATLite est organisée en une hiérarchie de plus en plus spécialisée d'une couche à l'autre (voir la Figure 3.2), afin que les programmeurs puissent sélectionner la couche appropriée leur permettant de construire leur propre système. Donc, un programmeur qui voudrait utiliser TCP/IP sans utiliser KQML peut utiliser seulement la «couche abstraite» et la «couche de base» (cf. Figure 3.2).

La «couche abstraite» prévoit la collection des classes abstraites nécessaire à la mise en oeuvre de JATLite. Bien que JATlite supporte toutes les connections pouvant être utilisées avec TCP/IP, nous pouvons rendre effectif des protocoles différents tel que UDP en étendant la «couche abstraite» de JATLite.

¹ Netscape Navigator est une marque déposée de Netscape Communications Corporation.

² Internet Explorer est une marque déposée de Microsoft Corporation.

La «couche de base» fournit la communication de base utilisant TCP/IP ainsi que la «couche abstraite». Il n'y a aucune restriction sur la langue du message ou sur le protocole. La «couche de base» peut être étendue, par exemple, pour autoriser des entrées via les sockets et des sorties vers un fichier. La «couche de base» peut également être étendue pour fournir aux agents des ports de transmission de messages multiples.

La «couche KQML» prévoit le stockage et la vérification syntaxique des messages KQML. Les extensions au niveau du standard de KQML, proposées par le *Center for Design Research* de l'Université Stanford (voir [KQM97]), sont rendues possibles dans la mesure où les messages peuvent être entièrement redéfinis par le programmeurs.

La «couche routeur» inscrit les noms des agents dans son répertoire et fait la redirection des messages. Tous les agents envoient et reçoivent des messages par le routeur qui les redirige vers leurs destinataires. Lorsqu'un agent se déconnecte intentionnellement, ou suite à un accident quelconque, le routeur entrepose les messages que ce dernier reçoit jusqu'à ce qu'il se reconnecte. Le routeur est particulièrement important pour les agents à base d'applets qui peuvent grâce à lui communiquer avec les autres agents sans être en conflit avec les restrictions de sécurité des applets imposées par les navigateurs.

Au dessus de la «couche routeur», la «couche de protocole» supportera l'internet standard et divers protocoles tel que SMTP, FTP, POP3, HTTP, etc. La version courante de JATLite supporte SMTP et FTP mais les autres protocoles peuvent être ajoutés à la couche de protocole facilement. Si les agents veulent transférer des données de manière non séquentielle ou si les agents ont besoin d'envoyer des messages KQML par courriel, la couche de protocole sera un bon point de départ.

3.1.3 Construction d'un agent avec JATLite

Nous allons voir dans cette section comment utiliser JATLite dans un agent. Une bonne structure d'agent doit être modulaire. Comme le montre la Figure 3.3. un agent possède deux modules principaux :

1. Le module de raisonnement est la partie intelligente de l'agent. C'est dans ce module que les réflexes, les raisonnements et la coopération entre agents sont implantés.
2. Le serveur de communication envoie, reçoit et traite les messages venant de l'agent routeur de messages (AMR). Il est également en charge de l'enregistrement, de la connexion et de la déconnexion de ce même AMR. C'est donc dans ce serveur que JATLite est utilisé.

Pour construire le serveur de communication, nous avons besoin de suivre les étapes suivantes :

Faire un constructeur pour AgentAction (par exemple : MyRouterClientAction) : Normalement, l'utilisation du constructeur de la super classe RouterClientAction est suffisante. Le constructeur par défaut RouterClientAction() initialise toutes les

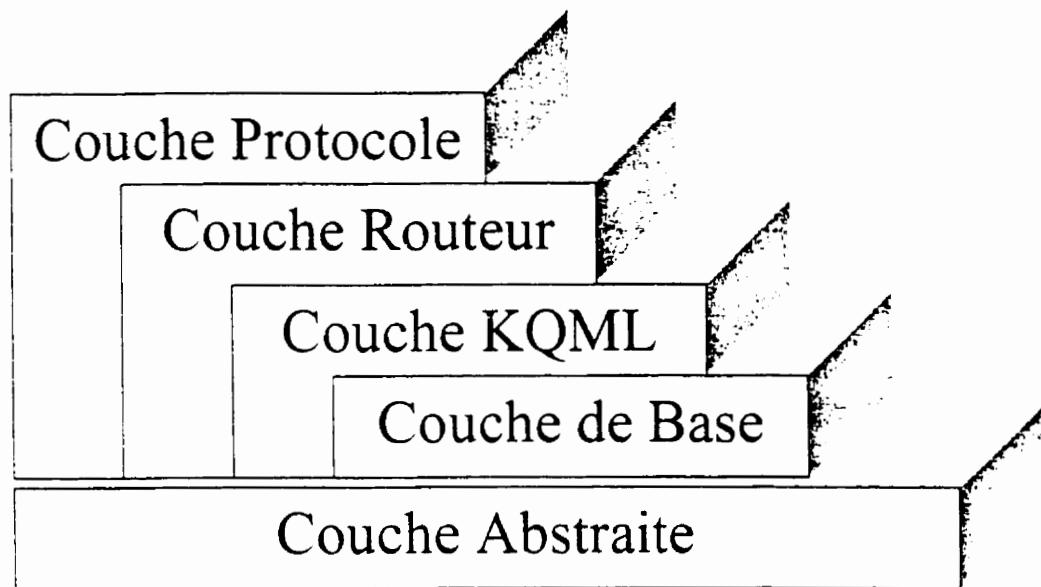


Figure 3.2: Architecture de JATLite

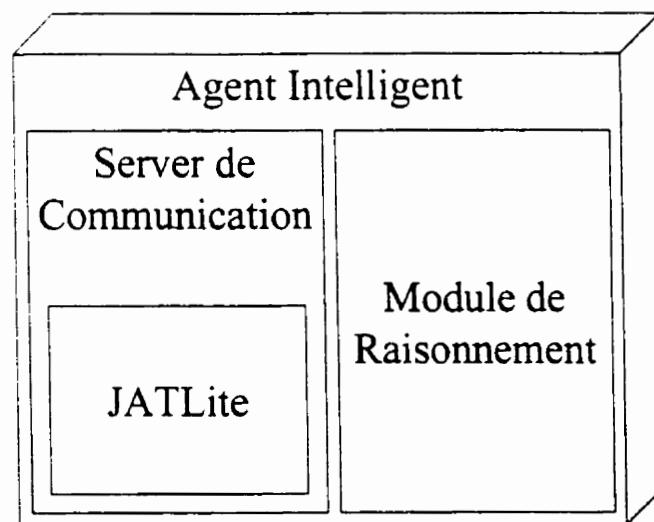


Figure 3.3: Exemple de structure d'agent modulaire.

données nécessaire à l'exception des adresses de l'agent, du routeur et du registrar³. Pour fournir ces adresses à l'objet RouterClientAction, il faut utiliser les méthodes setMyAddress(Address), setRouterAddress(Address) et setRegistrarAddress(Address). Le constructeur RouterClientAction(Address routerAddress, Address registrarAddress, Address myAddress, int maxIdleTime) permet également d'initialiser ces adresses immédiatement lors de l'appel du constructeur. La valeur maxIdleTime est le temps maximum d'opération en minute. Elle prend la valeur -1 pour un temps infini.

Remplir les méthodes Act(Object) et processMessage(String, Object) : Lorsqu'un message est reçu, la méthode Act(Object) est invoquée automatiquement. Par conséquent, vous devez définir le comportement que doit avoir votre agent lors de la réception d'un message dans cette méthode. Cependant lorsque le message a été traité, il ne faut pas oublier d'accuser sa réception auprès de l'AMR avec la méthode addToDeleteBuffer(int) afin de supprimer ce message de la file de messages qu'il conserve. Si le message n'est pas supprimé, il nous sera envoyé de nouveau plus tard. La méthode processMessage(String, Object) n'est pas une méthode essentielle mais elle peut être utilisée pour définir l'interaction avec les autres processus ou les GUI. Dans le cas où l'utilisateur ne veut pas rendre effective cette méthode, il lui revient alors de ne définir aucun code pour elle :

```
public void processMessage(String cmd, Object obj) {}
```

Créer l'objet RouterClientAction, l'enregistrer auprès de l'AMR (si nécessaire), et se connecter au routeur : Maintenant votre objet RouterClientAction est prêt. Faites une instance de votre RouterClientAction et appeler la méthode register() et/ou connect() de RouterClientAction. Par exemple :

```
MyRouterClientAction action =
new MyRouterClientAction(routerAddress, registrarAddress, myAddress, 100);
// Seulement si vous avez besoin de vous enregistrez
action.register();
action.connect();
```

Vous avez maintenant un serveur de communication qui utilise le routeur de JAT-Lite.

Envoyer un message : Pour envoyer un message vers un autre agent, vous devez utiliser les méthodes sendMessage(String receiver, String msg), sendMessage(String msg) ou sendKQMLMessage(String msg) de votre serveur de communication. Vous pouvez avec JATLite construire un message KQML en utilisant la classe KQMLmessage.

Déconnecter et annuler l'enregistrement à un AMR : Lorsque vous voulez arrêter l'agent, vous devez utiliser la méthode disconnect(). De son côté, la méthode unregister() annule l'enregistrement de votre agent à un AMR.

³Le registrar fait partie de l'AMR de JATLite et est utilisé pour valider les demandes de connexion.

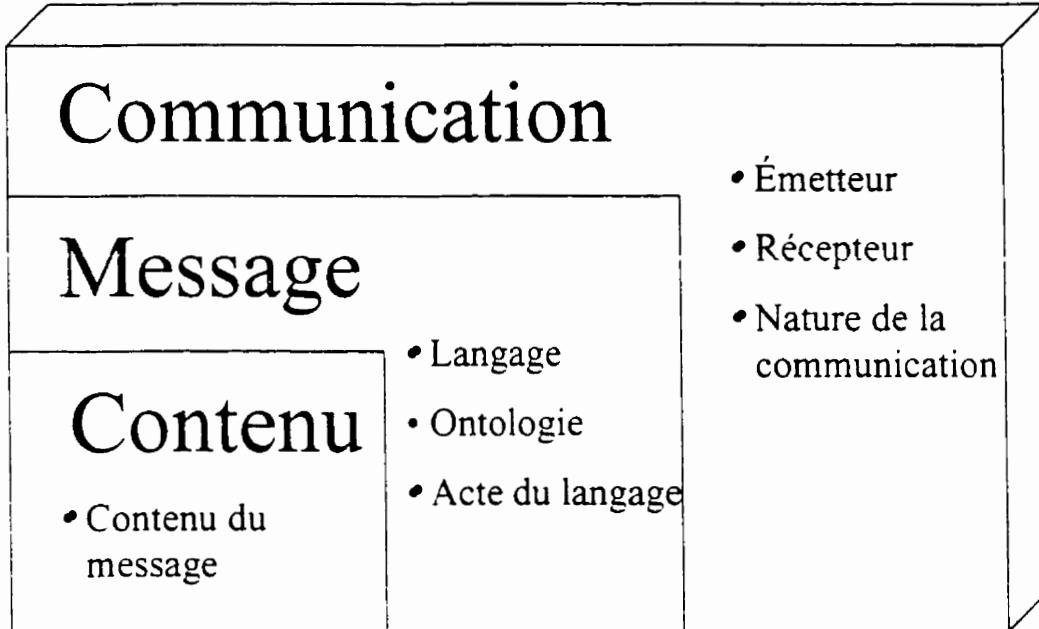


Figure 3.4: Couche de KQML.

3.2 KQML

KQML est un acronyme pour Knowledge Query and Manipulation Language[FFMM94]. C'est un langage de manipulation et de requête des connaissances utilisant les actes du langage. Les actes du langage sont des verbes utilisés par les agents pour échanger de l'information.

KQML est un langage à trois couches comme l'indique la Figure 3.4. La première couche est la couche dédiée au contenu et contient une expression écrite dans un langage permettant de structurer l'information à transmettre. La couche message est la deuxième couche de KQML. Elle ajoute à la première couche certaines propriétés comme la représentation de contenu, l'ontologie et le type d'acte de langage utilisé. La dernière couche, nommée couche de communication, insère à son tour, des éléments tels que l'émetteur, le récepteur et la nature de la communication.

3.2.1 Les paramètres réservés

Chacune des trois couches présentées précédemment ajoute un ou plusieurs paramètres à un message KQML. La liste complète de ces paramètres est résumée dans le Tableau 3.1. Dans ce paragraphe, nous allons étudier en détail chacun des paramètres.

Considérons tout d'abord l'exemple du message KQML suivant :

```
(ask-if
  :sender      A
  :receiver    B
  :language    Prolog
  :ontology    ÊtresHumains
  :reply-with  id1
  :content     ami(Nader,Marc) )
```

Dans cet exemple, *A* questionne *B*, en *Prolog* (valeur de :language), sur l'état de vérité de *ami(Nader,Marc)*, c'est-à-dire si cette expression existe dans la base de connaissances de *B*. Toute réponse à ce message KQML (acte illocutoire) sera identifiée par *id1* (la valeur de :reply-with). L'ontologie de nom *ÊtresHumains* peut apporter des informations supplémentaires concernant l'interprétation de :content.

Mot-clé	Signification
:content	contenu de la performatif
:sender	émetteur actuel de la performatif
:receiver	récepteur actuel de la performatif
:from	émetteur intermédiaire
:to	destinataire intermédiaire
:reply-with	référence de la réponse éventuelle
:in-reply-to	référence attendue lors d'une réponse
:language	langage de représentation du paramètre :content
:ontologie	l'ontologie supposée du paramètre :content

Tableau 3.1: Liste des paramètres réservés des performatives en KQML.

Dans la terminologie KQML, *ask-if* est une performatif (voir la section suivante). Les paramètres des performatives sont indexés par des mots-clés et sont dès lors indépendants de l'ordre. Ces mots-clés, appelés *noms de paramètres* doivent débuter par le signe «deux points» (:) et doivent précéder la *valeur du paramètre*. Les paramètres ont été identifiés par des mots-clés plutôt que par leur position, à cause du grand nombre de paramètres optionnels. Il y a 9 paramètres dits *réservés* car il faut les utiliser selon les définitions suivantes.

:content <expression>. Ce paramètre renferme l'information sur laquelle la performatif exprime une attitude. La valeur de :content est une expression dans un certain langage informatique interprété ou un autre message KQML, et représente le contenu de l'acte de communication (illocutoire). Les constantes utilisées dans <expression> doivent avoir été définies dans le paramètre :ontology.

Les autres paramètres contiennent les valeurs qui créent le contexte pour l'interprétation de :content et en même temps contiennent l'information pour faciliter le

traitement du message.

`:sender <word>, :receiver <word>`. Ces deux paramètres indiquent l'émetteur et le destinataire réels d'une performative.

`:from <word>, :to <word>`. Ces deux paramètres indiquent l'émetteur et le destinataire du message. Ces paramètres sont présents dans la performative **forward** (renvoi).

`:reply-with <word>`. L'émetteur d'un message indique que la réponse éventuelle à son message doit se faire avec cette référence. Cette dernière est précisée par le paramètre `:reply-with`.

`:in-reply-to <word>`. Le message qui sert de réponse à un certain message précédent reprend la référence correspondante, qu'il indique dans son paramètre `:in-reply-to`.

`:language <word>`. Ce paramètre indique le langage de représentation utilisé dans `:content`.

`:ontology <word>`. Ce paramètre indique l'ontologie (l'ensemble de définitions de termes) supposée pour `:content`.

3.2.2 Les 35 performatives du discours

Les performatives se classent en trois catégories : les performatives de discours, les performatives de régulation de conversation et celles d'assistance et de réseau.

Les 7 performatives de régulation de conversation : Leur rôle est d'intervenir dans le cours normal d'une conversation. Le cours normal d'une conversation est le suivant : l'agent *A* envoie à *B* un message KQML (débutant ainsi une conversation) et l'agent *B* lui répond s'il a une réponse à donner. Ainsi ces performatives peuvent soit terminer prématurément une conversation (par `error` ou `sorry`), soit outrepasser ce protocole par défaut (par `standby`, `next`, `rest` ou `discard`). La liste complète des performatives de régulation de conversation est donnée par le Tableau⁴3.2.

L'exemple suivant illustre une utilisation de la performative `sorry`. Dans cet exemple, l'agent *Nader* informe l'agent *Marc* que le fichier *F1* ne peut être effacé. L'agent *Nader* utilise le *Français* comme langage de description du contenu et précise que l'ontologie utilisée est celle de *GestionFichier*.

⁴Dans ce tableau, on désigne l'émetteur (sender) par *E*, le récepteur (Receiver) par *R* et la base de connaissances par *BC*.

<i>Performative</i>	<i>Signification</i>
error	<i>E</i> considère que le message précédent de <i>R</i> est mal formaté
sorry	<i>E</i> comprend le message de <i>R</i> mais ne peut donner de meilleure réponse
standby	<i>E</i> veut que <i>R</i> lui indique lorsqu'il sera prêt à répondre au message
ready	<i>E</i> est prêt répondre un message reçu précédemment de <i>R</i>
next	<i>E</i> veut la prochaine réponse à un message envoyé précédemment <i>R</i>
rest	<i>E</i> veut toutes les réponses restantes à un message envoyé précédemment <i>R</i>
discard	<i>E</i> n'a plus besoin des réponses restantes à un message envoyé précédemment <i>R</i>

Tableau 3.2: Les 7 performatives de régulation de conversation (*E* et *R* désignent respectivement l'émetteur et le récepteur).

```
(sorry
  :sender      Nader
  :receiver    Marc
  :language    Français
  :ontologie   GestionFichier
  :content     NePeutEffacerFichierF1)
```

Les 17 performatives de discours : C'est la catégorie qui s'apparente le plus aux actes de discours dans le sens linguistique. Bien sûr l'idée de donner un format explicite aux réponses (comme dans **stream-all** ou **ask-one**) est inhabituelle dans la perspective de la théorie des actes du langage, mais ces réponses représentent néanmoins encore des actes du langage au sens strict du terme. Elles s'utilisent dans un contexte de discours entre 2 agents s'échangeant des informations et des connaissances. La liste complète des performatives de discours est donnée par le Tableau 3.3.

L'exemple suivant illustre une utilisation de la performatif **tell**. Dans cet exemple, l'agent *Nader* informe l'agent *Marc* que le fichier *F1* est effacé. L'agent *Nader* utilise le *Français* comme langage de description du contenu et précise que l'ontologie utilisée est celle de *GestionFichier*.

```
( tell
  :sender      Nader
  :receiver    Marc
  :language    Français
  :ontologie   GestionFichier
  :content     FichierF1Effacé)
```

Les 11 performatives d'assistance et de réseau. Ces performatives ne sont pas des actes de langage dans le vrai sens du terme. Elles sont avant tout des performatives qui permettent à des agents de trouver d'autres agents qui puissent traiter leurs demandes. Ces performatives nécessitent le passage en quelque sorte

<i>Performative</i>	<i>Signification</i>
ask-if	E veut savoir si :content est dans la BC de R
ask-all	E veut connaître toutes les instances possible de :content dans la BC de R
ask-one	E veut connaître une instance possible de :content dans la BC de R
stream-all	version multi-réponse de ask-all
eos	marqueur de fin d'une multi-réponse
tell	l'expression dans :content est dans la base de connaissances de E
untell	l'expression dans :content n'est pas dans la BC de E
deny	la négation de l'expression est dans la BC de E
insert	E demande à R d'ajouter le :content dans sa BC
uninsert	E demande à R d'annuler l' insert précédent
delete-one	E veut que R retire une instance de :content de sa BC
delete-all	E veut que R retire toutes les instances de :content de sa BC
undelete	E demande à R d'annuler le delete précédent
achieve	E veut que R fasse une action pour rendre :content vrai
unachieve	E demande à R d'annuler le achieve précédent
advertise	E informe R qu'il sait traiter un message comme celui dans :content
subscribe	E veut être informé de tout changement futur de la réponse la performative contenu dans :content

Tableau 3.3: Les 17 performatives de discours (E et R désignent respectivement l'émetteur et le récepteur).

par des agents intermédiaires, appelés ici agents assistants. La liste complète des performatives d'assistance et de réseau est donnée par le tableau 3.4.

<i>Performative</i>	<i>Signification</i>
<code>register</code>	E annonce à R sa présence et son nom symbolique
<code>unregister</code>	E demande à R d'annuler le <code>register</code> précédent
<code>forward</code>	E veut que R renvoie la performative à l'agent indiqué par :to
<code>broadcast</code>	E veut que R envoie un message à tous les agents connus par lui
<code>transport-address</code>	E associe son nom symbolique à une nouvelle adresse de réseau
<code>broker-one</code>	E veut que R trouve une réponse (via un autre agent) à une performative donnée
<code>broker-all</code>	E veut que R trouve toutes les réponses (via d'autres agents) à une performative donnée
<code>recommand-one</code>	E veut se faire recommander par R un agent qui pourra répondre une performative donnée
<code>recommand-all</code>	E veut se faire recommander par R tous les agents qui pourront répondre une performative donnée
<code>recruit-one</code>	E veut que se fasse connaître à lui directement un agent qui saura répondre à une performative donnée
<code>recruit-all</code>	E veut que se fassent connaître lui à directement tous les agents qui sauront répondre à une performative donnée

Tableau 3.4: Les 11 performatives d'assistance et de réseau (E et R désignent respectivement l'émetteur et le récepteur).

L'exemple suivant illustre une utilisation de la performative `forward`. Dans cet exemple, l'agent *Nader* demande à l'agent *Marc* de transmettre le message à l'agent *Foxtrott*. Ce message dit que le fichier *F1* est effacé. L'agent *Ned* utilise le *Français* comme langage de description du contenu et précise que l'ontologie utilisée est celle de *GestionFichier*.

```
(forward
  :sender      Nader
  :receiver    Marc
  :to          Foxtrott
  :language    Français
  :ontologie   GestionFichier
  :content     FichierF1Effacé)
```

KQML est donc une collection de primitives de communication. Il n'existe pas d'implantation de KQML, dans le sens que ce n'est pas un langage interprété ou compilé tournant sur une plate-forme matérielle ou sur une machine abstraite. Les agents «communiquent en KQML» s'ils font usage des primitives de KQML, primitives dont nous avons donné un aperçu dans cette section. Il faut préciser que bien que la bibliothèque des primitives de KQML ne soit pas encore stable et ne constitue pas, pour

le moment, un standard, il est fort utile d'encourager les chercheurs à définir un ensemble de performatives qui soit cohérent et dont la sémantique soit claire. Il faudra en particulier, identifier de manière beaucoup plus précise les champs qui doivent être retenus dans chaque performative.

3.2.3 Utilisation de KQML dans la communication pour la recherche d'informations

Il existe plusieurs façons d'utiliser KQML dans la recherche d'information (cf. Figure 3.5). La plus simple est lorsque les agents connaissent où sont situés les informations (routage d'adresse). Dans ce cas, une communication directe entre l'agent demandeur et l'agent informateur suffit. Cependant, lorsque l'information est très dynamique et volatile, ce mode de communication est défaillant. Il est extrêmement ardu de maintenir à jour une liste exhaustive de la localisation de l'information.

Une manière un peu plus intelligente consiste à utiliser une entité intermédiaire qui nous permet de tenir à jour soit la localisation de l'information, soit l'information elle-même. Dans le cas d'un routage basé sur le contenu, un agent s'inscrit au niveau de l'agent intermédiaire pour lui demander de le tenir informer de tous changements au sujet d'une information précise (par exemple la météo). Lorsqu'un agent informe l'agent intermédiaire d'un changement, celui-ci informe à son tour tous les agents ayant fait la demande du service de mise à jour.

Un agent intermédiaire peut également servir de courtier. Souvent les agents ne sont pas très intéressés de savoir d'où vient l'information. Dans ce cas, un agent peut demander à un agent courtier de lui fournir l'information qu'il désire. Ce dernier connaît les agents fournissant des services dans le système multiagent et engage un processus de négociation pour obtenir l'information demandée par l'agent demandeur. Lorsqu'il a obtenu une réponse satisfaisante, le courtier la transmet à l'agent demandeur sans que celui-ci en connaisse la provenance.

Un autre moyen de router l'information est d'utiliser un recrutement. Le recrutement se fait via un agent recruteur qui connaît l'emplacement de l'information dans le système multiagent. Comme pour le courtier, l'agent demandeur demande à l'agent recruteur une information. De la même manière, ce dernier engage une négociation pour trouver l'agent qui répond le mieux aux critères de l'agent demandeur. Lorsque cet agent a été trouvé, il répond directement à l'agent demandeur. Ce processus évite l'utilisation d'un intermédiaire et permet à l'agent demandeur de connaître la provenance de l'information.

La dernière manière de router l'information consiste pour l'agent intermédiaire de jouer le rôle des pages jaunes. Comme nous les connaissons, nous retrouvons dans le pages jaunes les services disponibles et ceux qui les fournissent. Donc lorsqu'un agent consulte l'agent intermédiaire jouant le rôle de pages jaunes, il en retire le nom de l'agent qui répond le mieux à ces exigences. Ensuite, il communique directement avec l'agent qui possède l'information ou le service désiré.

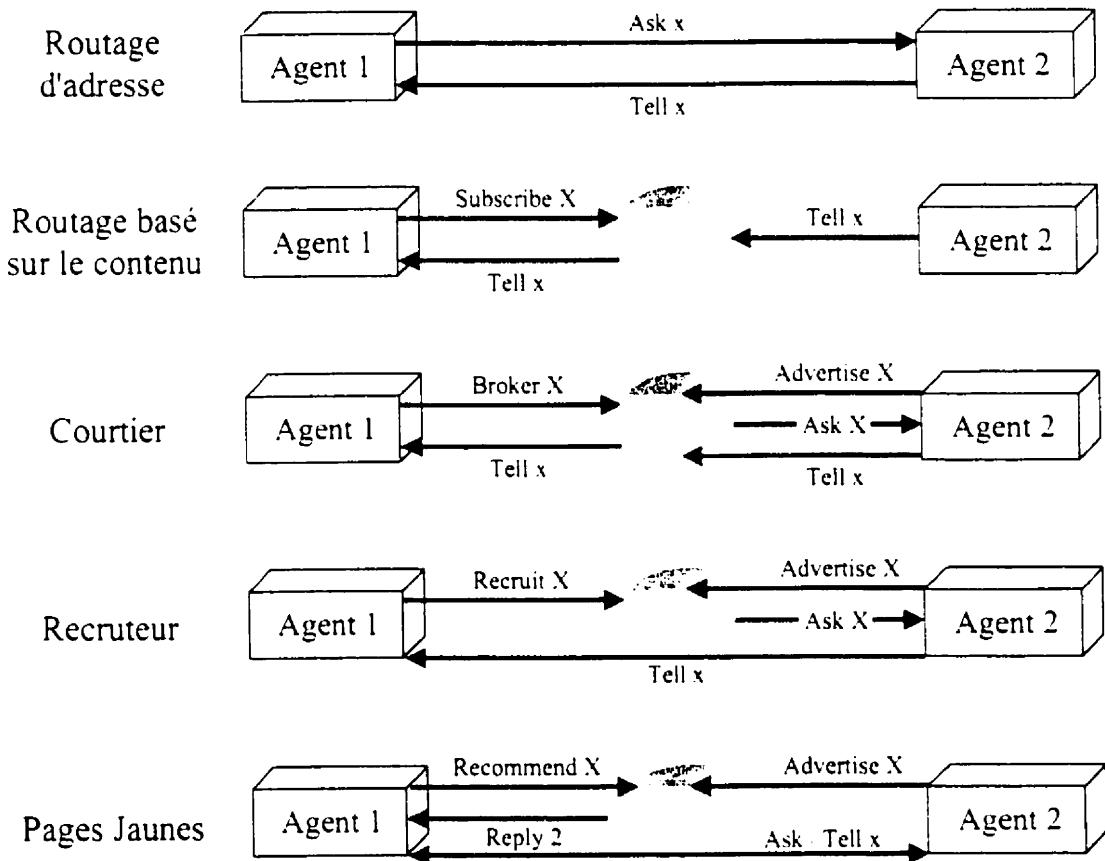


Figure 3.5: Modes de routage de l'information.

3.3 Conclusion

Dans ce chapitre, nous avons vu que JATLite est un outil de communication qui facilite la construction d'agent en fournissant une infrastructure de communication robuste et efficace. Cette infrastructure nous fournit plusieurs outils dont un agent routeur de messages et un serveur de nom d'agent. JATLite est très modulaire et divisé en 5 couches qui peuvent être utilisé en totalité ou en partie selon les besoins du programmeur.

Nous avons également présenté en détails le langage KQML, un langage de manipulation et de requête des connaissances.

Finalelement, nous avons présenté les cinq modes de routage de l'information utilisant KQML : le routage d'adresse, le routage basé sur le contenu, l'utilisation d'un courtier, l'utilisation d'un recruteur et la consultation de pages jaunes.

Chapitre 4

Architecture NetSA

Dans ce chapitre, nous présentons un exemple d'architecture à trois niveaux appelée NetSA (pour Networked Software Agent) qui a été conçue par M. Brahim Chaib-draa, Nader Troudi et moi-même (Marc Côté) dans le groupe DAMAS (Data-mining, Agent et Multiagents) à l'Université Laval. Cette architecture sera utilisée plus tard (voir chapitre 7) pour développer un prototype de système multiagent dédié au courtage d'hypothèques entre plusieurs banques utilisant les principes de vente aux enchères.

L'architecture NetSA est une architecture flexible offrant les outils nécessaires pour développer des architectures multiagents, particulièrement dans les domaines riches en informations. Notre propre contribution à cette architecture a porté sur la spécification des agents (agent utilisateur, agent superviseur, agent intermédiaire et agent ressource), ainsi que le développement des agents utilisateur et superviseur, agents que nous détaillerons plus tard. Pour bien comprendre le fonctionnement et les détails de l'architecture NetSA, nous allons maintenant expliquer plus en détails cette architecture ainsi que les agents qui la composent.

4.1 Caractéristiques de NetSA

Premièrement, NetSA est une architecture multiagent conçue particulièrement pour oeuvrer dans les environnements riches en informations. De nombreuses caractéristiques sont incluses dans cette architecture, caractéristiques que nous allons maintenant détailler.

4.1.1 Réutilisabilité et portabilité

NetSA se veut une architecture réutilisable. Par réutilisable, nous voulons signifier que les agents contenus dans l'architecture peuvent être utilisés à différentes fins, moyennant seulement une reconfiguration de chacun. Nous pouvons facilement utiliser l'architecture pour la recherche sur Internet, pour les finances ou pour la santé moyennant des changements quant au contenu des agents.

NetSA est également portable, c'est-à-dire qu'elle peut être exécutée sur plusieurs systèmes d'exploitation. La source de sa portabilité vient du fait qu'elle a été program-

mée à l'aide du langage Java de *Sun Microsystem*. Ce langage est exécutable sur la majorité des systèmes d'exploitation disponibles et ne nécessite aucune recompilation lors du transfert d'un environnement vers un autre. L'utilisateur peut donc exécuter NetSA sur une station Sun aussi bien que sur un PC ou un ordinateur Macintosh.

NetSA est aussi très accessible aux utilisateurs. Le moyen de communication entre les utilisateurs et NetSA est l'Internet par l'entremise du Web. La diffusion rapide sur Internet et son accessibilité font que NetSA peut être consultée à partir de n'importe quel point dans le monde. Pour cela, l'utilisateur doit posséder évidemment un ordinateur, un accès à Internet et un fureteur supportant le protocole HTTP pour pouvoir naviguer sur le Web. Les versions 2.0 et les versions ultérieures de Netscape et Internet Explorer sont donc compatibles avec NetSA.

La portabilité et l'universalité étant des facteurs prédominants dans la conception de notre système multiagent, l'utilisation de Java a été un choix quasi évident. Java est un langage du type compilé une fois, et exécuté sur toutes sortes de machines. De plus, Java est un langage robuste qui surveille ces accès mémoire continuellement.

4.1.2 Communication entre agents basée sur JATLite

JATLite (Dont les détails sont donnés au chapitre 3) a également été utilisé comme couche de communication. Il fournit toutes les fonctionnalités nécessaires à la communication entre les agents en plus de fournir un vérificateur de la syntaxe de KQML.

Rappelons que KQML (voir chapitre 3.2) est un langage d'interrogation et de manipulation des connaissances. C'est un langage basé sur les actes du langage naturel. Il est surtout utilisé pour la communication entre agents et les bases de données coopératives. Ce langage n'est pas encore normalisé mais étant donné sa forte utilisation, on peut dire qu'il est devenu une norme ad hoc.

Le contenu d'un message dans NetSA est divisé en deux parties. La première partie ([in]) est constituée de l'information utile pour faire la requête. La seconde partie ([out]) informe l'agent qui reçoit le message de ce qu'il doit retourner comme information ou ce qu'il doit faire comme travail. La différence entre information et travail est réalisée par le protocole de communication de chaque agent. Dans l'exemple ci-dessous, l'agent superviseur demande à l'agent utilisateur d'afficher une page Web contenant le nom et le prénom d'une personne en utilisant le fichier gabarit «*results.html*». Ici, un travail a été demandé, car le protocole de l'agent utilisateur lui dicte qu'un acte de langage «*reply*» que le message reçu comporte des données qui ont été demandées par l'utilisateur et qu'il doit en informer ce dernier.

```
(reply
  :sender    Supervisor
  :receiver   UserAgent
  :replyWith  Supervisor172539363539546
  :inReplyTo   UserAgent172539363533819
  :context     mortgage
  :content     /in/ name = Luc Côté; /out/ pageID = result.html;)
```

4.1.3 HTML, formulaire et JavaScript

Pour faire des formulaires, nous avons pensé utiliser des applets. Il réside cependant une inconsistance au niveau des contrôles graphiques entre les différentes plate-formes. Par exemple la qualité graphique des stations Sparc de Sun Microsystem provoque un rétrécissement des champs textes programmés sous Windows. De plus, les niveaux de sécurité implantés sur les différents navigateurs Internet nous compliquent grandement la tâche. Nous avons donc opté pour des pages HTML contenant des formulaires et des JavaScripts [Net97]. Ces pages sont expédiées et dirigées par un Servlet¹ [Mic98] selon les besoins de l'utilisateur.

Le formulaire est un moyen consistant pour recueillir des données à partir d'un navigateur Internet. Ceci repose sur le fait qu'un navigateur programmé pour un système d'exploitation utilise les contrôles de saisie de ce système et non ceux reprogrammés par Java. Nous sommes donc toujours certains d'obtenir un affichage impeccable peu importe le système d'exploitation.

Des JavaScripts sont utilisés dans les formulaires pour vérifier les données saisies par l'utilisateur. En les utilisant, nous pouvons vérifier les champs obligatoires et les champs critiques. Ils nous permettent en fait de faire un premier filtrage ou validation de l'information avant de l'envoyer vers l'agent utilisateur.

4.1.4 Les agents composants NetSA

L'architecture de NetSA peut être qualifiée d'architecture arborescente à trois niveaux d'abstraction. Chaque niveau constitue une unité abstraite qui occupe une tâche bien précise. Comme le montre la Figure 4.1, nous retrouvons les unités suivantes :

1. l'unité de communication avec l'utilisateur,
2. l'unité de traitement de l'information,
3. l'unité d'interrogation et d'extraction des données.

Dans NetSA, une requête est d'abord reçue par l'unité de communication. Cette unité communique alors avec l'unité de traitement de l'information qui décompose la requête en sous-requêtes (ou sous-plans) et les exécute jusqu'à ce qu'elle tombe sur un sous-plan nécessitant une information complémentaire. Dans ce cas, l'information manquante est fournie par un des agents de la troisième couche. À noter que la décomposition faite par l'unité de traitement est une décomposition qui s'arrête aux sous-plans pouvant donner lieu à des compléments d'information qu'on peut trouver auprès des agents de la troisième couche. Dans le cas où plusieurs réponses à des sous-plans sont demandées aux agents de la troisième couche, une synthèse est effectuée de manière à élaborer une réponse à l'utilisateur. Une fois le plan global traité, une réponse à la requête initiale est formulée et retournée à la première couche.

Ainsi nous passons donc d'une requête venant de l'utilisateur avec un haut niveau d'abstraction vers une décomposition de manière à aboutir à des requêtes de bas niveau

¹Un Servlet est un programme Java capable de traiter les données reçues via Internet comme le ferait un programme Perl pour la gestion de formulaire.

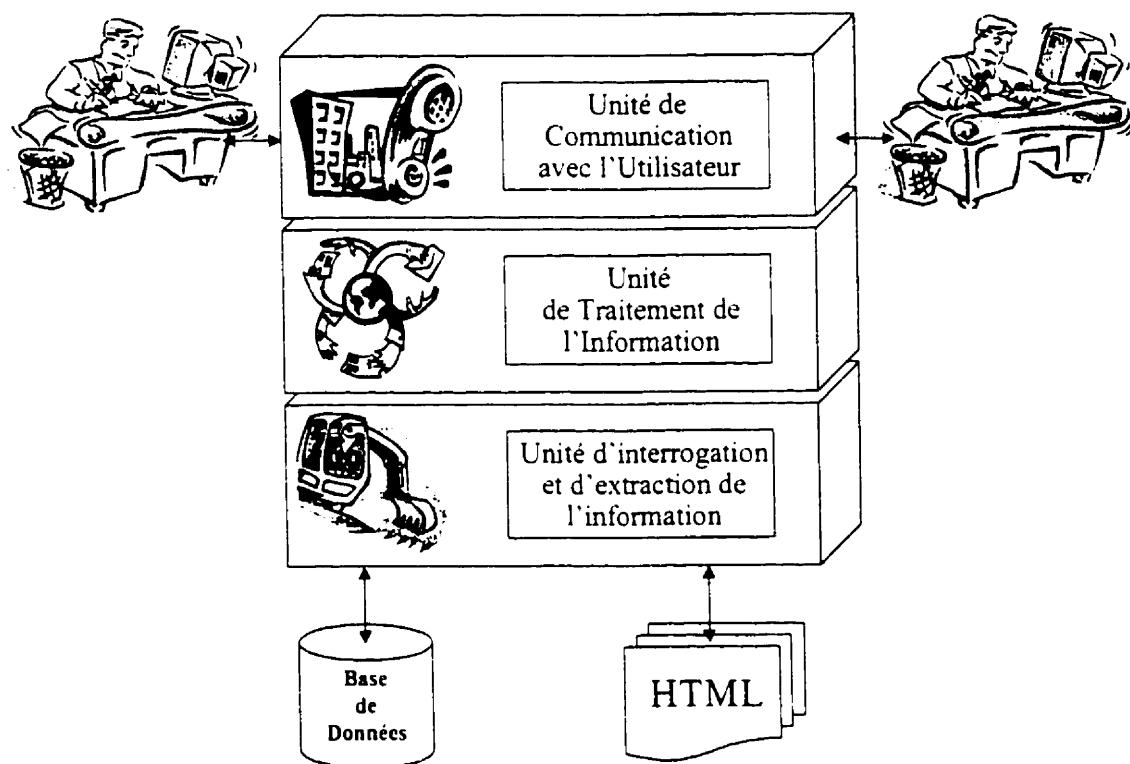


Figure 4.1: Les couches abstraites de NetSA.

d'interrogation pour les bases de données. Nous remontons ensuite la hiérarchie pour construire la réponse à la requête jusqu'à obtenir une réponse exprimée à un niveau d'abstraction suffisant pour la compréhension de l'utilisateur. En résumé, les trois unités de NetSA peuvent être vues de la manière suivante :

Unité de communication avec l'utilisateur : Cette unité est chargée des communications entre NetSA et l'usager. Elle comprend des agents interagissant avec l'utilisateur pour l'aider à réaliser une tâche bien précise. Cette interaction se traduit par une transformation des requêtes de l'usager qui, transformées en des actes du langage KQML, facilitent la communication avec les agents de l'unité de traitement. L'unité vérifie également la consistance des données fournies par l'utilisateur.

Unité de traitement d'informations : Cette unité reçoit de l'unité de communication les requêtes à satisfaire ainsi que les informations fournies par l'utilisateur. Elle décompose ces requêtes en sous-plans. Un sous-plan est une succession d'actions à exécuter dans le but d'atteindre un objectif intermédiaire et une série de sous-plans forme un plan global. L'unité comporte également une section pour rechercher des données dans le système multiagent. Tel un patrouilleur, elle dirige les agents vers la ressource désirée en fournissant le nom de l'agent en charge de cette ressource.

Unité d'interrogation et d'extraction d'informations : L'unité d'interrogation et d'extraction d'information est une interface entre les bases de données et l'unité de traitement d'information. Elle transforme les requêtes KQML reçues et les traduit en requêtes SQL pour interroger des bases de données. De ces bases de données, elle retire l'information pertinente et la redirige vers l'unité de traitement de l'information sous un format KQML. Elle peut également retirer l'information contenue dans une page HTML de l'Internet.

4.1.5 Propriétés des agents composant NetSA

L'architecture multiagent de NetSA comporte différents types d'agent dans des concentrations variables. Dans la structure arborescente de NetSA, nous pouvons facilement distinguer les trois couches d'abstraction et leur composition comme le montre la Figure 4.2. C'est ainsi que nous trouvons :

- au moins un agent utilisateur dans l'unité de communication avec l'utilisateur,
- au moins un agent superviseur dans l'unité de traitement de l'information,
- au moins un agent intermédiaire dans l'unité de traitement de l'information,
- plusieurs agents ressources dans l'unité d'interrogation et d'extraction de l'information.

Les agents de NetSA possèdent plusieurs propriétés qui sont conformes aux propriétés générales des agents. Ces propriétés sont :

- *Portabilité* : Pour assurer sa portabilité, l'agent est programmé avec le langage Java.

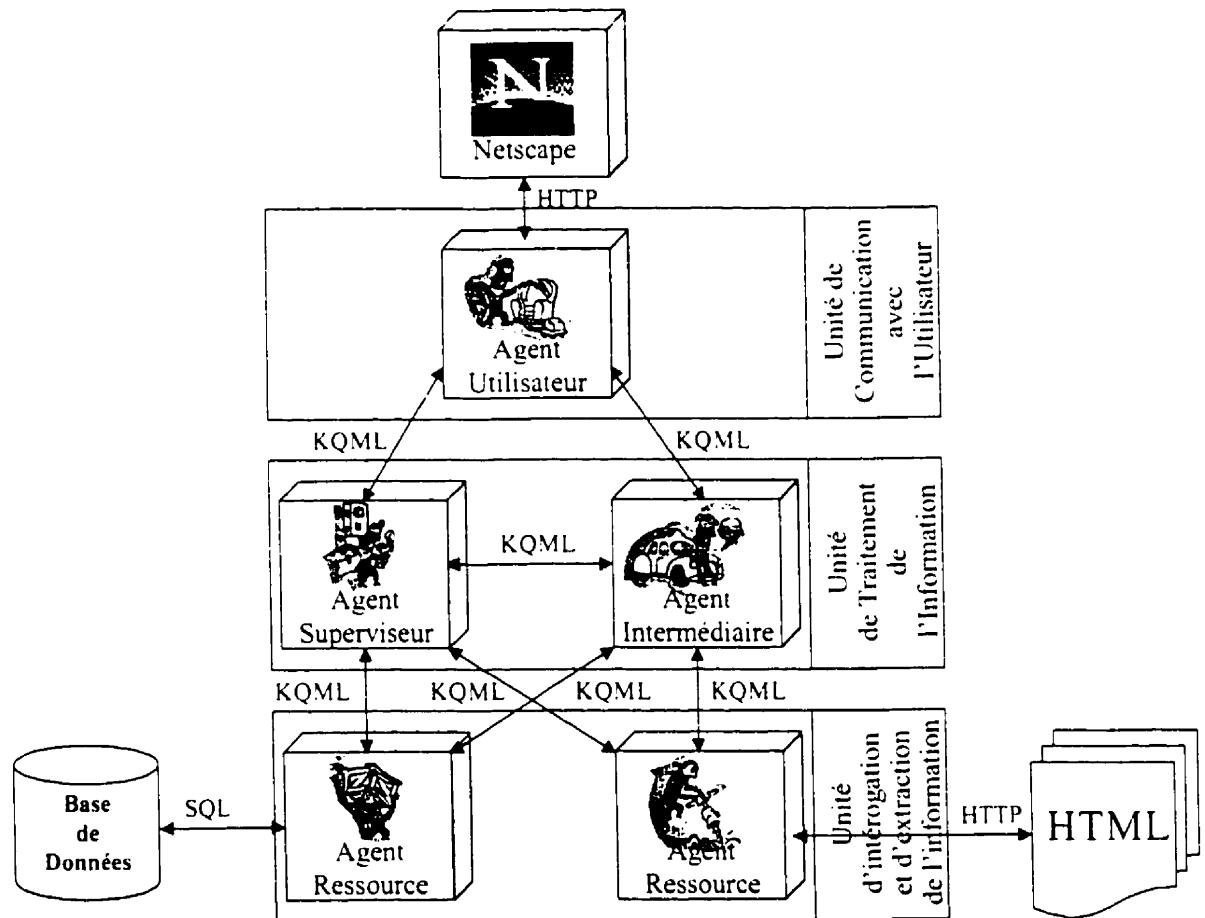


Figure 4.2: Architecture de NetSA

- *Autonomie* : L'agent fonctionne sans l'intervention de l'utilisateur.
- *Stabilité* : Une bonne gestion des exceptions permet à l'agent de demeurer dans un état stable.
- *Persistance* : L'agent reprend le cours normal de ses opérations après une panne du système informatique.
- *Flexibilité* : Les agents de NetSA doivent être en mesure de pouvoir être reconfigurés afin qu'ils puissent être facilement utilisés pour rechercher des informations dans d'autres domaines d'applications.

4.2 Agent Utilisateur dans NetSA

L'agent utilisateur est la porte d'entrée des requêtes externes au système. Il fournit à l'utilisateur le bon formulaire HTML lui permettant de faire une requête. L'agent traduit ensuite le formulaire soumis dans le protocole KQML en vue de son utilisation dans le système multiagent. Chaque agent est capable de s'occuper d'un ou plusieurs utilisateurs au même moment.

4.2.1 Description

L'agent utilisateur s'occupe de transmettre à l'utilisateur les pages Web requises pour la cueillette et l'affichage des informations pour le domaine auquel il a été destiné (voir Figure 4.3). Dans le présent travail, un système d'analyse dynamique (Servlet) a été utilisé pour interagir avec l'utilisateur.

L'utilisateur choisit d'abord le contexte dans lequel il veut travailler. L'agent lui donne alors la première page de cueillette d'information. Selon les réponses de l'utilisateur, l'agent déterminera la page suivante.

La gestion des pages Web pour la communication avec l'utilisateur est faite à l'aide d'un Servlet. Chaque page Web utilisera des formulaires et des JavaScript pour recueillir l'information. Ainsi nous pouvons gérer la cohérence des contrôles graphiques pour chaque systèmes d'exploitation. Chaque contrôle graphique portera un nom unique qui correspond à la variable dont le contrôle capte l'information. L'ordre dans lequel l'agent doit afficher les pages Web est défini dans l'attribut caché «nextID» qui prend la forme suivante : <input type="hidden" name="nextID" value="hypo2.html">.

4.2.2 Architecture interne

L'architecture interne de l'agent utilisateur est composée de 4 modules principaux, comme l'indique la Figure 4.4. Ces modules donnent à l'agent une plus grande extensibilité étant donné leur découpage selon les tâches.

Le module de communication utilisateur est un Servlet qui communique avec le module de traitement via des RMIs (Remote Method Invocation). Il reçoit les données de la page HTML et les transfert au module de traitement. L'opération inverse est

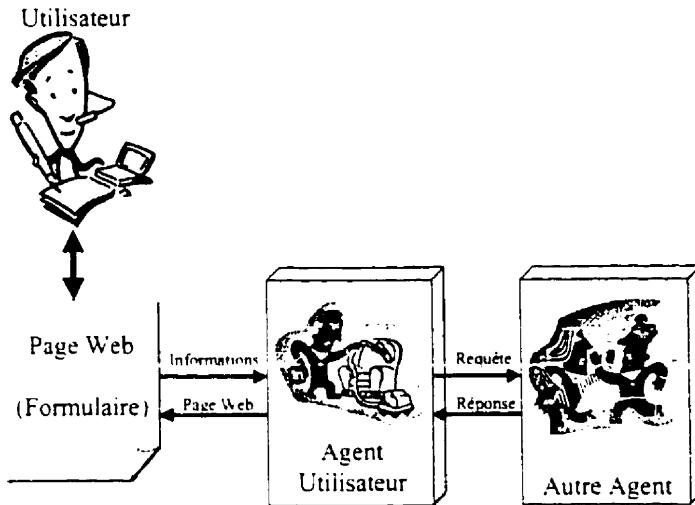


Figure 4.3: Description de l'agent utilisateur.

également disponible, c'est à dire que le module peut recevoir de l'information du module de traitement pour la montrer à l'utilisateur via une page Web.

Le module de communication inter-agents reçoit, du module de traitement, des demandes de transmission de messages KQML vers les autres agents. Il transfert également les informations reçues des agents du système multiagent au module de traitement.

Le module de traitement reçoit des données du module de communication utilisateur et les sauve dans le registre de l'agent. Ce registre contient toutes les informations recueillies par l'agent sur les utilisateurs du système multiagent. Il détermine ensuite si toute les informations nécessaires à la formation de la requête sont disponibles. Dans l'affirmative, il construit un message KQML et demande au module de communication inter-agents de le transmettre. Dans la négative, il demande les informations complémentaires à l'utilisateur via une page HTML qui est transmise au module de communication utilisateur ; ce dernier se chargera ensuite de la faire parvenir à l'utilisateur.

Le registre sauvegarde les données que l'utilisateur a fournies, de manière à regrouper par thème la cueillette d'information venant de celui-ci. Un tel découpage permet en fait de construire des formulaires HTML bien moins chargés et plus conviviaux.

4.3 Agent Ressource dans NetSA

L'agent ressource est aux données ce que l'agent utilisateur est à l'utilisateur. Cet agent reçoit des requêtes formulées en KQML et les transforme en requête SQL afin d'extraire l'information requise des bases de données où il recherche l'information demandée dans une page HTML.

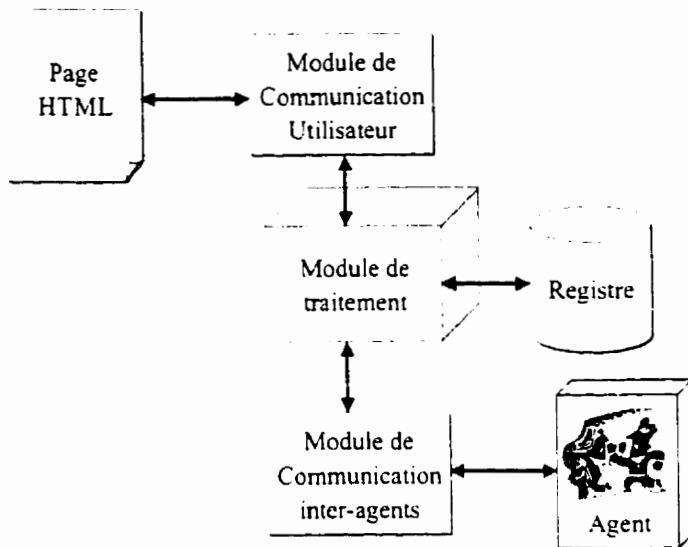


Figure 4.4: Architecture interne de l'agent utilisateur.

L'information trouvée est ensuite traduite en KQML pour répondre à la requête du demandeur. Principalement, un agent ressource gère une seule ressource. De toute évidence, plus le nombre d'agents ressource est grand, plus nous avons accès à une information complète et diverse.

4.3.1 Description

L'agent ressource a pour but d'interroger, d'extraire et de mettre à jour des données dans une base de données ou dans une page Web structurée. Il reçoit d'abord une requête d'un agent appelant, traduit cette requête, interroge la base de données et construit une réponse qui sera acheminée vers l'agent appelant.

La structure de la page Web utilisé a un format un peu spécial dans la mesure où nous avons ajouté des mots clés dans le code HTML pour reconnaître les champs donnant l'information spécifiée. Plus spécifiquement, nous avons introduit une étiquette nommée «VAR» qui décrit la donnée contenue dans la page Web. Par exemple, la représentation suivante donne à la variable Rate la valeur 4.3 :

Interest rate : <VAR = Rate> 4.3% </VAR>

4.3.2 Architecture interne

L'architecture interne de l'agent ressource est composée de 4 principaux modules comme indiquée en Figure 4.6. Ces modules donnent à l'agent une plus grande extensibilité étant donné leur découpage en tâches.

Le module de communication de l'agent ressource reçoit les requêtes sous la forme d'un messages KQML, suite à cela, il appelle le module de traitement. Il reçoit également des demandes de transmission de messages du module de traitement. Celles-ci

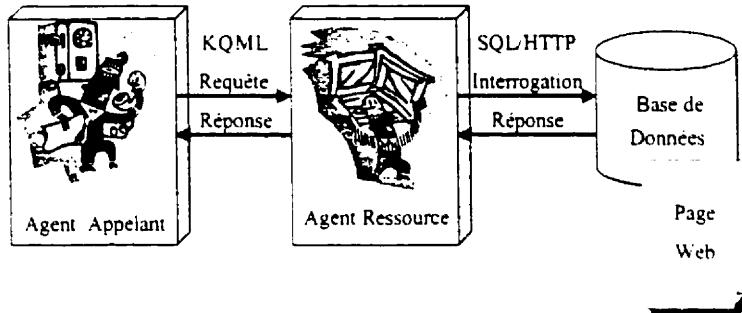


Figure 4.5: Description de l'agent ressource.

constituent des réponses aux requêtes reçues.

Le module de traitement traite les requêtes reçues par le module de communication. Pour cela, il la transforme en requête SQL et l'achemine l'interface avec vers la base de données ou ordonne une recherche à l'interface HTTP. Il construit ensuite une réponse sous forme de message KQML qu'il donne au module de communication pour être expédiée.

L'interface avec la base de données reçoit du module de traitement une demande de recherche de variable dans une base de données. L'interface exécute la requête SQL demandée par le module de traitement et retourne les valeurs trouvées à ce module.

L'interface HTTP reçoit du module de traitement une demande de recherche de variable dans une page Web. Elle parcourt alors la page Web à la recherche d'une étiquette «VAR» définissant la variable demandée. Elle retourne ensuite la valeur de cette variable au module de traitement.

4.3.3 Communication

Pour extraire les données nous utilisons les actes de KQML `ask-one` et `ask-all`. Le premier acte donne la première instance trouvée et le second retourne toutes les instances trouvées de l'information demandée. Pour modifier la base de données, nous utilisons les actes de langage suivants :

- `insert` pour insérer un nouvel enregistrement à la base de données,
- `delete-one` pour effacer la première instance trouvée à partir de l'information donnée,
- `delete-all` pour effacer toutes les instances trouvées à partir de l'information donnée,

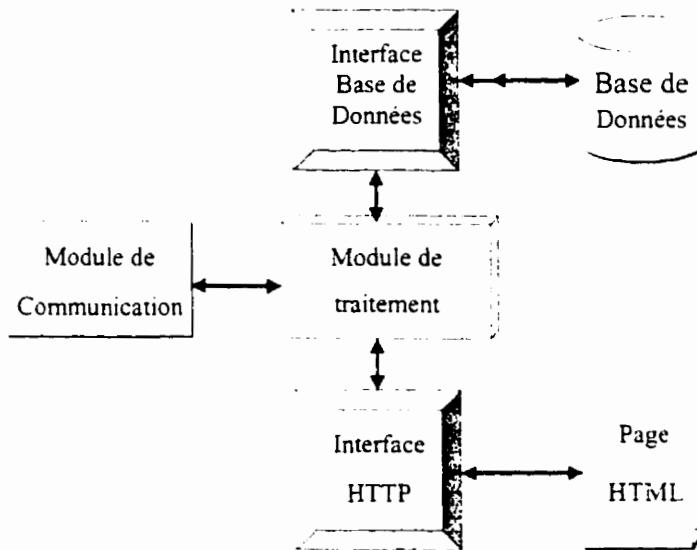


Figure 4.6: Architecture interne de l'agent resource.

- update pour modifier un enregistrement.

Les messages KQML reçus et envoyés par l'agent ressource suivent les formats illustrés à la Figure 4.7

4.4 Agent Intermédiaire dans NetSA

L'agent intermédiaire peut être vu comme un système de courtage. En effet, l'agent intermédiaire associe aux différentes requêtes les agents qui sont capables d'y répondre. Lors du démarrage, les agents doivent s'annoncer auprès de lui en lui envoyant un message KQML l'avertissant du type de service qu'ils sont capables de fournir. Lorsqu'un agent demande à l'agent intermédiaire s'il peut faire sa requête, celui-ci lui répond en lui donnant le nom du ou des agents aptes à la satisfaire. Il est à noter que nous pouvons rendre le système plus sécuritaire en ajoutant un second courtier qui serait capable de prendre la relève en cas de défaillance tout en allégeant la charge du courtier existant.

4.4.1 Description

Une des caractéristiques les plus importantes de NetSA est l'utilisation d'un système de courtage intelligent. Ce système permet de faire du courtage de deux façons : le courtage syntaxique et le courtage sémantique. Du point de vue utilisateur, un courtage sémantique permet aux requêtes d'être spécifiées en utilisant les concepts des ontologies²; les spécifications sont par la suite utilisées pour chercher les agents ressources qui sont capables de traiter les requêtes visées. Les ressources sélectionnées sont celles qui correspondent le mieux aux besoins des agents en quête de ressources.

²Une ontologie est une spécification d'une conceptualisation.

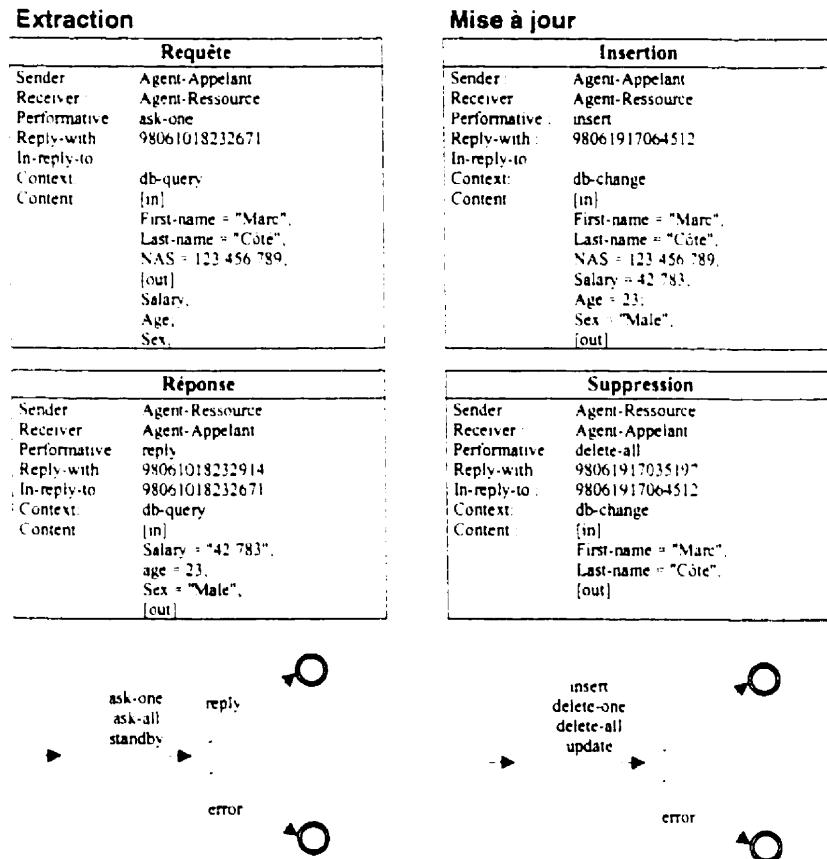


Figure 4.7: Exemple de communication pour l'agent ressource.

Le système de courtage tel qu'il vient d'être décrit se base essentiellement sur l'agent Intermédiaire. Dans d'autres systèmes, ce type d'agent est souvent désigné par courtier ou page jaune. Il existe plusieurs types d'agent intermédiaire que nous pouvons classer comme dans le tableau 4.1 (Dans ce tableau C, F et I signifient respectivement : Client, Fournisseurs et agent Intermédiaire).

demandes connues par	services connus par		
	F	F + I	F + I + C
C	diffuseur	représentant	pages jaunes
C + I	protecteur	courtier	conseiller
C + I + F	blackboard	garde du corps	médiateur

Tableau 4.1: Classification des agents intermédiaires

Dans l'architecture NetSA, l'agent intermédiaire est du type *pages jaunes* (voir Chapitre 3) donc l'agent intermédiaire ne donne que le nom de l'agent lors d'une requête. Par opposition, un agent intermédiaire qui serait du type *courtier* recevrait de l'agent client une requête pour un service. L'agent intermédiaire dans ce cas questionnerait directement les agents concernés et donnerait la «meilleure» réponse à l'agent client.

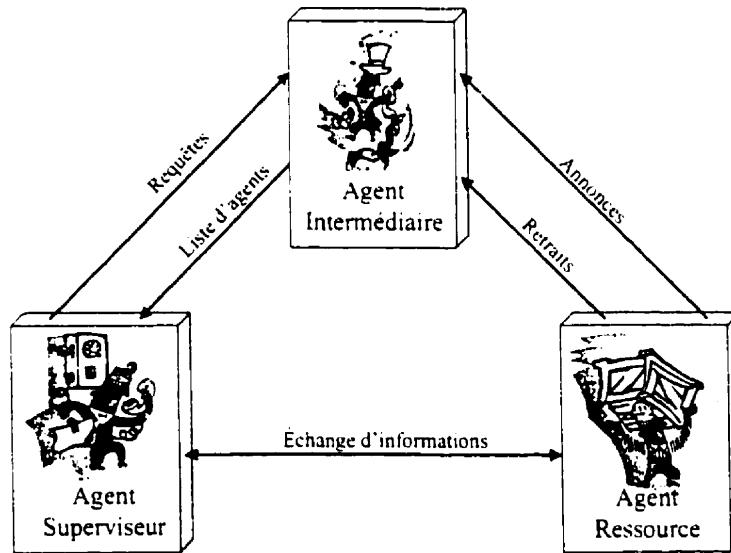


Figure 4.8: Description de l'agent intermédiaire.

4.4.2 Architecture Interne

L'architecture de l'agent intermédiaire comme le montre la Figure 4.9 comporte deux bases de données. La première est une base de règles qui fournit à l'agent le

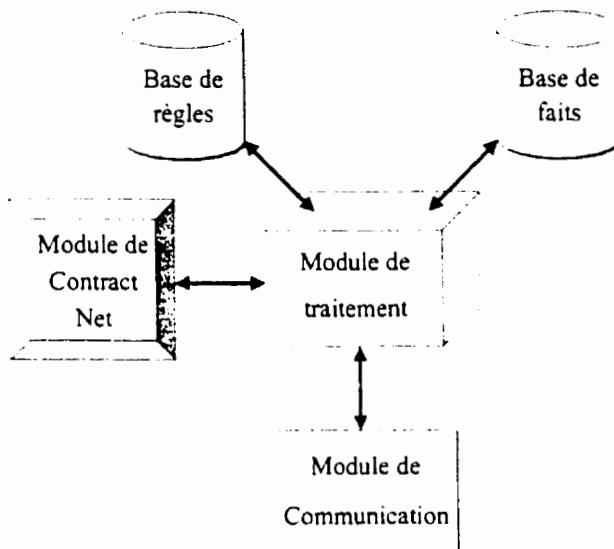


Figure 4.9: Architecture interne de l'agent Intermédiaire.

protocole à suivre lors de la communication. La deuxième est une base de faits, dans laquelle l'agent emmagasine les connaissances qu'il a de son environnement.

Le module de communication est une interface entre l'agent et son environnement. Il est utilisé pour transmettre et recevoir des messages sous la forme KQML.

Le module de réseaux à contrats (i.e. *Contract Net*) gère spécialement le protocole de réseau à contrats dont les spécifications sont stockées dans la base de règles. Le réseaux à contrats est un protocole de négociation qui s'utilise de la façon suivante : l'agent ayant besoin d'aide (ici l'agent intermédiaire) fait une annonce pour une tâche en spécifiant le travail à exécuter et les contraintes. Les sous-contractants potentiels répondent à l'agent appelant et ce dernier choisit l'agent répondant le mieux à ces attentes. Un moteur d'inférence développé en Java et nommé JESS est utilisé pour effectuer l'échange selon le réseaux à contrats.

Pour coordonner tous les modules de l'agent intermédiaire, un module de traitement a été développé. La synchronisation et la majorité des calculs sont effectués par ce module.

4.5 Agent Superviseur dans NetSA

L'agent superviseur est responsable de l'exécution à haut niveau des requêtes venant des agents utilisateurs. Lorsqu'il reçoit une requête, celle-ci est exécutée par le biais de plans prédéfinis. Majoritairement, un plan est composé de plusieurs sous-plans. Par exemple, un plan pour préparer un repas pourrait donner lieu à l'exécution des 10 sous-plans suivants :

1. Préparation de l'entrée,
2. Préparation de la soupe,

3. Préparation du plat principal,
4. Préparation du dessert,
5. Mettre la table,
6. Servir l'entrée,
7. Servir la soupe,
8. Servir le plat principal,
9. Servir le dessert,
10. Desservir la table.

Un sous-plan peut être également composé d'autres sous-plans. Si nous reprenons l'exemple précédent, le sous-plan qui consiste à servir la soupe pourrait se décomposer de la manière suivante :

1. Prendre un bol.
2. Agiter la soupe,
3. Verser la soupe dans le bol.
4. Mettre le bol sur la table.

Les plans sont programmés dans un langage spécialisé nommé POPA, un langage que nous avons développé pour les besoins de la cause. Une description de ce langage est faite à l'annexe A.

Lorsque l'agent superviseur manque d'informations pour la réalisation d'un plan, il recherche cette information auprès des agents ressources par une requête formulée dans la norme de NetSA. Après l'exécution du plan, l'agent superviseur synthétise les réponses et transmet une réponse finale à l'agent appelant ayant fait la requête d'origine.

4.5.1 Description

L'agent superviseur est au cœur de l'exécution des requêtes de NetSA. Situé dans la couche centrale de l'architecture, il est responsable de la quasi totalité des requêtes. Comme nous pouvons l'observer en Figure 4.10, l'agent superviseur est directement connecté à toute les couches de l'architecture voire même en liaison directe avec tous les agents.

4.5.2 Architecture interne

Comme tout agent de NetSA, l'agent superviseur à sa propre architecture interne. Celle-ci est divisée en trois modules : (1) un module de communication, (2) un module de planification et (3) un module d'évaluation d'expression. Ces trois modules sont accompagnés de deux base de données : la base de registre et la base des plans.

Examinons d'un peu plus près chacun des modules de l'agent superviseur.

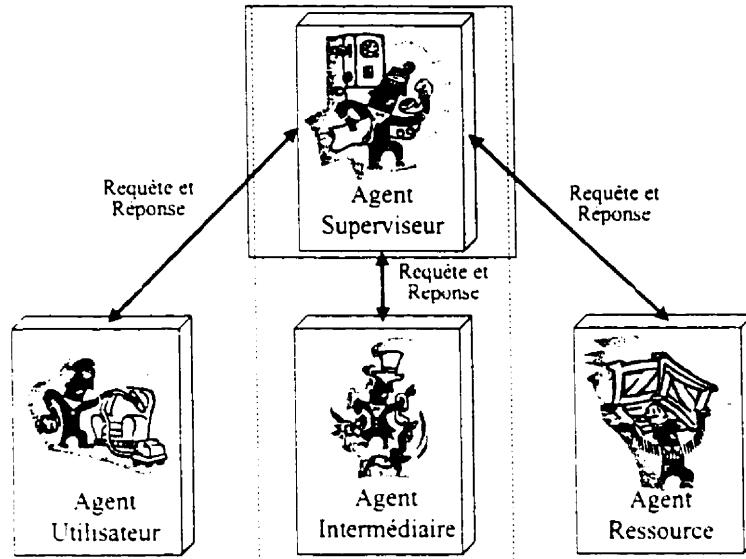


Figure 4.10: Description de l'agent superviseur.

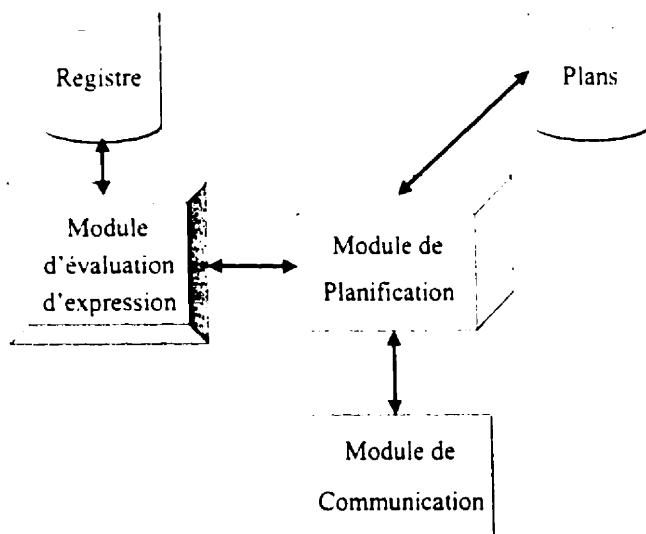


Figure 4.11: Architecture interne de l'agent superviseur.

Module de communication

Le module de communication est le lieu de transition de tous les messages entrant et sortant de l'agent superviseur. Il vérifie la syntaxe des messages reçus et appelle le plan nécessaire à la réalisation de la requête contenue dans le message. Il y a 5 types de messages reconnus et échangés par le module de communication.

1. *Réception d'une requête* : Cette réception de message provoque le démarrage de l'exécution d'un plan et contient l'acte de langage `ask-one` ou `ask-all`. Le contenu du message contient les informations pertinentes à la réalisation du plan dans le champ d'entrée (`[in]`) et le nom du plan souhaité dans le champ de sortie (`[out]`). Ce type de message provient la plupart du temps d'un agent utilisateur mais ce n'est qu'une constatation et non une restriction.
2. *Réception d'une réponse* : Les messages de ce type contiennent l'acte de langage `reply` ce qui déclenche un processus insertion des données contenues dans le champ d'entrée (`[in]`) vers la base de registre. Le champ de sortie (`[out]`) est habituellement vide dans ce cas.
3. *Transmission de requête ressource* : Le module de communication construit un message KQML pour être envoyé vers un agent ressource en utilisant les actes de langage `ask-one` ou `ask-all`. Le champ d'entrée du contenu du message contient les restrictions de la recherche d'information. Le champ de sortie indique quelle information doit être retournée.
4. *Transmission de requête page jaune* : Le module construit ici un message KQML pour être envoyé vers un agent intermédiaire en utilisant les actes de langage `recommend-one` ou `recommend-all`. Le champ d'entrée du contenu du message contient les restrictions de la recherche d'un agent. Le champ de sortie contient la valeur `AgentName` qui signifie à l'agent intermédiaire qu'il doit retourner le nom de l'agent trouvé.
5. *Transmission de réponse* : Lorsqu'un plan est terminé, le module de communication construit un message KQML utilisant l'acte de langage `reply` et le transmet au demandeur. La réponse est donnée dans le champ d'entrée du contenu du message. Pour l'agent utilisateur, nous ajoutons dans le champ de sortie le nom de la page HTML modèle qu'il devra utiliser pour afficher la réponse à l'utilisateur.

Le module de communication utilise les classes de JATLite pour effectuer le service de transmission de messages.

Module de gestion des plans

Étant donné que le plan est la base du raisonnement de l'agent superviseur, il nous a fallu un module pour interpréter les plans. Une base de plans est premièrement chargée lors de l'initialisation du module de gestion des plans. Par la suite, un registre est créé pour le stockage des données pendant la réalisation du plan.

Chaque plan est une suite d'expression écrite dans le langage POPA. Chacune de ces expressions est donc soumise au module d'évaluation d'expression. L'exécution d'un

plan doit être vu comme un processus indépendant ce qui permet à plusieurs plans de pouvoir s'exécuter en concurrence.

Module d'évaluation d'expression

Le module d'évaluation d'expression permet l'exécution du contenu des plans. Ce module est capable d'évaluer des expressions écrites dans le langage POPA. Il est donc utilisé pour faire des calculs, envoyer des messages, vérifier des informations, etc. La syntaxe de POPA est décrite dans l'annexe A.

4.6 Conclusion

Dans ce chapitre, nous avons présenté l'architecture NetSA qui est une architecture multiagent à trois couches. La première couche (l'unité de communication avec l'utilisateur) s'occupe de la traduction des requêtes de l'utilisateur en messages KQML compris par l'architecture. La seconde couche (l'unité de traitement de l'information) s'occupe de la réalisation des requêtes de l'utilisateur. La dernière couche (l'unité d'interrogation et d'extraction de l'information) exploite les informations contenues dans des sources distribuées et hétérogènes.

Nous avons également décrit les différents agents faisant partie de chacune des couches de l'architecture NetSA. La première couche est constituée essentiellement d'au moins un agent utilisateur. Dans la couche centrale, nous retrouvons les agents superviseurs et intermédiaires. De leur côté, les agents ressources sont localisés dans la couche inférieure.

Chapitre 5

Enchères

Lorsque nous parlons d'enchères nous imaginons presque toujours la vente d'un produit où le prix de base est fixé par un encanteur et où les gens misent des montants d'argent de plus en plus élevés jusqu'à ce qu'il ne reste qu'un acheteur potentiel. Dans ce cas, l'encanteur s'écrie : «une fois... deux fois... trois fois... vendu!» et le produit est ainsi enlevé par le dernier enchérisseur au prix qu'il a donné. Il y a bien entendu d'autres techniques d'enchères en particulier, la hollandaise, le meilleur prix, la Vickrey, et double et bien d'autres (généralement des variantes). Ces enchères ont créé un réel engouement dans Internet. Des sites comme *Auction Universe* [Auc98] vendent tous les jours une variété de produits en utilisant les enchères.

Chacun de ces types d'enchères a son protocole, ses avantages et ses inconvénients. Nous décrirons dans les sections suivantes de ce chapitre les différents type d'enchères.

5.1 Enchères anglaises

Les enchères anglaises aussi connu sous le nom de criée ou enchères à prix ascendant, sont les enchères les plus populaires. Elles sont utilisées pour vendre des œuvres d'art, du vin et bien d'autres marchandises.

L'enchère anglaise débute par le plus bas prix acceptable (prix du marché) pour un bien donné. Par une succession de mises de plus en plus élevées de la part des enchérisseurs, le prix du bien augmente jusqu'à ce qu'un seul enchérisseur demeure dans la course, les autres ayant jugé trop élevé le prix demandé pour le bien convoité (voir Figure 5.1). L'encanteur prononce, après un certain temps sans offre, le bien vendu à cet enchérisseur qui a offert le plus d'argent.

Contrairement aux croyances, ce ne sont pas toutes les marchandises d'une enchère qui sont concrètement et définitivement vendues. Dans quelques cas, quand le prix du marché n'a pas été rencontré, la vente de l'article est annulée. Cette possibilité d'annulation de la vente doit être convenue avant le début l'enchère. Dans le cas où il y a possibilité d'annulation, si le prix minimum n'est pas obtenu, l'encanteur peut interrompre la vente et tenter de la reprendre plus tard.

Quelquefois l'encanteur maintiendra le prix du marché secret. Il doit commencer l'enchère sans révéler le prix acceptable le plus bas. Une explication possible pour ce

secret est qu'il vise à contrecarrer les plans des enchérisseurs qui se sont entendus pour obtenir le bien à un prix très bas. En cachant le prix de base, l'encanteur peut faire gonflé le prix du produit bien en haut de sa valeur réelle étant donné que tous les acheteurs ignorent sa véritable valeur.

En dépit de sa simplicité apparente, ce format d'encheres peut devenir assez complexe. Souvent les mises ne sont pas faites à haute voix, mais plutôt signalées en tirant sur son oreille, en montrant un carton de mise, etc. Ce système de signalement a plusieurs avantages :

- Il permet d'éviter les chahuts,
 - Les encanteurs étant des personnes humaines, ils peuvent mal entendre les mises s'il y a beaucoup de bruit lors de la vente et désavantager certaines personnes.
- Le système de signes élimine les erreurs reliées à l'audition de l'encanteur.

Beaucoup de négociants préfèrent le semi-anonymat. Un expert connu dans un certain domaine peut vouloir que les autres ne sachent pas qu'il a fait une offre parce que cela influencerait probablement à la hausse la valeur du produit, ce qui n'est pas souhaité par l'expert. Dans ce cas, l'encanteur propose des prix par une augmentation graduelle de la valeur du produit et les enchérisseurs font un geste discret pour accepter le prix. Le montant de l'augmentation de la mise est à la discréction de l'encanteur. Il arrive souvent que l'encanteur par ses gestes et son intonation incite les acheteurs à miser de plus en plus haut.

Bien entendu, la compétition pourrait atteindre son plus haut dans l'encheré anglaise. Parfois nous pouvons observer de réelles guerres de prix pour savoir qui enlèvera le produit, bien que le prix réel du produit soit largement dépassé. La clef à toute encheré prospère (du point de vue du vendeur) est l'effet de compétition sur les acheteurs potentiels. C'est ce que l'encheré anglaise favorise !

5.1.1 Avantages et inconvénients

La simplicité et la popularité de ce type d'encheré en font un excellent moyen de vente. La possibilité de pouvoir annuler une vente qui n'a pas rencontré le prix du marché est un atout pour le vendeur.

Il est toutefois facile pour un groupe d'acheteurs de former une coalition qui a pour but de faire diminuer le prix des biens. Dans le cas de la vente d'animaux de boucherie où il y a peu d'acheteurs par exemple, trois acheteurs peuvent s'unir pour ne pas dépasser un montant d'argent.

Un autre inconvénient est que les acheteurs doivent être présents lors de la vente. Ils doivent donc se déplacer ce qui peut être coûteux pour l'acheteur. Il risque donc d'y avoir moins d'acheteurs si l'encaen a lieu dans un endroit éloigné.

5.1.2 Algorithme

Dans l'algorithme de l'encheré anglaise, nous débutons avec un prix qui est arbitrairement 3 fois moins que le prix réel du produit. Nous utilisons également une

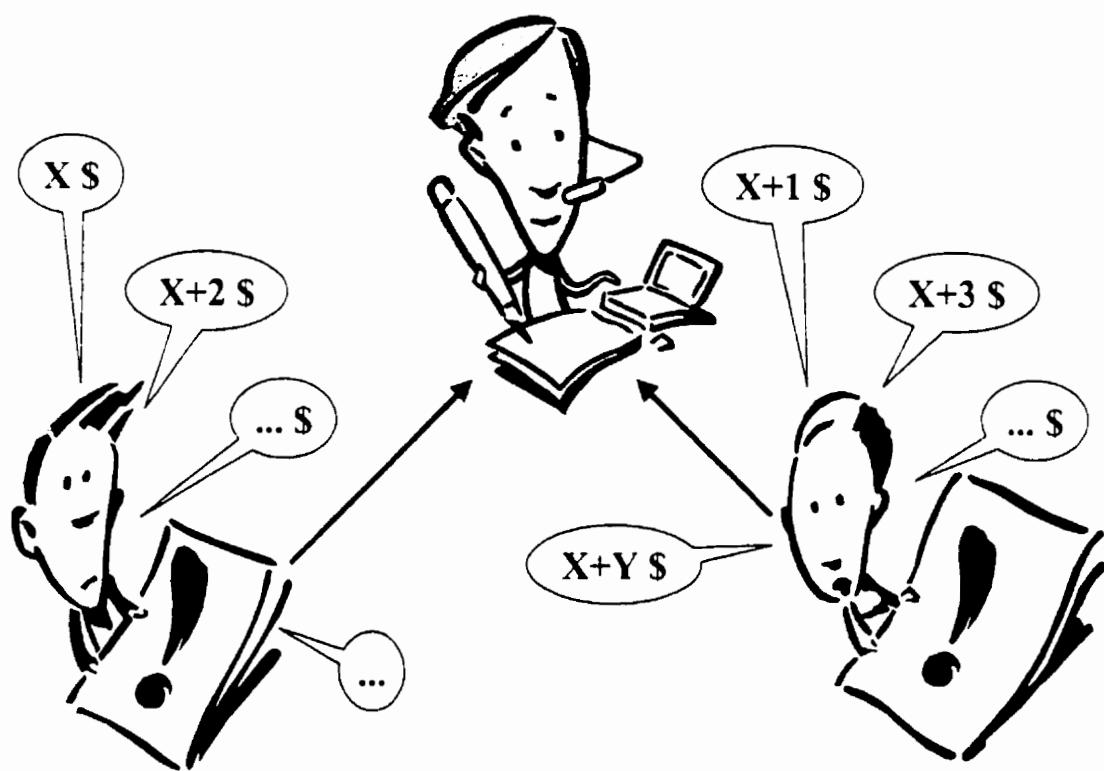


Figure 5.1: Exemple du fonctionnement de l'enchère anglaise.

minuterie (Timer) pour limiter le temps pour faire une offre. Si le temps alloué est expiré alors le bien est attribué au dernier enchérisseur.

```

Prix ← PrixRéel ÷ 3
DernierEnchérisseurs ← null
Timer ← 10
Annoncer(Prix, DernierEnchérisseurs)
TANTQUE Timer > 0
    Offre ← RecevoirOffres()
    SI Offre.Prix > Prix ALORS
        Prix ← Offre.Prix
        DernierEnchérisseurs ← Offre.Enchérisseur
        Timer ← 10
        Annoncer(Prix, DernierEnchérisseurs)
    FIN SI
FIN TANTQUE
AnnoncerGagnant(Prix, DernierEnchérisseurs)
```

5.2 Enchères hollandaises

L'enchère à prix descendant, mieux connue dans la littérature académique comme l'enchère hollandaise, utilise un format de mise publique (comme l'enchère anglaise) plutôt qu'une méthode de mise secrète. Cette technique est utilisée en Hollande pour vendre aux enchères des produits alimentaires et des fleurs. Malheureusement, le monde financier a boudé l'enchère hollandaise.

Dans l'enchère hollandaise, une offre est faite dès le début à un prix extrêmement haut. L'encanteur diminue le prix progressivement jusqu'à ce qu'un acheteur désire l'article en criant «le mien» (voir Figure 5.2), ou en appuyant sur un bouton qui signale la vente de l'article. Quand des articles multiples (comme les fleurs) sont vendus aux enchères, après le premier arrêt du prix, l'enchère continue et les prix descendent jusqu'à ce qu'il n'y ai plus de marchandise. À chaque arrêt, il revient à l'acheteur ayant crié «le mien» de choisir les produits de meilleure qualité.

Les enchères hollandaises ont été utilisées pour financer le crédit en Roumanie, pour l'échange de devises étrangères en Bolivie, Jamaïque, Zambie et pour vendre du poisson en Angleterre et en Israël.

5.2.1 Avantages et inconvénients

Étant donné que le prix débute à un seuil très élevé, il y a moins de risque pour le vendeur d'obtenir un prix inférieur au prix du marché. Si une personne veut vraiment un article il n'a pas à attendre longtemps avant de faire son offre. Lorsqu'il y a plusieurs objets du même type à vendre (comme les fleurs) cette méthode de vente pourrait s'avérer rapide et efficace.

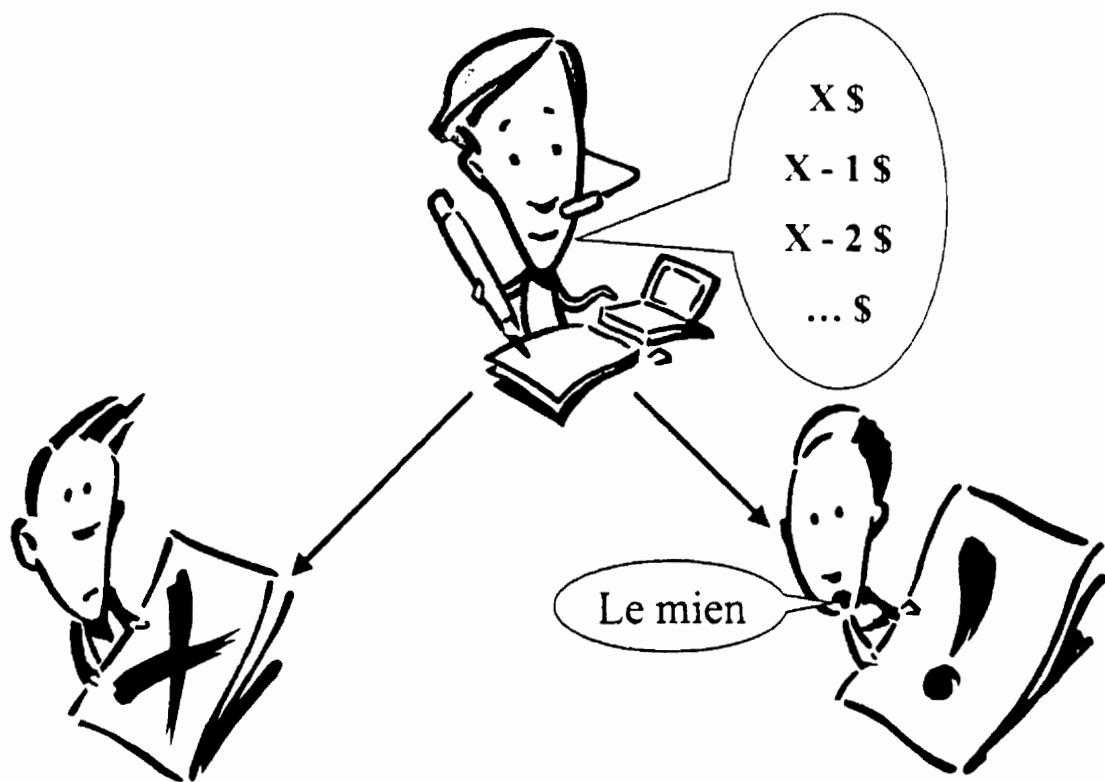


Figure 5.2: Exemple du fonctionnement de l'enchère hollandaise.

Du côté des inconvénients, nous ne voyons pas de compétition pouvant faire monter le prix d'un article à un niveau exorbitant comme dans le cas de l'enchère anglaise. Il faudrait mettre le prix de base à un prix exorbitant pour observer ce phénomène mais cela ralentirait le processus de vente. Comme pour l'enchère anglaise, il est facile pour un groupe d'acheteurs de former une coalition qui a pour but de faire diminuer le prix des marchandises.

5.2.2 Algorithme

L'algorithme de l'enchère hollandaise débute arbitrairement avec un prix 3 fois (par exemple) plus élevé que le prix réel du produit. À chaque itération, nous décrémentons le prix du produit et ce tant que personne n'accepte le prix proposé.

```

Prix ← PrixRéel × 3
Annoncer(Prix)
Offre ← RecevoirOffres()
TANTQUE Offre = null
    Prix ← Prix * 0.99
    Annoncer(Prix)
    Offre ← RecevoirOffres()
FIN TANTQUE
AnnoncerGagnant(Prix, Offre.Échérisseur)
```

5.3 Enchères du meilleur prix

Cette enchère a comme caractéristique principale le fait d'être secrète. Les mises sont faites par écrit et mise dans des enveloppes scellées pour conserver la confidentialité. Le gagnant de l'enchère est celui qui a la mise la plus élevée comme son nom l'indique (voir Figure 5.3).

Il y a donc deux phases dans le processus de vente selon cette enchère qui ressemble énormément au processus d'élection :

1. Phase de proposition ; les enchérisseurs font une offre.
2. Phase de compilation ; l'encanteur fait le dépouillement des mise pour déterminer le vainqueur.

Dans le cas de la mise en enchère de plusieurs produits de même type (comme les fleurs), la personne ayant la plus haute mise choisira la première, suivie de la deuxième plus haute mise et ainsi de suite jusqu'à ce qu'il n'y ai plus rien à vendre. Donc, ce ne sont pas tous les acheteurs qui auront la chance d'obtenir quelque chose.

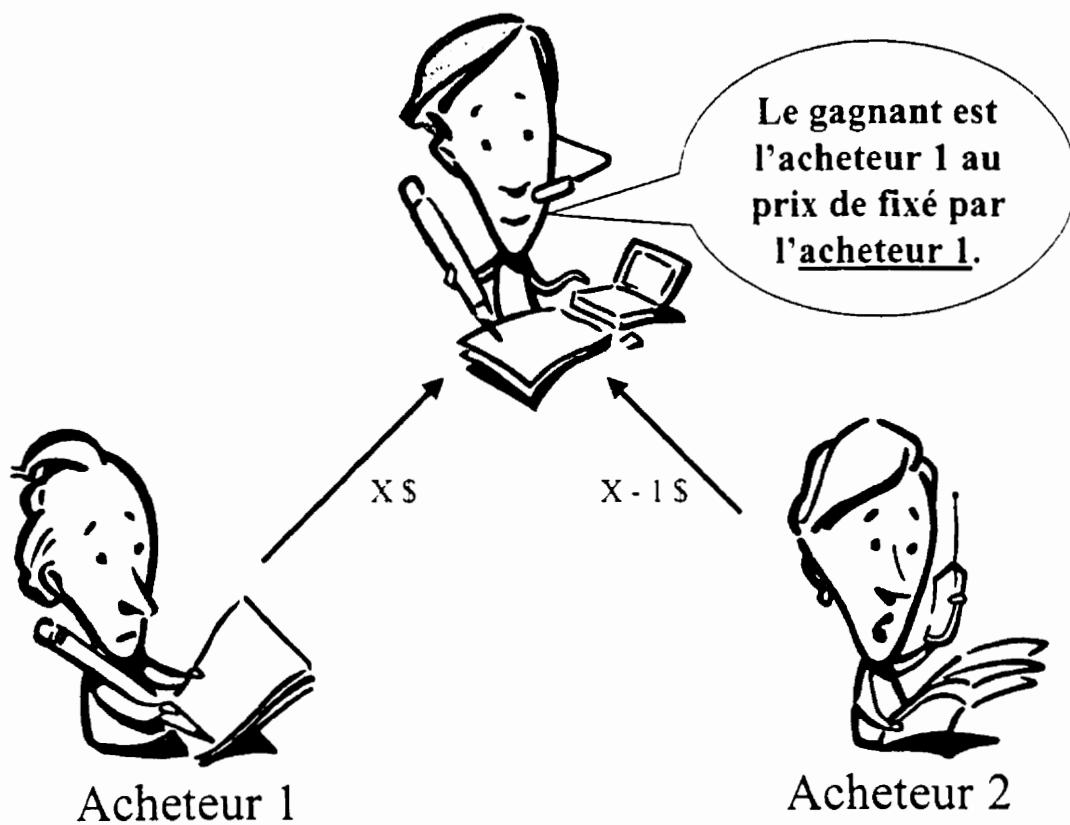


Figure 5.3: Exemple du fonctionnement de l'enquête du meilleur prix.

5.3.1 Avantages et inconvénients

Pour une personne qui désire vraiment le produit vendu, il lui est possible de l'obtenir en misant une très grosse somme. Or, ce n'est généralement pas le cas et les prix ont plutôt tendance à se rapprocher du prix du marché, car les gens ne veulent pas payer trop cher. Donc une bonne stratégie serait de miser un peu plus bas que le pris du marché pour obtenir un bon prix. Les mises scellées offrent aussi l'avantage de ne pas influencer les enchérisseurs. Il est également plus difficile de former des ententes malhonnêtes, car les enchérisseurs peuvent venir de partout. Finalement, les enchérisseurs n'ont pas besoin d'être présents lors de la vente et la mise peut se faire par la poste.

Le plus grand inconvénient de ce type d'enchère est qu'une personne possédant une très grande fortune est fortement avantagée. En misant une somme considérable, elle empêche les autres acheteurs de participer.

5.3.2 Algorithme

Au début de l'algorithme, on annonce le produit à vendre. Il faut ensuite recevoir les offres venant des enchérisseurs. Finalement, les offres sont dépouillées afin de trouver l'offre la plus élevée.

```

Annoncer(ObjetAVendre)
Offre ← RecevoirOffres()
MeilleureOffre ← null
POUR toute les Offres reçues
    SI Offre.Prix > MeilleureOffre.Prix ALORS
        MeilleureOffre ← Offre
    FIN SI
FIN POUR
AnnoncerGagnant(MeilleureOffre.Prix, MeilleureOffre.Enchérisseur)
```

5.4 Enchères de Vickrey

Portant le nom de son créateur, William Vickrey [Vic61] (prix Nobel en économie de 1996), l'enchère de Vickrey est proche de l'enchère dite de meilleur prix vue précédemment. Comme les mises sont déposées dans des enveloppes scellées pour des raisons d'anonymat. C'est également la plus haute mise qui l'emporte mais le gagnant paiera le prix de la deuxième plus haute mise comme le montre la Figure 5.4.

Par exemple, si nous avons trois enchérisseurs. Le premier enchérisseur offre 10 \$, le deuxième 15 \$ et le troisième 20 \$. Le troisième enchérisseur sera le vainqueur, car il a misé 20 \$ mais il paiera seulement 15\$ soit la deuxième plus haute mise.

Lorsqu'il y a plusieurs produits de même type, l'enchère fonctionne un peu différemment. Pour n produits, les n plus hautes mises paieront le montant de la première mise sous la valeur de leurs mises.

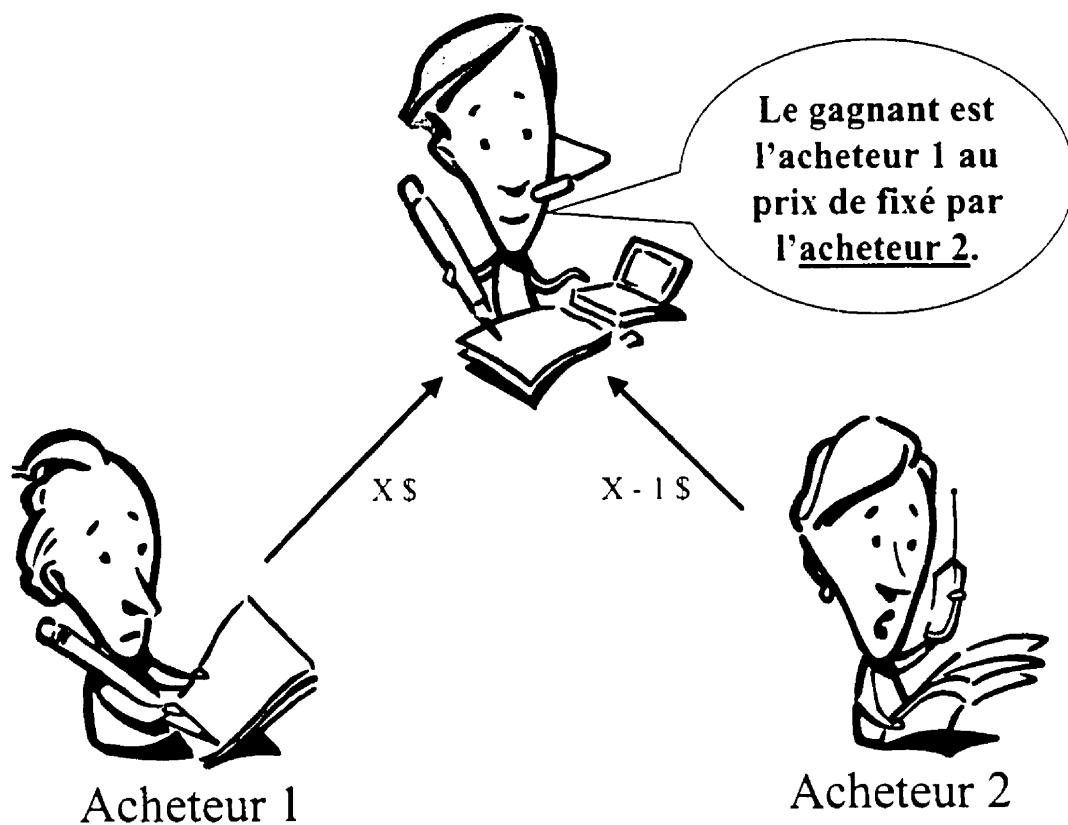


Figure 5.4: Exemple du fonctionnement de l'encheré Vickrey.

5.4.1 Avantages et inconvénients

Le but de ce type d'enchères est de donner un prix au produit qui se rapproche de celui du marché. Nous voulons éviter que des gens trop zélés fassent augmenter le prix d'un produit démesurément. Il ne semble pas intéressant pour un vendeur d'utiliser ce mode de vente, car les prix semblent suivre le marché. Mais ce n'est qu'une illusion. Des études ont prouvé que les gens misent toujours un peu plus haut, car ils savent que le prix payé sera moindre que la mise faite.

Malheureusement, lors de la vente de plusieurs produits de même type, les personnes ayant misé bas devront quand même payer un prix plus élevé que leur mise.

5.4.2 Algorithme

L'algorithme débute par l'annonce du produit à vendre. Les offres venant des enchérisseurs sont ensuite reçues et dépouillées afin de trouver la première et la deuxième plus élevée. Finalement, on adjuge le bien à celui qui a offert la meilleure mise, en exigeant de lui qu'il paye la deuxième meilleure offre.

```

Annoncer(ObjetAVendre)
Offre ← RecevoirOffres()
MeilleureOffre ← null
SecondeMeilleureOffre ← null
POUR toute les Offres reçues
    SI Offre.Prix > MeilleureOffre.Prix ALORS
        SecondeMeilleureOffre ← MeilleureOffre
        MeilleureOffre ← Offre
    FIN SI
FIN POUR
AnnoncerGagnant(SecondeMeilleureOffre.Prix, MeilleureOffre.Enchérisseur)
```

5.5 Enchères doubles

Bien que ce ne soit pas un type d'enchères familier, l'enchère double est la principale forme de commerce des institutions financières américaines depuis environ 100 ans. Son nom lui vient du fait que les acheteurs et les vendeurs font une offre pour le produit désiré.

Nous avons donc d'un côté des offres de vente triées en ordre croissant et de l'autre des offres d'achat triées en ordre décroissant. Nous associons les offres une à une tant que l'offre d'achat est supérieure ou égale à l'offre de vente [FR93b].

Par exemple, comme le montre la Figure 5.5, pour des offres de ventes de 400 \$, 300 \$, 200 \$ et 100\$ et des offres d'achat de 400\$, 300 \$, 250 \$ et 50 \$ [FR93a] :

- l'offre d'achat de 400 \$ sera jumelé à l'offre de vente de 100 \$,
- l'offre d'achat de 300 \$ sera jumelé à l'offre de vente de 200 \$,

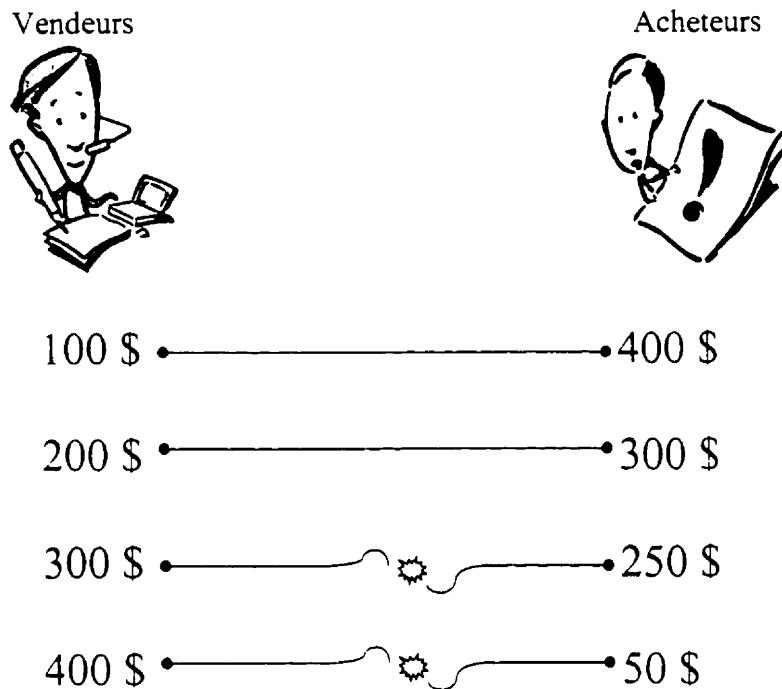


Figure 5.5: Exemple du fonctionnement de l'enquête double.

- les offres d'achat de 250 \$ et 50 \$ et les offres de vente de 300\$ et 400 \$ ne trouveront pas preneurs.

5.5.1 Avantages et inconvénients

Une personne aimant le risque sera avantagée par ce genre d'enquêtes. Les économistes prévoient un avenir prometteur pour ce type d'enquêtes dans le domaine du commerce électronique. Du point de vue des inconvénients, nous n'en voyons aucun.

5.5.2 Algorithme

Pour réaliser la vente aux enquêtes doubles, il nous faut 2 listes : une première contenant les offres de ventes triée par prix croissant et une deuxième contenant les offres d'achats triée par ordre décroissant. Il faut ensuite parcourir séquentiellement la liste en attribuant les offres de la liste des offres de ventes à leurs homologues dans la liste des offres d'achats tant que le prix de vente est inférieur au prix d'achat.

ListeVente est une liste croissante

ListeAchat est une liste décroissante

```

TANTQUE Vrai
    Offre ← RecevoirOffres()
    SI estUneVente(Offre) ALORS
        Ajouter(Offre, ListeVente)
    SINON
        Ajouter(Offre, ListeAchat)
    FIN SI
    TANTQUE ListeVente.Item(0).Prix < ListeAchat.Item(0).Prix
        ConclureVente(ListeVente.Item(0), ListeAchat.Item(0))
    FIN TANTQUE
FIN TANTQUE

```

5.6 Conclusion

Les enchères anglaises et hollandaises sont des enchères itératives où plusieurs séquences d'offres sont soumises. Ce type d'enchères demande énormément de communication entre les deux parties (vendeurs et acheteurs) et ont une durée de vie assez longue étant donnée le grand nombre d'itérations requises avant la conclusion d'une vente. Ce type d'enchère itérative peut donc être utilisé lorsqu'il n'y a pas de contrainte de temps ou lorsque cette contrainte est suffisamment grande.

Les enchères Vickrey, meilleur prix et double nécessitent qu'une seule itération pour ce qui est des offres. Donc lorsque la contrainte de temps est limitée, ces enchères sont fortement conseillées. C'est pourquoi nous avons convenu d'utiliser les enchères de type non-itérative pour planter un système de courtage de prêts hypothécaires dans NetSA.

Chapitre 6

Hypothèque

Dans la mesure où on veut utiliser NetSA pour la compétition entre plusieurs banques dans le cadre de l'attribution des prêts hypothécaires, nous avons jugé bon de dire en quoi consiste une hypothèque. Nous avons également jugé bon de décrire les éléments qui peuvent faire varier le coût entre les prêts hypothécaires des différentes banques.

Un prêt hypothécaire est un prêt accordé par une institution financière à un demandeur dans le but d'acquérir une maison ou un immeuble. L'immeuble ou la maison achetée est la garantie de paiement de l'hypothèque. Dans ce cas, la maison demeure la propriété de l'institution financière tant que le paiement total n'est pas effectué. La maison fait gage de garantie de paiement.

Un prêt hypothécaire a différentes caractéristiques, ainsi que des options disponibles pour l'emprunteur, afin de lui accorder certains droits au cours du paiement de son hypothèque. Nous allons examiner de près ces caractéristiques.

6.1 Période d'amortissement

La période d'amortissement est la période totale sur laquelle s'étire le paiement de l'hypothèque. Cette période peut être de 10, 15, 20 ou 25 ans, certaines institutions offrent aussi 30 ans. Plus la période d'amortissement est longue, plus nous payons d'intérêt. Nous pouvons sauver beaucoup d'argent en intérêt en payant sur 20 ans au lieu de 25 ans, et cela si notre budget le permet.

6.2 Terme

Le terme est la période de validité du contrat d'hypothèque, renouvelable après chaque terme. Ainsi, une période d'amortissement de 25 ans est divisée en plusieurs termes. À chaque terme, l'hypothèque doit être renégociée par les 2 parties (emprunteur et institution financière), au taux d'intérêt en vigueur lors de la renégociation. Ainsi, quand les taux d'intérêt sont bas, l'emprunteur a avantage à préférer un certain type de prêt, puis au renouvellement du terme il pourrait, au contraire, opter pour un autre type de prêt si les taux ont changé. Un terme est habituellement entre 6 mois et 5 ans,

mais peut parfois s'étirer jusqu'à 10 ans. En général, plus le terme est court, plus les taux d'intérêts sont bas pour un même type de prêt. Par exemple, un prêt à taux fixe sur un terme de 2 ans aura un taux d'intérêt inférieur à un prêt sur un terme de 5 ans.

6.3 Types de prêts hypothécaires

Un prêt peut d'abord être ouvert ou fermé. Un prêt fermé ne peut être remboursé par anticipation (c-à-d en payant un montant d'argent supplémentaire qui réduira le montant restant à rembourser ou qui couvrira la totalité du prêt), et ce sans pénalités à payer. Il ne peut pas être non plus renégocié. Un prêt ouvert peut être remboursé par anticipation, en totalité ou en partie, n'importe quand, sans pénalités et nous pouvons également le renégocier.

Les clients ont aussi à choisir entre un prêt à taux fixe et un prêt à taux variable. Les prêts à taux fixe ont un taux qui ne varie pas pour toute la durée du terme. Le prêt à taux variable a un taux qui suit les fluctuations du taux hypothécaire en vigueur. Cependant, des options sont offertes, comme des prêts à taux variable mais à taux plafond, qui nous assurent de ne jamais dépasser la valeur maximum ce taux. Les prêts à taux variable ont en général de meilleurs taux que les prêts à taux fixe, quoique avec un risque plus élevé.

6.4 Taux d'intérêt

Chaque banque a ses propres taux d'intérêt, et ce pour chaque type d'hypothèque. Ainsi, pour chaque type de prêt et chaque terme offert par une institution financière, un taux d'intérêt est associé. Le taux d'intérêt ou taux hypothécaire minimum sera bien sur le plus avantageux, réduisant ainsi les frais d'intérêt.

6.5 Fréquence de paiement

Les institutions financières offrent les options de paiement suivantes :

- Paiement mensuel (une date fixe à chaque mois) ;
- Paiement bimensuel (2 dates fixes dans le mois, le 1 et le 15 du mois par exemple) ;
- Paiement hebdomadaire (paiement annuel divisé par 52) ;
- Paiement aux 2 semaines (paiement annuel divisé par 26) ;
- Paiement aux 2 semaines accéléré (paiement d'un mois divisé par 2) ;
- Paiement hebdomadaire accéléré (paiement d'un mois divisé par 4).

Les paiements plus rapprochés font sauver des frais d'intérêt, ainsi du plus cher au plus économique, nous avons : le paiement mensuel, bimensuel, aux deux semaines, et à la semaine.

Ensuite, nous avons les options de paiement accéléré, qui consistent à prendre le paiement d'un mois et à le diviser en deux, et nous payions cette somme aux deux

semaines. Ainsi, après 24 paiements nous avons payé la somme d'une année, mais nous faisons 26 paiements dans l'année, donc nous avons 2 paiements additionnels pour la même année ce qui réduit considérablement la durée du terme et les intérêts.

Le tableau 6.1 est un tableau comparatif des frais d'intérêts selon les différents paiements (source Banque Royale) pour un prêt de 100 000\$.

Versement	Montant	Amortissement	Intérêt	Montant épargné
Mensuel	688,11\$	25,0	106 434\$	0\$
Bimensuel	344,06\$	24,9	105 688\$	746\$
Aux deux semaines	317,59\$	24,8	104 535\$	1 899\$
Hebdomadaire	158,79\$	24,7	104 211\$	2 223\$
2 semaines accéléré	344,06\$	20,6	83 914\$	22 520\$
Hebdomadaire accéléré	172,03\$	20,5	83 662\$	22 722\$

Tableau 6.1: Tableau comparatif des frais d'intérêts selon les différents paiements

Nous voyons donc qu'avec une augmentation de 26,47\$ à chaque deux semaines (de 317,59\$ pour le paiement aux 2 semaines à 344,06\$ pour le paiement aux 2 semaines accéléré) nous économisons 20 621\$ d'intérêts, soit 22 520\$ - 1 899\$. ce qui est considérable. Le type de paiement dit accéléré semble cependant être une méthode de marketing, car nous constatons qu'un prêt ordinaire de 20 ans payé aux deux semaines équivaut, à quelques dollars près, à un prêt sur 25 ans calculé en paiements accélérés aux deux semaines. C'est pourquoi certaines banques n'offrent pas les types de paiements dits «accélérés».

6.6 Privilège de pré-paiement sans pénalités

Les prêts fermés sont, par définition, non remboursable avant la fin du terme. Ainsi, si un emprunteur a un soudain surplus d'argent, il ne peut la mettre sur son hypothèque sans en subir des pénalités. Les institutions financières offrent cependant des options aux clients afin de pouvoir modifier leurs paiements, sans en payer les pénalités.

Trois options existent :

- Doubler le paiement mensuel ;
- Augmenter le paiement mensuel d'un certain pourcentage jusqu'à la fin du terme ;
- Faire un paiement d'un coup d'un certain maximum par rapport au prêt initial.

Doubler un paiement consiste à payer plus que la somme prévue pour un paiement mensuel, habituellement le double comme l'indique son nom, mais le paiement supplémentaire peut parfois être inférieur à cette somme.

L'augmentation mensuelle des paiements consiste en une hausse des versements mensuels, souvent de 10 ou 15% dépendant des banques, et cette augmentation est applicable une fois l'an. Cette option signifie que, par exemple pour un paiement de 750\$ par mois, nous pourrons augmenter ce paiement de 112,50\$ (pour une institution offrant une possibilité de 15% d'augmentation), et ainsi établir les nouveaux paiements

mensuels à 862,50\$, et ce pour le reste du terme. Cette opération est renouvelable à tous les ans.

Le paiement d'un coup est une option permettant à un client, une fois l'an, de payer jusqu'à 10 ou 15% de son prêt total sans frais.

Les options de pré-paiement anticipé sans frais sont disponibles dans toutes les institutions financières pour les prêts fermés fixes. Étant donné que les caractéristiques qui différencient les institutions financières sur ce point tournent autour du pourcentage permis et que cette différence n'a que peu d'intérêt d'une banque à une autre, nous n'utiliseront pas cette option dans la détermination de la meilleure banque.

6.7 Assurances vie/invalidité hypothécaire

Toutes les institutions financières offrent l'assurance hypothécaire. Il s'agit d'une assurance qui paye le solde restant de la maison en cas du décès de l'emprunteur, et qui peut payer jusqu'à environ 3 000\$ par mois en cas d'invalidité. Cette assurance est ajoutée au paiement mensuel à payer à chaque mois, et chaque banque a sa propre formule pour calculer ce taux. Le prix de l'assurance invalidité est une fonction directe basée sur les paiements mensuels, et l'assurance vie est calculée en fonction du montant total de l'hypothèque. Ces 2 assurances sont aussi fonctions de l'âge du client, mais il n'y a pas d'examen médical à passer contrairement aux assurances-vie traditionnelles. Seule l'âge compte. Nous détaillerons ce mode de calcul plus en détails dans les sections suivantes.

6.8 Ratio d'endettement

Chaque institution financière a son propre ratio pour calculer l'endettement d'un client. Le ratio est la proportion du salaire brut du demandeur qui peut être utilisée pour payer toutes ses dettes à chaque mois. Par exemple, une banque peut accepter 40%, c'est à dire que pour un salaire de 50 000\$, le demandeur pourra consacrer 20 000\$ par année à ses dettes, soit 1 667\$ par mois. Ce 1 667\$ contiendra le paiement de l'hypothèque, le paiement des taxes foncières, de l'électricité et du chauffage de la résidence, ainsi que des dettes de carte de crédit, des voitures et autres achats à payer chaque mois, ainsi que les marges de crédit. Pour calculer les dettes de carte de crédit et de marge de crédit mensuelle, la méthode utilisée est habituellement de prendre un pourcentage du total de crédit disponible sur la carte. Par exemple, pour une limite de 2 000\$ sur une carte de crédit, nous considérerons que le client dépensera environ 5% de son 2 000\$ par mois, soit 100\$ par mois. Pour une marge de crédit, le principe est le même, sauf que nous utilisons 10% (selon les Caisses Populaires Desjardins). Chaque banque a un ratio différent, tournant souvent autour de 40%, mais pouvant être de 38% ou de 45% dépendamment des exigences de l'institution financière dans son choix de client. Ainsi, un client qui est refusé à une banque car ses dettes sont trop élevées, pourrait être accepté à une autre, avec le même ratio de dettes.

6.9 Calcul d'hypothèque

Le calcul d'hypothèque est différent selon le pays où nous nous trouvons. La formule pour calculer une hypothèque aux États-Unis est différente de celle du Canada. Cela dépend des différentes lois en vigueur pour les prêts des différents pays. Certains pays peuvent même utiliser plusieurs formules de calcul d'hypothèque. Dans le cadre de ce mémoire, nous nous concentrerons sur le calcul d'hypothèque canadien.

Premièrement, la capacité de remboursement de l'emprunteur doit être calculé à l'aide du ratio d'endettement. Nous pouvons utiliser la formule suivante pour exécuter ce calcul.

$$\text{Capacité} = \left(\frac{\text{Salaire} \times \text{Ratio}}{12} \right) - (\text{Dépenses})$$

où :

- Salaire : le salaire annuel de l'emprunteur,
- Ratio : le ratio autorisé par l'institution financière (i.e. 0.40 pour 40%),
- Dépenses : la somme des dépenses mensuels de l'emprunteur après l'achat d'une maison (taxes foncières, électricité, chauffage, téléphone, carte de crédit, voitures, marges de crédit et autres achats mensuels).

Ensuite, nous calculons le montant d'argent que l'emprunteur doit payer mensuellement avec le type d'hypothèque qu'il a choisi. Ce montant est calculé par la formule suivante :

$$\text{Montant} = \text{Capital} \times \left[\frac{\left(1 + \frac{\text{taux}}{200}\right)^{\frac{1}{6}} - 1}{1 - \left(\left(1 + \frac{\text{taux}}{200}\right)^{\frac{1}{6}}\right)^{-nbMois}} \right]$$

où :

- Capital : la valeur du montant emprunté ;
- taux : le taux d'intérêt en vigueur ;
- nbMois : la durée de l'hypothèque en mois.

Pour que le prêt hypothécaire soit consenti, il faut que la capacité de remboursement soit supérieur au montant mensuel de l'hypothèque.

Chapitre 7

Application de NetSA au domaine financier

Dans ce chapitre, nous allons voir comment intégrer les systèmes multiagents et les enchères aux systèmes financiers. Premièrement, l'utilisateur voulant accéder à NetSA doit le faire via un navigateur Internet (Netscape, Internet Explorer, HotJava, etc.). Pour le moment, NetSA est accessible à l'adresse :

<http://damas.ift.ulaval.ca/~netsa/useragent.html>

Nous trouvons à cette adresse la page d'accueil de NetSA où l'utilisateur choisit le type d'opération qu'il veut faire. Pour le moment, seul le courtage de prêt hypothécaire est actif (voir Figure 7.1).

La deuxième page, fait accéder l'utilisateur à un formulaire. Ce formulaire est utilisé pour capter ses informations personnelles (nom, prénom, numéro d'assurance sociale, adresse, âge, etc. comme en Figure 7.2). Ces informations sont utilisées pour vérifier son identité, sa solvabilité et son dossier de crédit.

Le formulaire suivant demande à l'utilisateur l'informations sur la maison qu'il souhaite achetée (cf. Figure 7.3). Il doit y inscrire l'adresse de la maison pour vérifier son existence et sa valeur. Le prix de vente de la nouvelle maison est également

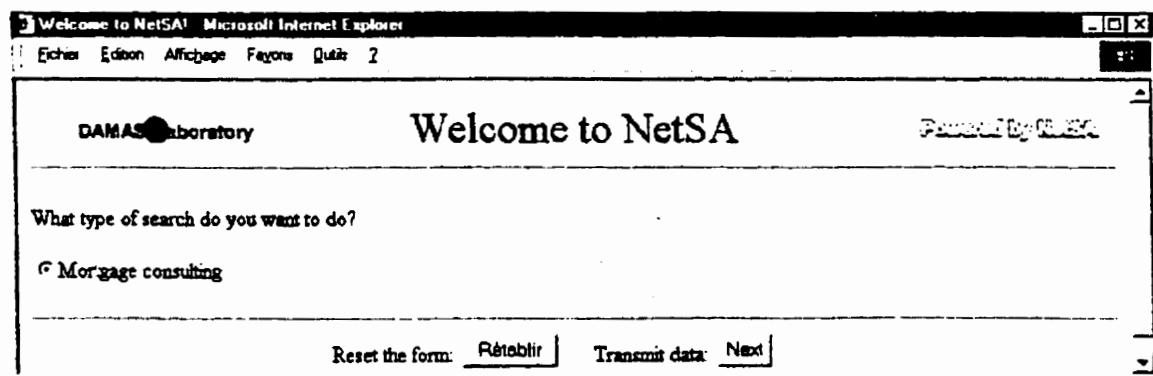


Figure 7.1: Page d'accueil de NetSA.

The screenshot shows a web page titled "Ask mortgage consulting" with a sub-header "Personal informations". On the left, there is a logo for "DAMAS Laboratory". The right side features a large graphic of a house. The form is divided into two sections: "Applicant" and "Second applicant (if applicable)".

Applicant		Second applicant (if applicable)
First name:	Marc	
Last name:	Cote	
Social Insurance Number:	123 456 789	
Address:		
City:		
Zip code:		
Age:	24	

At the bottom, there are buttons for "Reset the form: Rétablir" and "Transmit data: Next".

Figure 7.2: Informations personnelles concernant l'utilisateur.

demandé pour le comparer avec l'estimation de la maison afin d'éviter les arnaques venant des vendeurs. Finalement, on recueille les dépenses normale (électricité, chauffage, téléphone, etc.) encouru par l'achat de la nouvelle maison. Ces dépenses doivent être soustraite de l'avoir mensuel de l'utilisateur.

Les deux formulaires suivants présentés en Figure 7.4 et Figure 7.5 informent le système multiagent des dépenses et les gains de l'utilisateur. À l'aide de ces informations, NetSA est en mesure de calculer combien l'utilisateur peut payer chaque mois pour la maison qu'il désire acheter. Ceci indiquera s'il y a un risque de problèmes financiers pour l'utilisateur en ajoutant cette charge monétaire supplémentaire.

Le dernier formulaire (voir Figure 7.6), mais non le moins important, est un formulaire pour aider l'utilisateur à choisir le type de prêt hypothécaire qui lui convient le mieux. Le formulaire débute par un petit sondage qui, selon les réponses données, conseillera l'utilisateur sur le type de prêt hypothécaire le plus rentable pour lui.

S'il le désire, l'utilisateur peut choisir lui-même le type de prêt qu'il veut. Il indique le montant de son prêt, la durée totale du prêt, le type désiré, la durée du contrat, le dépôt initial et les assurances désirées.

Les informations recueillies par les formulaires sont par la suite acheminées à l'agent utilisateur (Figure 7.7). Ce dernier construit un message KQML contenant l'information venant des formulaires et le transmet à l'agent superviseur en charge de la vente aux enchères.

Ask Mortgage Consulting - New House Information - Microsoft Internet Explorer

Ficher Edition Affichage Favors Quits ?

DAMAS Laboratory Ask mortgage consulting ADDITIONAL INFORMATION

New house informations

Note: When you enter your amounts, use "round" numbers, without spaces or commas.



Informations

Type of house:	Bungalow
Address:	710 Horizon
City:	Ste-Foy
Age:	0 year(s)
Cost of the property:	30000 \$
Monthly heating costs (0 if unknown):	0 \$
Property tax monthly (0 if unknown):	0 \$
Co-property fees (if applicable):	0 \$

Reset the form Rétabrir Transmit data: [Next](#)

Figure 7.3: Informations sur la maison.

Ask Mortgage Consulting Financial Obligations Microsoft Internet Explorer

Échec Édition Affichage Foyers Quitté 2

DAMAS laboratory Ask mortgage consulting Personnalized Home

Financial obligations

Note: When you enter your amounts, use "round" numbers, without spaces or commas

What you owe	Monthly amount
Total other mortgage(s)	0 \$
Total car loan(s)	0 \$
Total personal loan(s)	0 \$

Credit	Credit limit
Personal line(s) of credit	0 \$ 0 \$ 0 \$
Credit card(s)	0 \$ 0 \$
Other obligations (ex: child support payments or alimony)	0 \$

Reset the form Rétabir Transmit data:

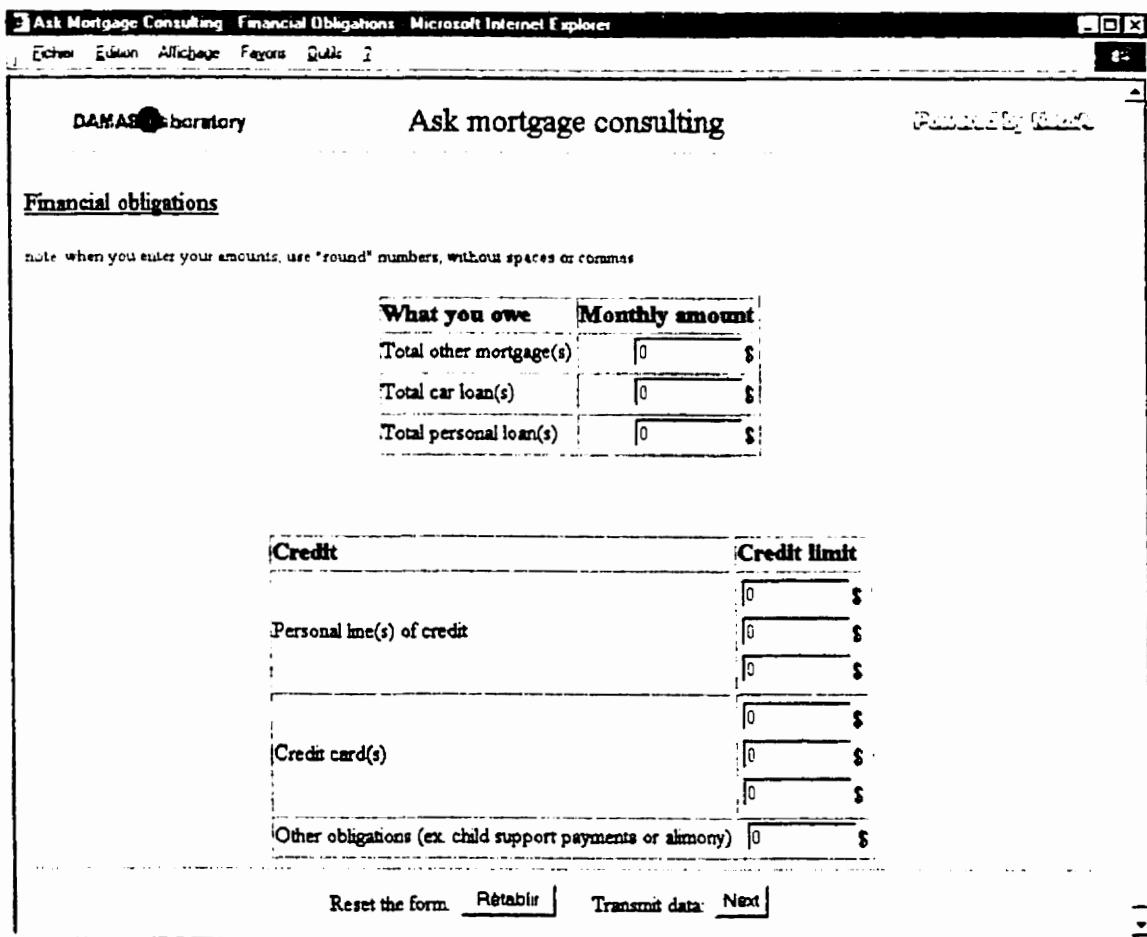


Figure 7.4: Dépenses de l'utilisateur.

The screenshot shows a Microsoft Internet Explorer window with the title bar "Ask Mortgage Consulting - Incomes Microsoft Internet Explorer". The menu bar includes "Fichier", "Edition", "Affichage", "Favoris", and "Aide". The main content area has a header "Ask mortgage consulting" and a sub-header "Revenus de l'utilisateur". A note below the sub-header says: "note: when you enter your amounts, use "round" numbers, without spaces or commas." There is a cartoon illustration of a hand holding a credit card. Below it is a table with five rows and one column for entering amounts. The table has a header row with "Amount" and a sub-header row with "\$". The rows are: "Annual Income (without taxes)" with value "30000 \$"; "Alimony, child support payments or others (monthly)" with value "0 \$"; "monthly interest/dividend" with value "0 \$"; "Location (monthly)" with value "0 \$"; and "Other income source (monthly)" with value "0 \$". At the bottom of the form are buttons: "Reset the form" (French: "Rétablir"), "Transmit data" (French: "Transmettre les données"), and "Next" (French: "Suivant").

Amount	
\$	
Annual Income (without taxes)	30000
Alimony, child support payments or others (monthly)	0
monthly interest/dividend	0
Location (monthly)	0
Other income source (monthly)	0

Figure 7.5: Revenus de l'utilisateur.

Le processus de vente aux enchères peut alors commencer. Nous avons opté pour une enchère de type *Meilleur Prix* (voir chapitre 5 à la section 5.3) dans la mesure où cette enchère nécessite moins de communication donc moins de temps, ce qui est très important si nous voulons un système rapide.

Le superviseur de l'enchère, comme celui présenté en Figure 7.8, demande donc à chacun des superviseurs des institutions financières de faire des offres à partir des informations provenant de l'agent utilisateur. Chacun de ces superviseurs consultent les agents ressources (voir Figure 7.9) qui lui sont attribués afin d'obtenir les informations essentielles à l'élaboration d'un bon prêt hypothécaire. Ces informations peuvent être : (1) le dossier de crédit, (2) l'estimation de la maison, (3) les promotions en vigueur, (4) le coût des assurances, (5) les taux d'intérêt, etc.

Le superviseur en charge de l'enchère comptabilise ensuite toutes les offres en analysant le coût mensuel, les économies réalisées et le coût total du prêt. Le superviseur annonce à l'utilisateur par l'intermédiaire de l'agent utilisateur, le gagnant ainsi que la position des autres participants comme montré en Figure 7.10. L'utilisateur reste toujours libre de choisir l'institution financière avec laquelle il veut faire affaire.

http://caligula.ilt.uwaterloo.ca:8080/servlet/ouervict - Microsoft Internet Explorer

Fichier Édition Affichage Foyers Dossiers 2

DAMAS Laboratory Ask mortgage consulting

Your mortgage selection

These preferences will help us to suggest you the best type of loan being appropriate for your needs. If you already know what type of loan you want, go immediately to the second table.

To pay your house, will you have :	<ul style="list-style-type: none"> <input checked="" type="radio"/> to cut your expenses <input type="radio"/> to reorganize your budget <input type="radio"/> let your budget as it is
During your term, will you accept :	<ul style="list-style-type: none"> <input checked="" type="radio"/> no variation in your monthly payments <input type="radio"/> slight variations of your payments <input type="radio"/> Large fluctuations of your payments according to interest rates
For the duration of your term, do you believe that the interest rates will :	<ul style="list-style-type: none"> <input checked="" type="radio"/> Increase <input type="radio"/> To remain stable / I do not know it <input type="radio"/> Decrease
Do you believe that you will have to make repayments before due date	<ul style="list-style-type: none"> <input checked="" type="radio"/> Lower than 15% of the total of your loan <input type="radio"/> Superior to 15% of your loan <input type="radio"/> I will not make repayments before due date

Note: When you enter your amounts, use "round" numbers, without spaces or commas

		Informations
Amount to borrow	30000 \$	
Amortization period:	25 years	
Type of rate:	Fixed Rate	
Type of mortgage:	Open	
Term:	0.5 year(s)	
Down payment:	5000 \$	
Do you want a mortgage life-insurance?	<input checked="" type="radio"/> yes <input type="radio"/> no	
Do you want a mortgage disablement-insurance?	<input checked="" type="radio"/> yes <input type="radio"/> no	

Reset the form Rétablir Transmit data Finish

Figure 7.6: Choix du type de prêt hypothécaire.

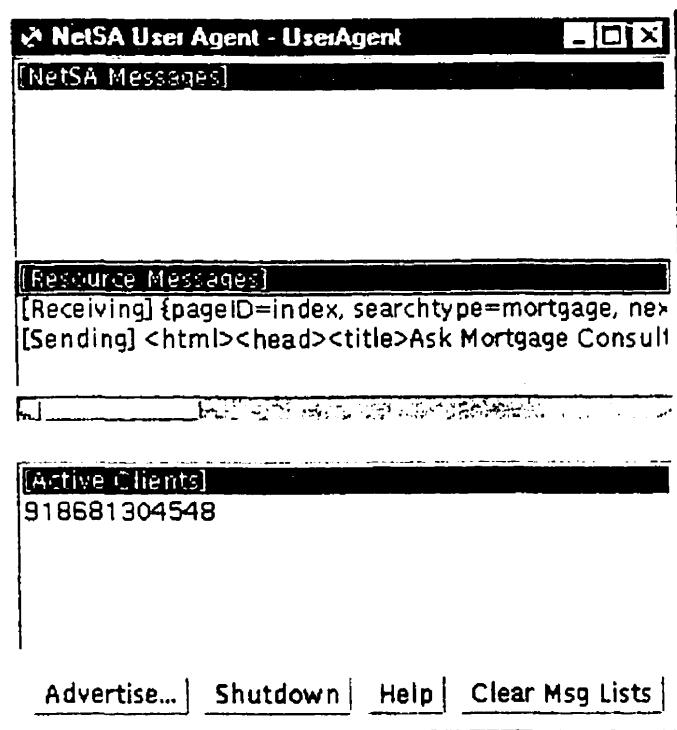


Figure 7.7: Agent utilisateur.

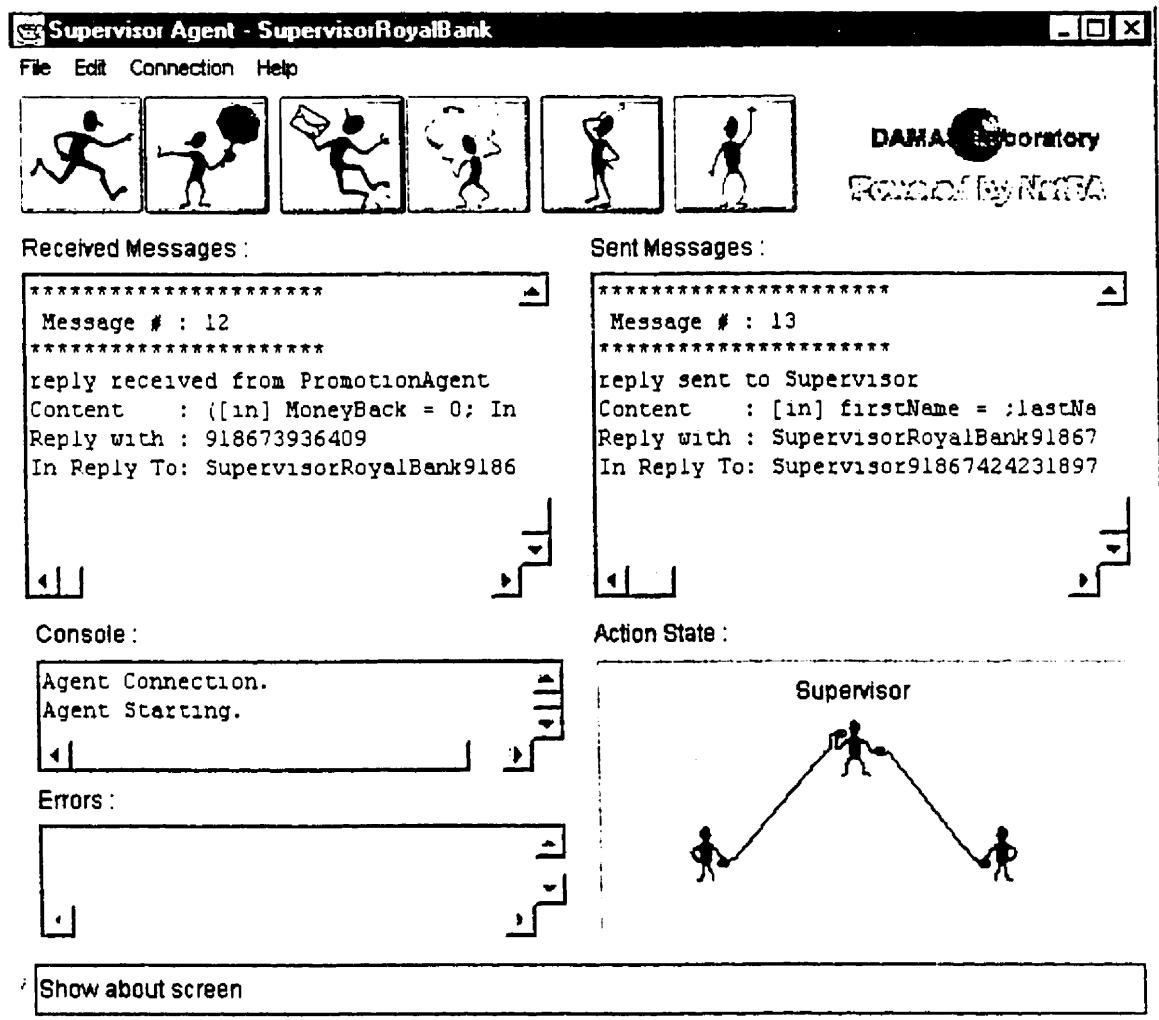


Figure 7.8: Agent superviseur.

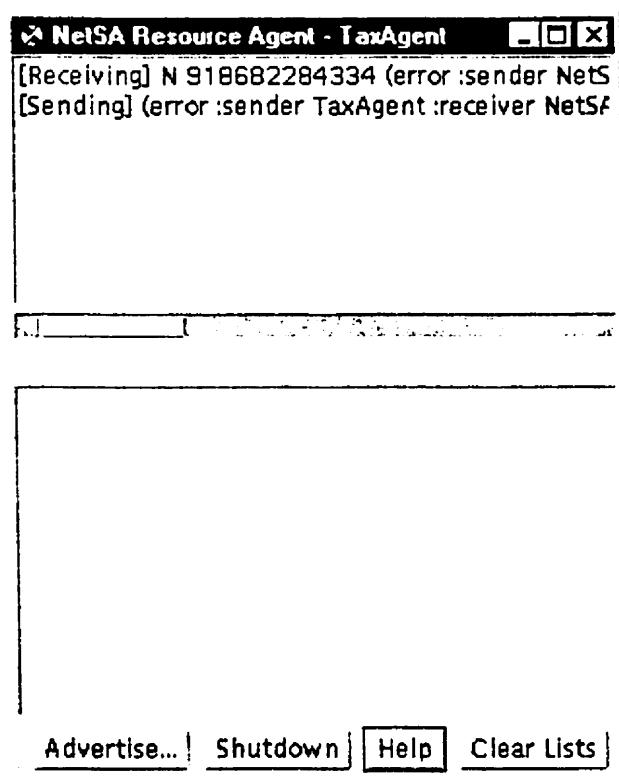


Figure 7.9: Agent ressource.

Your Mortgage Results - Microsoft Internet Explorer
 Fichier Édition Affichage Favoris Quits 2

Your Mortgage Results

Your mortgage is accepted conditionally. Please print this page and contact our bank for a definitive acceptance of your mortgage. The following lines describe the mortgage you asking for.

Personal Informations

Name	Marc Cote
Address	710 Horizon
City	Ste Foy
Estimation	30000 0 \$
Sold Price	30000 \$
Amortization	25 0 year(s)
Type	Fixed Rate Open
Term	0.5 year(s)

Best

	Royal Bank	National Bank	Montreal Bank
Interest Rate	7.35 %	7.44 %	7.35 %
Normal Monthly Payment	180.53 \$	182.10 \$	180.53 \$
Monthly Insurance Cost	2.66 \$	2.56 \$	2.72 \$
Sub-total	183.20 \$	184.66 \$	183.26 \$
Monthly Promotion Saving :	3.75 \$	2.5 \$	1.25 \$
Monthly Payment :	179.45 \$	182.16 \$	182.01 \$

[For more informations please contact us](#)

Figure 7.10: Résultats de l'enchère *meilleur prix*.

Chapitre 8

Conclusion

Ce mémoire a présenté NetSA un architecture multiagent dédiée aux environnement riches en informations. Nous avons premièrement vu différentes architecture d'agents et de systèmes multiagents suivie de quelques précisions sur les communications. Le chapitre 4 nous a ensuite expliqué l'architecture NetSA et ses différentes composantes en portant une attention particulière à l'agent superviseur. Nous avons vu également les enchères et le fonctionnement de l'hypothèque canadienne. Finalement, nous avons validé notre architecture NetSA en développant un prototype d'architecture multiagent capable d'aider les utilisateurs dans le cadre de l'hypothèque canadienne.

Dans ce qui suit nous rappelons les objectifs qui ont été fixé dans le chapitre d'introduction et nous commenterons chacun de ceux-ci.

Objectif 1 : étudier les agents et les systèmes multiagents. Pour réaliser cet objectif, nous avons présenté en détail dans le chapitre 2.5 explique les concepts d'agents et de systèmes multiagents. Ces concepts aident les lecteurs à mieux comprendre les agents et les système multiagent ce qui est essentiel à la bonne compréhension de ce mémoire.

Objectif 2 : étudier les architectures des systèmes multiagents. Le chapitre 2.5 présente également les architectures d'agents et de systèmes multiagents. Dans ce chapitre nous avons particulièrement détaillé un type d'architecture cognitive ainsi que plusieurs architectures dédiés à la recherche d'information. Mentionnons InteRRaP pour les agents et InfoSleuth et Retsina pour les systèmes multiagents.

Objectif 3 : créer une architecture de système multiagent. NetSa est l'architecture développée dans le cadre de ce mémoire. C'est une architecture ouverte à trois couches spécialisées dans le recherche d'informations. C'est dans le chapitre 4 que nous avons détaillé le fonctionnement de cette architecture.

Objectif 4 : étudier le principe des prêts hypothécaires. Pour réaliser un système de courtage de prêts hypothécaires, il fallait bien connaître le domaine. Dans le

chapitre 6 nous avons expliqué en détails le fonctionnement du calcul et de l'acceptation des prêts hypothécaires canadiens.

Objectif 5 : étudier les différents types d'enchères. Le courtage que nous avons fait auprès des institutions financières nécessitait l'élaboration d'un protocole de vente concret et efficace. Nous avons choisi d'utiliser les ventes aux enchères pour instaurer un environnement compétitif entre les banques.

Objectif 6 : créer un système multiagent appliquéd au domaine financier. La réunion de tous les objectifs précédents nous a conduit vers notre objectif ultime qui était de créer un système multiagent appliquéd au domaine financier. Nous avons donc réalisé un système multiagent qui fait le courtage de prêts hypothécaires à l'aide d'un algorithme d'enchère.

8.1 Résultats de NetSA

Dans une première phase de NetSA, les plans s'exécutaient de manière séquentielle. Ce type de traitement comportait un énorme désavantage dans la mesure où l'agent ne pouvait que communiquer avec un seul agent à la fois ce qui rendait le traitement extrêmement lent. Dans la nouvelle implémentation de NetSA, des plans ont la possibilité de s'exécuter en concurrence.. Lorsque nous voulons plusieurs informations situées sur plusieurs agents, nous pouvons utiliser un plan pour chaque agent qui détient l'information désirée. Ceci nous donne une augmentation des performances de calcul puisque nous éliminons une grande partie de la latence entre les communications entre agents.

Nous avons fait des comparaisons entre un calcul de prêt hypothécaire effectué avec une exécution séquentielle et une exécution concurrente des plans. Le graphique de la Figure 8.1 montre les résultats obtenus.

Comme nous pouvons constater, la distribution concurrente de messages est supérieure à la distribution séquentielle. Ce résultat s'explique dans la réduction de l'attente inutile provoquée par la distribution séquentielle. Au lieu d'envoyer un message et d'attendre sa réponse, il est évidemment préférable d'envoyer tous les messages en même temps. Ceci surcharge un peu plus le réseau mais la baisse de performance qui en résulte est moindre que le temps d'attente provoqué par la distribution séquentielle des messages.

Nous pouvons également remarquer que le temps de réponse est relativement lent même lorsque les plans sont exécutés de manière concurrentes (environ 21 secondes). Ce résultat est attribuable à l'interpréteur Java qui n'est évidemment pas aussi rapide que le pourrait l'être du code spécialement conçu pour une machine et un système d'exploitation spécifique. La communication par réseau est aussi un facteur de perte de performance. Lorsque le réseau est très achalandé, nous pouvons remarquer une baisse de la performance de NetSA.

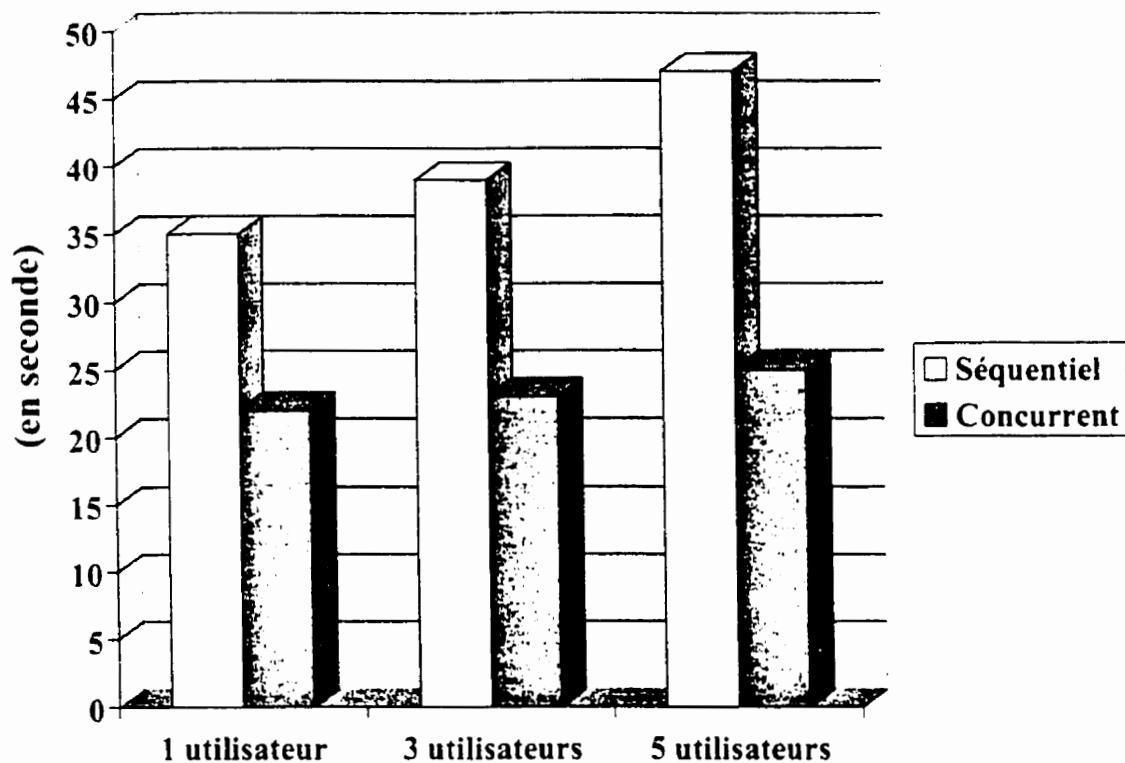


Figure 8.1: Comparaison entre une exécution séquentielle et concurrente des plans.

8.2 Développements à venir

Bien que NetSA soit opérationnelle, il y a toujours une place pour l'amélioration. Dans ce cadre, nous pouvons inclure deux nouveaux types d'agent soit le serveur d'ontologie et l'agent d'extraction de connaissances.

8.2.1 Serveur d'ontologie

L'ontologie est la manière de définir le sens des mots. Lorsque deux agent discutent ensemble, ils doivent avoir la même ontologie pour se comprendre. Par exemple, si un agent demande à un autre : «connais-tu Java ?». Si agent en question ne connaît pas l'ontologie, il peut penser que Java est soit un langage de programmation, une danse, une île ou un café. Pour éviter ce genre d'interprétations ou de confusions, il faut instaurer une ontologie claire au sein du système multiagent.

Le rôle du serveur d'ontologie est de fournir aux nouveaux agents entrant sur le système la définition de l'ontologie utilisée. Ainsi tout agent voulant entrer dans le système est tenu de connaître l'ontologie à laquelle il se réfère. Nous évitons par se fait les communication inutile pouvant être provoqué par des agents inconnus. Le serveur d'ontologie fait partie de la couche centrale de NetSA.

8.2.2 Agent d'extraction de connaissances

La complexité de l'information stockée dans les grandes bases de données est immense. Nous avons souvent différentes données qui, mises en commun, peuvent nous faire découvrir de nouvelles données. Par exemple, nous pourrions découvrir quels sont les symptômes d'une maladie difficile à détecter.

Cet agent, par des procédés d'extraction de connaissances (data-mining) est en mesure de découvrir des règles qui peuvent aboutir à la découverte de nouvelle données. L'agent d'extraction de connaissances agira donc comme un agent ressource spécialisé.

Bibliographie

- [AC87] P. Agre and D. Chapman. Pengi : An implementation of a theory of activity. In *Proceedings AAAI-87*, San Mateo, CA, 1987. Morgan Kaufmann.
- [AGST75] R. A. Amsler, E. M. Greenawalt, J. Slocum, and M. Tyson. *LISP Reference Manual CDC - 6000*. University of Texas at Austin. Computation Center. CCUM 2. Austin, December 1975.
- [Ark92] R. C. Arkin. AuRA : A hybrid reactive/hierarchical robot architecture. In *Proc. of the IEEE Int. Conf. on Robotics and Automation, Workshop on Architecture for Intelligent Control Systems*, pages 9–13. 1992.
- [Auc98] Auction universe. 1998. <http://www.auctions.com>.
- [BBB+96] R. Bayardo, W. Bohrer, R. Brice, A. Cichocki, G. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. Semantic integration of information in open and dynamic environments. Technical Report MCC-INSL-088-96, MCC, October 1996.
- [BFG+97] R. P. Bonasso, R. J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. to appear in *Journal of Experimental and Theoretical Artificial Intelligence*. January 1997.
- [BIP88] M. Bratman, D. Israel, and M. Pollack. Plan and resource-bounded practical reasoning. *Computational Intelligence*, 4 :349–355, 1988.
- [BLFF96] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol - http/1.0. Request for comment : 1945, May 1996.
- [Bro89] R. A. Brooks. A robot that walks : emergent behaviours from a carefully evolved network. volume 2, pages 692–696, 1989.
- [CDR99] Stanford University CDR. Jatlite, 1999. <http://java.stanford.edu>.
- [DL90] K. Decker and V. Lesser. Extending the partial global planning framework for cooperative distributed problem solving network control. Technical Report UM-CS-1990-081, University of Massachusetts, Amherst, Computer Science, October, 1990.
- [Fer92] I. A. Ferguson. Touring Machines : an architecture for dynamic, rational, mobile agents. Technical Report No. 273, U. of Cambridge, November 1992.

- [FFMM94] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an agent communication language. In ACM Press, editor, *Proceedings of the Third International Conference on Information and Knowledge Management*, November 1994.
- [Fir95] R. J. Firby. The RAP language manual. Animate Agent Project Working Note AAP-6, University of Chicago, March 1995.
- [FJ91] J. Ferber and E. Jacopin. The framework of ECO-problem solving. In Y. Demazeau and J.-P. Müller, editors, *Decentralized A.I.*, volume 2, pages 181–193. North-Holland, 1991.
- [FMP95] K. Fischer, J. P. Müller, and M. Pischel. Cooperative transportation scheduling : an application domain for dai. Technical Report Research Report RR-95-01, DFKI GmbH, Saarbrücken, 1995.
- [FN71] R. E. Fikes and N. J. Nilsson. STRIPS : A new approach to the application of theorem proving to problem solving. In D. C. Cooper, editor, *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, pages 608–620, London, UK, September 1971. William Kaufmann.
- [FR93a] R. A. Feldman and M. Rajnosh. Auctions : A sampling of techniques. *Finance and Development*, pages 32–35, September 1993.
- [FR93b] R. A. Feldman and M. Rajnosh. Auctions theory and applications. *IMF Staff Papers*, pages 485–511, September 1993.
- [Gea92] M. R. Genesereth and R. E. Fikes et al. Knowledge interchange format version 3 reference manual. Technical Report Logic-92-1, Stanford University Logic Group, 1992.
- [GM95] J. Gosling and H. McGilton. The Java(tm) language environment : A white paper. Technical report, Sun Microsystems, 1995.
- [HC92] D. M. Hart and P. R. Cohen. Predicting and explaining success and task duration in the phoenix planner. In James Hendler, editor, *Artificial Intelligence Planning Systems : Proceedings of the First International Conference (AIPS 92)*, pages 106–115, College Park, Maryland, USA, June 1992. Morgan Kaufmann.
- [HC96] G. Hamilton and R. Cattell. Jdbc : A Java sql api. 1996. <http://splash.javasoft.com/jdbc/jdbc.sp>, JavaSoft.
- [HRWL83] F. Hayes-Roth, D. Waterman, and D. Lenat. Building expert systems. In Addison-Wesley, editor, *Building Expert Systems*, pages 139–149, New York, 1983.
- [Jan95] P. C. Janca. Pragmatic application of information agents : Bis strategic decisions. Technical report, Norwell, United States, May 1995.
- [Jen92] N. R. Jennings. Using GRATE to build cooperating agents for industrial control. In *Proc. IFAC/IFIP/IMACS Int. Sym. on Artificial Intelligence in Real Time Control, Delft, The Netherlands*, pages 691–696, 1992.

- [JSW98] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. In *Autonomous Agents and Multi-Agent Systems*, pages 7–38, Boston, 1998. Kluwer Academic Publishers.
- [KQM97] New KQML admin msgs. 1997. <http://cdr.stanford.edu/ProcessLink/kqml-proposed.htm>.
- [LR92] R. Levy and J. S. Rosenschein. A game theoretic approach to the pursuit problem. In *Working Papers of the 11th International Workshop on Distributed Artificial Intelligence*, pages 195–213, February 1992.
- [Mae89] P. Maes. The dynamics of action selection. In *Proceedings of IJCAI-89*, volume 2, pages 991–998, 1989.
- [MCD96] B. Moulin and B. Chaib-Draa. An overview of distributed artificial intelligence. *Foundations of Distributed Artificial Intelligence*, O'Hare, G. and Jennings, N. (eds), Wiley Inter-Science, pages 3–55, 1996.
- [Mic98] Sun Microsystem. The Java servlet API. 1998. <http://www.micro.rmc.ca:8080/system/doc/servlets/api.html>.
- [MP93] J. P. Müller and M. Pischel. The agent architecture inteRRaP : Concept and application. Research Report RR-93-26. Deutsches Forschungszentrum für künstliche Intelligenz, Kaiserslautern, Germany, 1993.
- [Net97] Netscape. *JavaScript Reference*. Netscape Communications Corporation, 1997.
- [Nwa96] H. S. Nwana. Software agents : An overview. *Knowledge Engineering Review*, 11(3) :205–244, October/November 1996.
- [RG91] A. S. Rao and M. P. Georgeff. Modeling agents within a BDI-architecture. In R. Fikes and E. Sandewall, editors. *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484. Cambridge, Mass., April 1991. Morgan Kaufmann publishers Inc. : San Mateo, CA, USA.
- [Ril91] G. Riley. Clips : An expert system building tool. In *Proceeding of the Technology 2001 Conference*, San Jose, CA, December 1991.
- [RK86] S. Rosenschein and L. Kaelbling. Synthesis of digital machine with provable epistemic properties. In Joseph Halpern, editor, *RAK86*, pages 83–98. Monterey, March 1986.
- [SCS94] D. C. Smith, A. Cypher, and J. Spohrer. Programming agent without a programming language. *Communications of the ACM*, 37(7) :55–67, 1994.
- [Sel94] Y. Selker. A teaching agent that learns. *Communications of the ACM*, 37(7) :92–99, 1994.
- [Sho91] Y. Shoham. AGENT0 : A simple agent language and its interpreter. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, volume 2, pages 704–709, Anaheim, California, USA, July 1991. AAAI Press/MIT Press.

- [Smi77] S. G. Smith. The contract net : A formalism for the control of distributed problem solving. In *The Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, Cambridge, MA, 1977.
- [SPW⁺96] K. Sycara, A. Pannu, M. Williamson, D. Zeng, and K. Decker. Distributed intelligent agents. *IEEE Expert*, pages 36–46, December 1996.
- [Ste94] L. Steels. Equilibrium analysis of behaviour systems. In *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI'94)*, pages 714–718, Amsterdam, NL, 1994.
- [Syc87] K. P. Sycara. *Resolving Adversarial Conflicts : An approach integrating case-based and analytic methods*. PhD thesis, Georgia Institute of Technology, Atlanta, Georgia, June 1987.
- [VD95] J. M. Vidal and E. H. Durfee. Task planning agents in the UMDL. In Tim Finin and James Mayfield, editors, *Proceedings of the CIKM '95 Workshop on Intelligent Information Agents*, Baltimore, Maryland, 1995.
- [Vic61] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, Vol. 16 :8–37, March 1961.
- [Wat85] D. Waterman. *A Guide to Expert Systems*. Addison-Wesley, New York, 1985.
- [WCH⁺93] D. Woelk, P. Cannata, M. Huhns, W. M. Shen, and C. Tomlinson. Using carnot for enterprise information integration (project synopsis). In *Parallel and Distributed Information Systems (PDIS '93)*, pages 133–137, Los Alamitos, Ca., USA, January 1993. IEEE Computer Society Press.
- [Wei95] T. Weiser. Akb : Assertion knowledge base. Technical report, Internal Report, 1995.
- [WJ95] M. J. Wooldridge and N. R. Jennings. Intelligent agents : Theory and practice. *Knowledge Engineering Review*, 10(2) :115–152, 1995.
- [Zan91] C. Zaniolo. The logical data language (ldl) : An integrated approach to logic and databases. Technical Report Technical Report STP-LD-328-91, MCC, 1991.

Annexe A

Programmation Orientée Plan pour les Agents (POPA)

Lorsque nous parlons de plans, nous voyons des schémas, des directives. Pour un agent, un plan va un peu dans ce sens. Nous avons une structure pour nous permettre de bien suivre le déroulement des plans. Nous avons aussi un langage pour diriger les actions des plans. Cette structure et ce langage forment un nouveau pseudo-paradigme nommé POPA (Programmation Orientée Plan pour les Agents).

Dans POPA, un plan est formé d'un ou d'une succession de sous-plan. Les sous-plans sont constitués d'une déclaration et d'actions.

La déclaration d'un sous-plan débute toujours par la primitive SubPlan suivit du nom du sous-plan. Nous terminons un sous-plan toujours par la primitive End SubPlan.

La zone destinée aux actions suit toujours la déclaration SubPlan. Les actions peuvent être des calculs mathématiques, des assignations, des appels de fonctions prédéfinies, etc.

Les expressions dans le langage de POPA sont toutes composées d'opérateurs binaires qu'il faut délimiter, lorsqu'il y a risque de confusion, par des parenthèses. Les opérateurs sont :

- + ⇒addition,
- - ⇒soustraction,
- * ⇒multiplication,
- / ⇒division,
- ^ ⇒puissance,
- = ⇒comparaison d'égalité,
- # ⇒comparaison d'inégalité,
- < ⇒comparaison d'infériorité,
- > ⇒comparaison de supériorité,
- : ⇒assignation,
- & ⇒et logique,
- | ⇒ou logique.

A.1 Variables

Pour un même plan, toutes les variables sont globales. Si un sous-plan utilise une variable, toutes les autres sous-plans ont accès à cette variable. POPA utilise un langage de programmation fortement typé avec un caractère spécial à la fin de chaque nom de variable. Nous y retrouvons trois types de données :

1. Chaîne de caractères,
2. Nombres.
3. Booléens.

A.1.1 Type de données

La chaîne de caractère est utilisée pour stocker un ou plusieurs caractères. Le nom d'une variable de ce type se termine toujours par un signe de dollar (\$). L'opérateur <<+>> peut être utilisé pour concaténer deux chaînes de caractères ou une chaîne de caractère et un nombre.

Le nombre est utilisé pour stocker un nombre avec une précision quelconque. Le nom d'une variable de ce type se termine toujours par un signe de pourcentage (%). Des opérateurs mathématiques tels que <<+, -, *, /, ^>> effectuent dans l'ordre l'addition, la soustraction, la multiplication, la division et la puissance entre deux nombres.

Le booléen est utilisé pour stocker une valeur vrai ou fausse comme dans la logique booléenne. Le nom d'une variable de ce type se termine toujours par un point d'interrogation (?).

Exemples

```
Chaîne$ : "Allô" (est une chaîne de caractère.)
Nombre% : 34 (est un nombre.)
Booléen? : true (est en booléen.)
```

A.1.2 Tableau

Dans POPA toutes les variables sont des tableaux de variables. Lorsque nous accédons à une variable, nous accédons à la valeur de l'indice 0 du tableau de variable. Pour accéder aux autres indices du tableau, nous utilisons un nombre ou une variable de type nombre entre crochets ([]) après le nom de la variable.

Exemples

```
Chaîne$ : "Allô"
Chaîne$[1] : "toi"
Chaîne$[2] : "mon"
Chaîne$[3] : "ami"
Le tableau Chaîne$ contient maintenant 4 valeurs : "Allô", "toi", "mon",
"ami".
```

A.2 Fonctions prédéfinies

Les fonctions prédéfinies sont des outils spécialisés pour la communication entre les agents, le traitement de tableaux, etc. POPA possède trois fonctions prédéfinies.

A.2.1 NbReply% : send{Dest\$, Perf\$, Context\$, InReplyTo\$, In\$, Out\$}

La fonction «`send`» est utilisée pour envoyer un message à un autre agent. Lorsqu'un message est envoyé, si le champ contenant l'identificateur de requête est vide alors l'agent poursuit l'exécution de la requête sans attendre de réponse. Cependant, dans le cas contraire le plan se met en veille jusqu'à réception de la réponse.

Lorsqu'un message est reçu par l'agent, les variables `requestAgent$` et `requestReplyWith$` sont initialisées. Ces variables contiennent respectivement le nom de l'agent requérant et la valeur de l'identificateur de requête.

- `NbReply%` : Nombre de réponses retournées.
- `Dest$` : Destinataire du message.
- `Perf$` : Acte de langage.
- `Context$` : Contexte de la requête.
- `InReplyTo$` : Identificateur de retour de la requête.
- `In$` : Paramètres [in] du contenu.
- `Out$` : Paramètres [out] du contenu.

Exemple

```
send{requestAgent$, "reply", "Netsa", requestReplyWith$, in$, out$}
```

A.2.2 TableSize% : size{VarName\$}

La fonction `size` retourne la grandeur de la variable tableau correspondant à la variable `VarName$`. Donc, c'est la valeur du plus grand indice de la variable tableau plus un.

- `TableSize%` : Grandeur de la variable tableau,
- `VarName$` : Nom de la variable tableau.

Exemple

```
Chaîne$[0] : "Allô"
Chaîne$[1] : "toi"
Chaîne$[2] : "mon"
Chaîne$[3] : "ami"
X% = size(Chaîne$)
X% contiendra la valeur 4.
```

A.2.3 NewText\$ / NewNumber% : format\$ / format%{Text\$ / Nombre%, Precision%}

La fonction `format` tronque un nombre ou un texte représentant un nombre avec `Precision%` chiffres après le point en retourne le tout dans une chaîne de caractère.

- `NewText$` : Nouvelle chaîne de caractère formé par le nombre tronqué,
- `NewText$` : Nouveau nombre formé par le nombre tronqué,
- `Text$` : Chaîne de caractère représentant un nombre à tronquer,
- `Nombre%` : Nombre à tronquer,
- `Precision%` : Nombre de chiffre après le point désiré.

Exemple

```
Chaîne$ : Format${12.3456, 2}
Chaîne$ contiendra la chaîne de caractère "12.34".
Nombre% : Format${12.3456, 2}
Nombre% contiendra la valeur 12.34.
```

A.2.4 print{Text\$}

La fonction `print` imprime sur la sortie standard (habituellement l'écran) du système le texte contenu dans `Text$`.

- `Text$` : Chaine de caractère à imprimer.

Exemple

```
print{"allô"}
Imprime le mot allô à l'écran.
```

A.3 Mots réservés

A.3.1 While{Condition ?} - EndWhile

La boucle `While` fait des itérations d'un segment de code tant que la condition `Condition ?` est vraie. Le segment de code débute sous l'instruction `While` et se termine avec l'instruction `EndWhile`.

- `Condition ?` : Expression booléenne permettant le rebouclage.

Exemple

```

X% = 0
While{x% < 10}
    x% : x% + 1
    print{x}
EndWhile

```

Imprime les nombres de 1 à 10.

A.3.2 If{Condition ?} - Else - EndIf

Le mot réservés **If** permet d'exécuter un segment de code si la condition **Condition ?** est vraie. Le segment de code débute sous l'instruction **If** et se termine avec l'instruction **Else** ou **EndIf**. Si l'instruction **Else** délimite le segment de code du **If** et si la condition **Condition ?** est fausse alors le segment de code sous l'instruction **Else** sera exécuté. Le segment de code de l'instruction **Else** se termine avec l'instruction **EndIf**.

- **Condition ?** : Expression booléenne permettant l'exécution du code.

Exemple

```

If{x% < 0}
    print{"x est négatif"}
Else{
    print{"x est nul ou positif"}
EndIf

```

A.3.3 StopPlan

StopPlan met fin à l'exécution du plan courant.

Exemple

```

If{fin? = true}
    StopPlan
EndIf

```

A.3.4 BeginConcurrentPlans - EndConcurrentPlans

Le mot réservé **BeginConcurrentPlans** délimite une zone d'appel de plan devant s'exécuter en concurrence. Ainsi chaque plans contenus entre l'instruction **BeginConcurrentPlans** et **EndConcurrentPlans** sera appeler et le plan courant sera mise en attente tant que tous les plans ne seront pas terminés.

Exemple

```

BeginConcurrentPlans
    Get Tax
    Get Promotion
EndConcurrentPlans

```

A.4 Concurrence

Lorsque des plans sont exécuter en concurrences avec le mot réservés `BeginConcurrentPlans`. l'interpréteur des plans crée un nouveau registre de variable pour chacun des plans exécutés. Pour accéder à une variable contenue dans le plan parent, il faut ajouter un signe de soulignement (`_`) avant le nom de la variable.

Exemple

```
x = _taux * _montant
```

A.5 Exemple

```

SubPlan {Compute Mortgage}
    BeginConcurrentPlans
        Check Credit
        Check Insurance
        Get Rate
        Get Ratio
        Get Tax
        Get Promotion
    EndConcurrentPlans
    If{Cote% = 0}
        print{'You were refused by the credit Agency.''}
        send{requestAgent$, ''reply'', 'Netsa',
            requestReplyWith$, ''message = You were refused by the
            credit Agency. ;'', ''pageID = Error.html ;''}
    StopPlan
EndIf
//Compute Payment
x% : (1 + (Rate% / 2)) ^ (1 / 6)
num% : x% - 1
n% : amortization% * 12
den% : 1 - (x% ^ (0 - n%))
payment% : num% / den%
subLoan% : loan% - downPayment%
subPayment% : subLoan% * payment%

```

```
BeginConcurrentPlans
    Send Results
EndConcurrentPlans
EndSubPlan
```

Annexe B

Captures d'écran

Voici quelques captures d'écran de l'agent superviseur.

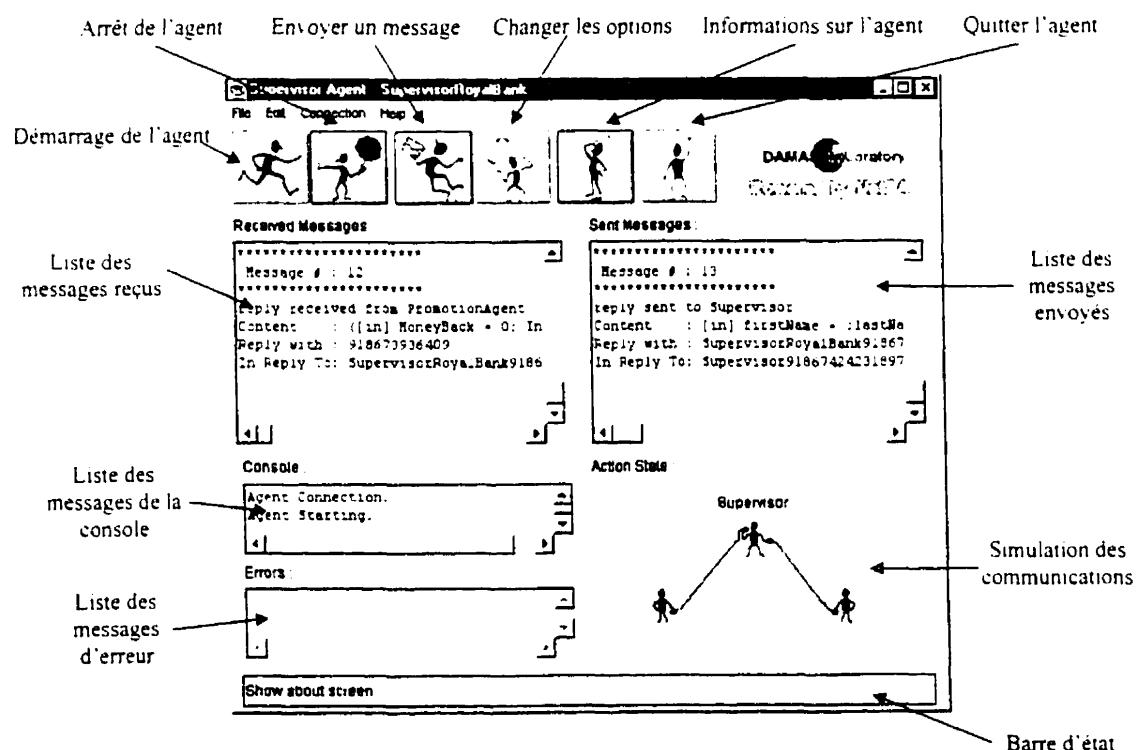


Figure B.1: Fenêtre principale de l'agent superviseur.

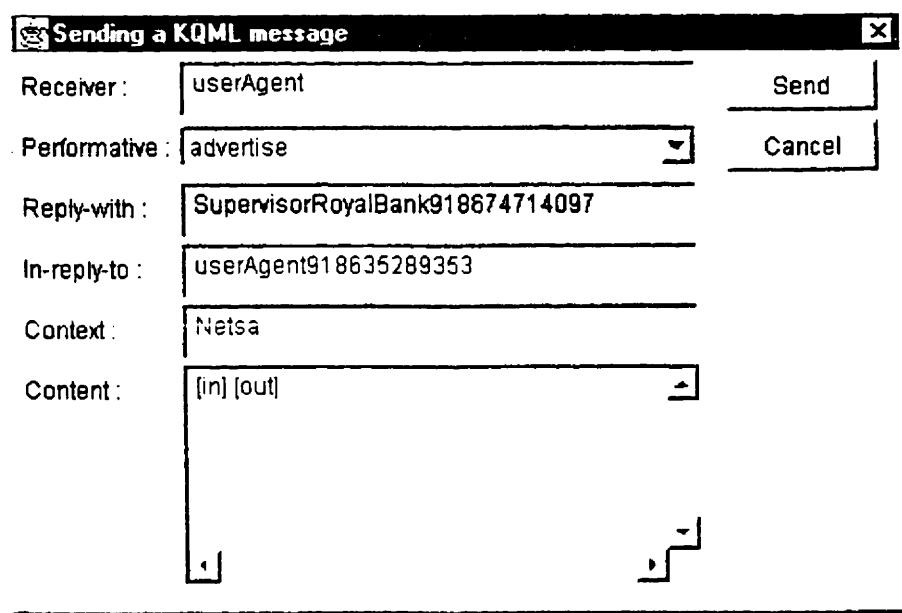


Figure B.2: Fenêtre d'envoi manuel de messages de l'agent superviseur.

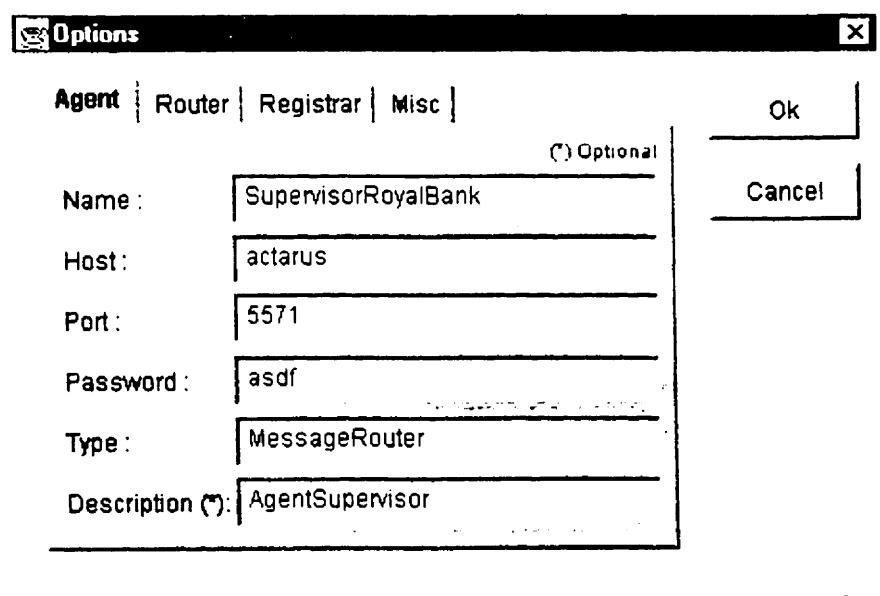


Figure B.3: Fenêtre de configuration pour l'agent.

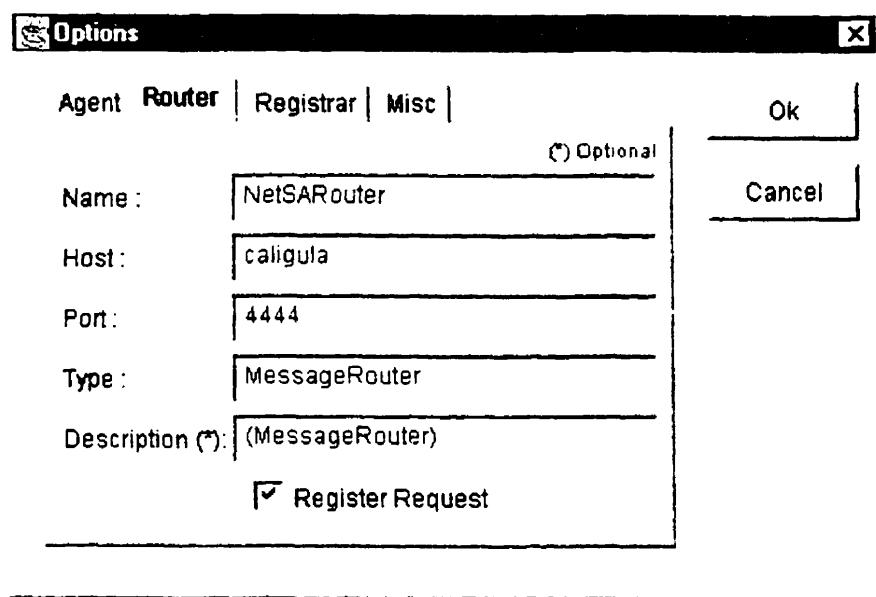


Figure B.4: Fenêtre de configuration pour le router.

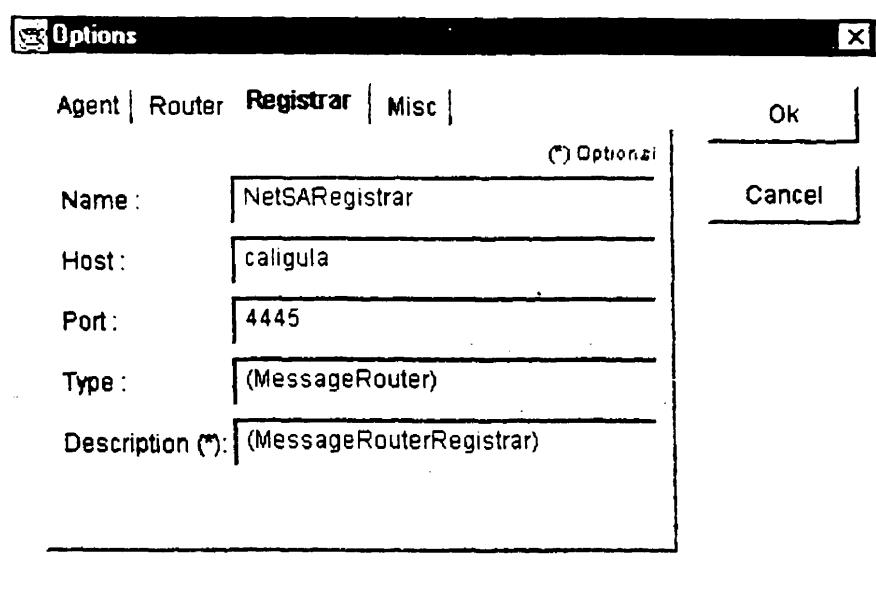


Figure B.5: Fenêtre de configuration pour le registrar.

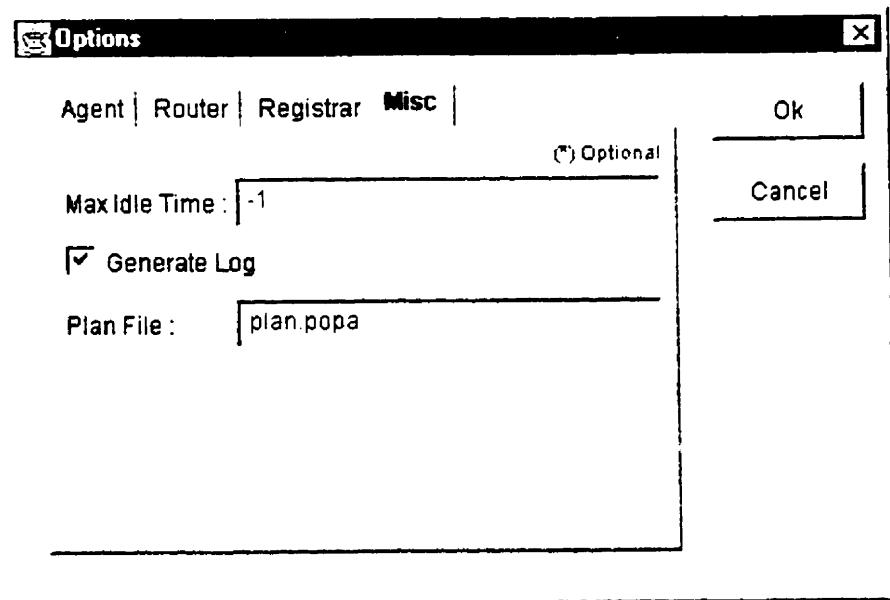


Figure B.6: Fenêtre de configuration générale.