UNIVERSITE DE TECHNOLOGIE DE COMPIEGNE

U. F. R. DE SCIENCES ET TECHNOLOGIE

Doctorat

Technologies de l'information et des systèmes

FABRICIO ENEMBRECK

CONTRIBUTION A LA CONCEPTION D'AGENTS ASSISTANTS PERSONNELS ADAPTATIFS

Thèse dirigée par **Jean-Paul BARTHÈS**

Soutenue le 05 décembre 2003

Jury :

M. Joël QUINQUETON (Président)
M. Philippe MATHIEU (Rapporteur)
M. Bernard ESPINASSE (Rapporteur)
M. Dominique FONTAINE
M. Jean Paul BARTHÈS

Je dédie la réalisation de ce travail à mes parents José et Carmela, à Fabiane et à mes nièces Fábia et Flávia qui ont supporté toutes ces années d'absence.

Remerciements

Je voudrais exprimer mon immense gratitude à mon directeur de thèse Jean-Paul Barthès, qui a su me soutenir pendant les moments difficiles, me guider pendant les moments de doute et par les innombrables heures de travail nécessaires à la correction des mes articles et de ce mémoire.

Je remercie mes collègues de travail Cesar, Emerson et Marco pour les discussions qui, sans aucun doute, ont contribué à l'évolution des idées présentées dans ce mémoire.

Je tiens à remercier Indira Thouvenin, Marie-Hélène Abel, Gilles Kassel, Jean-Yves Fortier et Bruno Ramon, qui ont contribué au bon déroulement du projet AACC, au sein duquel j'ai travaillé pendant la durée de la thèse.

Je remercie le personnel administratif et technique du laboratoire HEUDIASYC qui m'a aidé de nombreuses fois à résoudre mes problèmes.

Je remercie le Conseil de la Région Picardie qui a financé cette recherche pendant ces trois années.

Je remercie également le personnel de l'Association Gradient pour leur professionnalisme et la bonne volonté mise dans le suivi de mon dossier pendant toutes ces années.

Résumé

Tout au long de ce mémoire nous étudierons plusieurs aspects de la conception des agents assistants personnels. Nous avons développé au long de la thèse plusieurs mécanismes qui permettent de matérialiser les représentations conceptuelles proposées par Ramos (2000), comme les modèles du monde, de soi, du maître ou encore, de l'interaction. Nous mettons l'accent sur l'aspect adaptatif de ces modèles, ce qui permet à l'assistant de personnaliser ses actions en fonction du maître. Pendant le développement de nos travaux nous nous sommes aperçus que la plupart des méthodes d'apprentissage couramment utilisées ne sont pas vraiment adaptées aux agents assistants personnels, car elles sont inefficaces lorsqu'un petit nombre d'exemples d'apprentissage est disponible et lorsque le temps pour l'apprentissage est réduit. Nous avons donc développé notre propre méthode d'apprentissage (ELA – Entropy-based Learning Approach) qui a été testée et dont les résultats sont significatifs. Nos travaux s'achèvent sur une étude des modèles permettant d'intégrer les différentes représentations développées tout au long de la thèse à travers différents prototypes. Nous avons obtenu un modèle de système multi-agent capable de gérer les différentes représentations dynamiques de l'utilisateur à travers des agents distribués dotés de mécanismes d'apprentissage automatique. Nous avons donc conçu un modèle générique de système multi-agent personnalisé (SMAP) qui résout les problèmes inhérents à l'architecture centralisée de Ramos et conserve l'aspect de personnalisation au niveau de l'interaction et de l'exécution des services.

Mots-Clés :

Agent assistant personnel, Système multi-agent, Apprentissage automatique, Personnalisation.

Abstract

In this thesis we study several aspects related to the developpement of personnal assistant agents. We have developped some mechanisms allowing the implementation of the conceptual models proposed by Ramos (2000) like models of the world, of self, of user or model of interaction. We stress the importance of the adaptativity of such models to the user's actions. During the development of the work we perceived that common methods of adaptation are not good enough for adaptative personal assistant agents, because they cannot learn enough from a few learning examples in a limited time. Thus, we introduced a new learning approach (ELA – Entropy-based Learning Approach) that has been evaluated and the results of which are quite good. Our work ends up with a study allowing the integration of the different representations developed in the prototypes. We have designed a multi-agent representation for managing the different dynamic representations of the user with distributed learning agents. This representation is modeled by a generic personalized multi-agent system architecture (SMAP – Système Multi-Agent Personnalisé) solving some of the problems introduced by Ramos' centralized architecture and allowing a personalization of the interaction and of services.

Keywords :

Personal Assistant Agent, Multi-Agent System, Learning, Personalization.

Table de Matières

1 Introduction 1
1.1. L'organisation du mémoire 3
2 Les agents assistants personnels 7
2.1. L'intelligence artificielle distribuée 7
2.2. Qu'est ce qu'un agent ? 10
2.3. Qu'est ce qu'un agent assistant personnel? 10
2.3.1. Agent intelligent et agent assistant personnel 11
2.3.2. Agent intelligent et apprentissage automatique 12
2.3.3. Modélisation de l'utilisateur et apprentissage automatique 13
2.4. Structure générale d'un agent assistant personnel 14
2.4.1. Les actions 14
2.4.2. Le modèle des tâches 15
2.4.3. Les ontologies 16
2.4.4. Le modèle du monde 18
2.4.5. Le modèle de soi-même 19
2.4.6. Le modèle du maître 19
2.5. La plate-forme OMAS 20
2.5.1. Le modèle interne des agents OMAS 20
2.5.2. Communication 21
2.5.3. OMAS et FIPA 21
2.6. Les assistants personnels et les agents autonomes adaptatifs 22
2.6.1. Les agents autonomes 23
2.6.2. Les agents autonomes réactifs 25
2.6.3. Les agents autonomes délibératifs 26
2.6.4. Les agents autonomes adaptatifs 29
2.6.5. Les agents assistants adaptatifs 37
2.7. Les problématiques 41
2.7.1. L'architecture d'agent assistant personnel 41
2.7.2. Les agents assistants personnels adaptatifs 42
2.8. Conclusions 43
3 Adaptation et personnalisation de l'interaction 45
3.1. Les assistants, les interfaces adaptatives et les systèmes hypermédias adaptatifs 45
3.2. L'interaction par dialogue et l'adaptation 47
3.3. La conception d'un agent assistant capable de dialoguer 50
3.3.1. Le système de dialogue 51
3.3.2. Le modèle de dialogue 51
3.3.3. Comment interpréter les expressions de l'utilisateur ! 55
3.3.4. Coordination du dialogue 58
3.3.5. Le modèle de tâches 62
3.3.6. Utilisation du profil de l'utilisateur 64
3.3.7. Modèle du domaine 66
3.3.8. Considérations sur les assistants de dialogue 68
3.4. Conclusions 69

4 Adaptation et personnalisation des compétences 71
4.1. Modélisation de l'utilisateur 72
4.1.1. La modélisation de comportement 74
4.1.2. La modélisation d'intérêts 75
4.2. Les assistants adaptatifs et le raisonnement utilisant la mémoire 76
4.3. Les assistants personnels, la classification et le filtrage de documents 79
4.3.1. La classification et le filtrage de documents 80
4.3.2. Les agents et le filtrage collaboratif 82
4.3.3. L'apprentissage du profil de l'utilisateur 83
4.3.4. Le centre de gravité pondéré et l'agent décision 85
4.3.5. Le pouvoir de discrimination des caractéristiques 86
4.3.6. La classification des documents 87
4.3.7. Performance 89
4.3.8. Considérations sur le filtrage collaboratif 90
4.3.9. Discussion et travaux concernés 92
4.4. Les assistants personnels et la capitalisation des connaissances 93
4.4.1. La structure générale de l'agent assistant personnel 96
4.4.2. L'architecture du projet AACC et le rôle de l'agent assistant personnel 97
4.4.3. Le rôle de l'assistant 99
4.4.4. L'interface de l'utilisateur 101
4.4.5. Le modèle de tâches 102
4.4.6. Group Awareness 104
4.4.7. Comportement pro-actif 104
4.4.8. Organisation et récupération de tâches 105
4.4.9. Représentation des préférences de l'utilisateur 105
4.4.10. L'aide en langage naturel 106
4.4.11. Discussion sur les agents et la capitalisation des connaissances 108
4.5. Conclusions 110

5 La méthode ELA (Entropy-based Learning Approach) 111
5.1. Caractéristiques 111
5.2. Le mécanisme d'apprentissage 113
5.2.1. La représentation des données 113
5.2.2. Classification 116
5.3. Pourquoi cela marche-t-il? 119
5.4. Complexité de calcul 120
5.5. Résultats expérimentaux 122
5.5.1. Méthode de comparaison 122
5.5.2. Les résultats 122
5.5.3. ELA et NN 124
5.6. Réduction des données 125
5.6.1. Quelques techniques connues 125
5.6.2. Réduction des données avec ELA 127
5.6.3. Méthode de comparaison 127
5.6.4. ELA, ELAr et RT3 128
5.6.5. ELAr et Moyenne16 128
5.6.6. Commentaires généraux 129
5.7. Des travaux similaires 129
5.7.1. Construction de graphes 129
5.7.2. L'usage du graphe 130

5.8. La problèmatique et la solution apportée 130
5.9. Conclusions 131

6 Le système MAIS 133
6.1. Problématique 134
6.2. Les agents adaptatifs et le web 135
6.3. Les systèmes multi-agents et le web 136
6.4. Pourquoi des systèmes multi-agents dans la recherche d'information 137
6.5. Architecture MAIS 138
6.5.1. La gestion des sites favoris 139
6.5.2. Le processus de recherche 139
6.5.3. L'agent assistant personnel (AP) 140
6.5.4. L'agent bibliothèque (AB) 145
6.5.5. L'agent de filtrage (AF) 146
6.5.6. L'agent de recherche (AR) 150
6.5.7. La conception du système MAIS 150
6.6. Le Protocole de Recherche 151
6.6.1. AUML 151
6.6.2. La représentation des Messages 153
6.6.3. Les lignes de vie parallèles 153
6.6.4. Les protocoles imbriqués 154
6.6.5. AUML étendue 154
6.6.6. Le Protocole de Recherche 155
6.7. Qualité des Résultats 157
6.8. Discussion et travaux concernés 161
6.8.1. Qualité du profil de l'utilisateur 162
6.8.2. Couverture de l'espace Internet 162
6.8.3. Des informations sémantiques 163
6.9. Conclusions 163

7 Un SMA personnalisé 165
7.1. Les architectures des SMA et les assistants 165
7.1.1. Des SMA interactifs non-personnalisés 166
7.1.2. Des SMA interactifs personnalisés : le concept du *majordome* 167
7.1.3. L'assistant et son staff 168
7.1.4. L'assistant dans une coterie 169
7.1.5. L'assistant et le staff dans une coterie 170
7.2. Idées générales 171
7.2.1. La capitalisation des connaissances 172
7.2.2. Le filtrage et la classification de documents 173
7.2.3. La recherche personnalisée d'information 173
7.3. Le déploiement 173
7.4. Définition formelle d'un SMA personnalisé 175
7.4.1. Les agents du système 175
7.4.2. La définition formelle du système 176
7.5. Services distribués vs résolution distribuée de problèmes 184
7.6. La problématique et la solution apportée 186
7.7. Conclusions 187

8 Conclusions Générales 189
Les perspectives de travaux futurs 190

Références bibliographiques 193

Annexes 205

Annexe 1 - Conception des SMA 207
Annexe 2 - Grammaire de la langue anglaise 211
Annexe 3 - Représentation de l'ontologie avec MOSS 219
Annexe 4 - Le langage MAIS-L 223

Table de Figures

2 Les agents assistants personnels..... 7
Figure 1 L'agent assistant et les domaines de recherche. 11
Figure 2 L'architecture générique d'un agent assistant personnel. 14
Figure 3 (a) Représentation de plan. (b) Représentation de tâche. 16
Figure 4 Un message FIPA avec des échanges entre connaissances d'une ontologie. 17
Figure 5 L'architecture d'un agent OMAS. 21
Figure 6 L'architecture générale d'une application OMAS. 22
Figure 7 Comportement d'un agent autonome. 23
Figure 8 Architecture générale d'un agent autonome. 24
Figure 9 Le modèle de subsumption. 25
Figure 10 L'Architecture d'un agent STRIPS. 27
Figure 11 L'architecture de dMARS (PRS). 28
Figure 12 L'assistant comme un agent autonome. 38
Figure 13 Classification de l'Agent Assistant Personnel. 39

3 Adaptation et personnalisation de l'interaction 45
Figure 14 Un dialogue ouvert. 52
Figure 15 Architecture du système. 54
Figure 16 Les structures internes. 55
Figure 17 Une requête formelle. 57
Figure 18 Les fonctions utilisées pour l'analyse sémantique. 58
Figure 19 Le contrôle du dialogue. 60
Figure 20 La définition d'une tâche avant son exécution. 63
Figure 21 Un dialogue intelligent. 67
Figure 22 Base de connaissances. 68

4 Adaptation et personnalisation des compétences 71
Figure 23 Un exemple de modèle d'intérêts. 76
Figure 24 Approche traditionnelle de classification de document 81
Figure 25 La technique du centre de gravité pondéré. 82
Figure 26 Architecture du système multi-agent. 83
Figure 27 Algorithme de Classification. 87
Figure 28 Comparaison avec l'algorithme Rocchio. 90
Figure 29 Queue de priorité. 92
Figure 30 Architecture du système. 98
Figure 31 L'interface de l'assistant. 102
Figure 32 Structure d'une tâche. 103
Figure 33 Types de tâches. 103
Figure 34 Une tâche complexe. 106
Figure 35 Vecteur de termes qui représentent la tâche de la Figure 34. 106
Figure 36 Hiérarchie de Concepts. 108
Figure 37 Fenêtre d'aide. 108

5 La méthode ELA (Entropy-based Learning Approach) 111
Figure 38 La construction du graphe. 114

Figure 39 Graphe créé à partir des concepts extraits du travail de Quinlan (1986). . 115
Figure 40 Schéma de l'entropie. 117
Figure 41 L'algorithme de classification. 118
Figure 42 Classification d'un exemple. 119
Figure 43 Représentation Graphique des résultats du Tableau 10 (* Résultats de (Kasif et al. 1998)). 123
Figure 44 Représentation Graphique des résultats du Tableau 11 (* Résultats de (Eklund et Hoang)). 124
Figure 45 Processus d'élimination des cas. 127

6 Le système MAIS 133

Figure 46 L'architecture du système MAIS. 138
Figure 47 L'interface de l'agent assistant personnel. 141
Figure 48 Les centres d'intérêts publics. 142
Figure 49 L'exécution de services avec des agents JADE. 145
Figure 50 Exemple d'une liste de termes choisis à partir du domaine « Agents Autonomes » avec la mesure TF-IDF. 146
Figure 51 Combinaison des structures de documents dans une structure générale. .. 148
Figure 52 Ressources Informatiques utilisés par le système. 151
Figure 53 Le Protocole FIPA Contract-Net. 153
Figure 54 Extension de la conjonction AUML. Ici (a), (b) et (c) représentent "conjonction avec timeout". 155
Figure 55 Protocole d'interaction - Partie I. 155
Figure 56 Protocole d'interaction - Partie II. 156
Figure 57 Le processus de classification des sites. 157
Figure 58 Processus de traitement de chaque site. 157
Figure 59 Performance mesurée sur les 20 premiers sites. 159
Figure 60 Performance mesurée sur les 30 premiers sites. 160
Figure 61 Résultats avec et sans les centres d'intérêts publics. 160

7 Un SMA personnalisé 165

Figure 62 Les SMAs non-personnalisés. 166
Figure 63 Les architectures des SMAs (a) PA avec staff (b) PA simple et coterie (c) PA et staff dans une coterie. 168
Figure 64 Architecture d'un SMA destiné à la personnalisation de services. 172
Figure 65 Un SMA personnalisé. 174
Figure 66 Architecture résultante d'agent assistant personnel. 187

Annexes 205

Figure 67 Communication entre agents d'un SMA. 207

1 Introduction

Dans ce mémoire nous étudierons plusieurs aspects concernant la conception d'agents assistants personnels au service d'un utilisateur appelé le *maître*. Nous avons développé tout au long de la thèse plusieurs mécanismes qui permettent de matérialiser les représentations conceptuelles proposées par Ramos (2000), comme les modèles du monde, de soi, du maître ou encore, d'interaction. Nous mettons l'accent sur l'aspect adaptatif de ces modèles, ce qui permet à l'assistant de personnaliser ses actions en fonction du maître. Malgré la cohérence des modèles proposés par Ramos, nous montrons que l'architecture centralisée qu'il propose peut devenir rapidement non maîtrisable lorsque l'assistant doit gérer plusieurs activités, car chaque activité peut faire appel à des mécanismes complexes. Ceci nous a donc conduit à proposer une nouvelle architecture décentralisée en dotant l'agent assistant personnel d'un *staff d'agents spécialisés* dotés individuellement de mécanismes de personnalisation. Une telle approche limite la complexité de l'assistant, conserve la modularité du système et favorise la personnalisation des services.

Nous avons développé au cours de nos travaux plusieurs prototypes qui mettent en évidence les mécanismes d'adaptation et l'évolution des modèles internes d'agent assistant personnel proposé par Ramos. Les mécanismes d'adaptation mis en œuvre sous forme d'algorithmes d'apprentissage peuvent être utiles dans deux cas :

- *Pour améliorer l'interaction* ;
- *Pour personnaliser les compétences*.

Dans le premier cas, l'adaptation sert à minimiser le besoin de feedback de l'utilisateur pendant le processus de compréhension d'un problème et à présenter des informations de façon personnalisée. Dans le deuxième cas, l'agent peut améliorer progressivement ses résultats en fonction des actions de l'utilisateur de façon à développer un comportement intelligent.

Au cours de nos travaux, nous nous sommes aperçus que la plupart des méthodes d'apprentissage couramment utilisées ne sont pas vraiment adaptées aux agents assistants personnels, car elles sont inefficaces lorsqu'un petit nombre d'exemples d'apprentissage est disponible et quand le temps pour l'apprentissage est réduit. Nous avons donc développé notre propre méthode d'apprentissage (ELA – Entropy-based Learning Approach) qui a été testée et pour laquelle nous avons obtenu des résultats significatifs.

ELA (Entropy-based Learning Approach) est une méthode incrémentale d'apprentissage, utilisant quelques concepts dérivés des « lazy algorithms » comme MBR, KNN et des arbres de décision incrémentaux pour construire dynamiquement un modèle[1] à partir des données. Les données sont représentées par un graphe de concepts. Chaque nœud du graphe représente une condition attribut-valeur, c'est-à-dire, une partition. Lorsqu'un exemple doit être classé, la méthode utilise l'entropie pour mesurer la qualité des partitions et la distribution des exemples, pour finalement identifier la qualité de chaque classe et retourner la meilleure. Nous montrons que dans le pire des cas la complexité de l'algorithme est $m \times n^2$. La méthode est performante et peut réduire considérablement la quantité de données nécessaires à l'apprentissage. Ces caractéristiques sont très importantes pour la conception d'un agent assistant personnel adaptatif.

La méthode ELA a été ensuite utilisée pour la construction d'agents adaptatifs d'un système multi-agent capable de personnaliser les résultats de recherches réalisées sur la toile. Nous avons constaté à partir de cette application qu'un mécanisme d'adaptation peut améliorer considérablement les recherches produisant des résultats beaucoup plus précis que ceux des moteurs de recherche classiques.

Nos travaux s'achèvent sur une étude des architectures des SMA permettant d'intégrer les différents mécanismes développés tout au long de la thèse à travers les différents prototypes.

---

[1] Ici nous appelons *modèle* une représentation conceptuelle qui décrit les patrons acquis à partir des données. Ce concept est utilisé dans le domaine de l'apprentissage automatique.

[2] *m* correspond au nombre d'attributs et *n* au nombre d'exemples d'une base de données.

Nous avons obtenu une architecture de système multi-agent capable de gérer les différentes représentations dynamiques de l'utilisateur à travers des agents dotés de mécanismes d'apprentissage automatique. Dans cette architecture l'agent assistant personnel est spécialisé dans l'interaction avec le maître, et l'exécution des services est assurée par des agents distribués. Nous introduisons avec cette architecture le concept de système multi-agent personnalisé (SMAP) démontrant que le concept d'agent assistant personnel de Ramos est insuffisant, pour la personnalisation des activités professionnelles d'un utilisateur, car le nombre de représentations à gérer et à mettre à jour est proportionnel au nombre de services et à la complexité de chaque service, ce qui aboutit à un agent assistant personnel beaucoup trop complexe.

1.1. L'organisation du mémoire

Ce mémoire est organisé de la manière suivante :

Chapitre II - Les agents assistants personnels

Dans ce chapitre nous introduisons les aspects théoriques liés au concept d'agent assistant personnel. Cela comprend une courte introduction à l'intelligence artificielle distribuée, la définition d'agent assistant personnel, la définition de la structure générique d'agent assistant personnel développée par Ramos (2000) qui a servi de point de départ pour nos agents et les mécanismes d'agents autonomes adaptatifs normalement utilisés pour mettre en place des entités autonomes et adaptatives.

Chapitre III - Adaptation et personnalisation de l'interaction

Dans ce chapitre nous discutons comment l'adaptation peut être mise en place pour personnaliser l'interaction avec l'utilisateur. Dans un premier temps, nous discutons l'adaptation des interfaces classiques, mettant en évidence les relations entre les agents assistants personnels, les interfaces adaptatives et les systèmes hypermédias adaptatifs. Dans un deuxième temps, nous introduisons les interfaces basées sur le langage naturel, décrivant en détail une application comprenant un système de dialogue développé pendant la thèse. Nous discutons également les mécanismes d'adaptation concernant ce mode d'interaction.

Chapitre IV - Adaptation et personnalisation des compétences

Dans ce chapitre nous discutons comment faire évoluer les compétences de l'agent assistant personnel de façon à personnaliser les services. Cette tâche est normalement accomplie à l'aide d'un modèle précis de l'utilisateur. Nous discutons donc comment créer et faire évoluer le modèle de l'utilisateur. Nous constatons à partir de l'analyse de plusieurs travaux que certaines techniques sont plus fréquemment mises en œuvre que d'autres pour l'apprentissage du modèle de l'utilisateur, par exemple, les algorithmes d'apprentissage en ligne. Dans ce chapitre nous présentons deux applications qui utilisent cette technique d'apprentissage. La première est liée à la classification et au filtrage automatique de documents et la deuxième concerne un assistant destiné à favoriser le travail collaboratif.

Chapitre V - La méthode ELA (Entropy-based Learning Approach)

Dans ce chapitre nous discutons les problèmes posés par les techniques classiques de modélisation automatique de l'utilisateur. Nous introduisons la méthode que nous avons développée pour résoudre ces problèmes. Nous comparons le comportement de cette méthode à d'autres travaux similaires sur plusieurs critères comme complexité, performance et capacité de réduction de données. Nous concluons que la méthode est performante et peut réduire considérablement le volume de données d'apprentissage tout en nécessitant moins de calculs que l'algorithme classique du plus proche voisin.

Chapitre VI – Le système MAIS

Dans ce chapitre nous discutons comment la méthode développée au chapitre V (la méthode ELA) peut être utilisée pour la recherche d'information personnalisée sur le web. Nous détaillons la conception du système MAIS (Multi-Agent based Internet Search). MAIS est un système multi-agent pour la recherche d'information capable d'adapter les résultats des recherches en fonction du modèle d'utilisateur. Les résultats obtenus par MAIS sont meilleurs que ceux des moteurs de recherche classiques comme Google, Altavista ou All-the-web.

Chapitre VII - Un SMA personnalisé

Alors que dans les chapitres précédents nous avons présenté séparément plusieurs applications d'agents assistants personnels destinés à l'adaptation de l'interaction et des services fournis à l'utilisateur, dans ce chapitre nous discutons la conception d'un seul SMA

capable d'accommoder tous ces services, tout en gardant l'aspect adaptatif des agents. Nous introduisons alors un modèle formel de SMA destiné à la personnalisation et capable d'interagir avec l'utilisateur.

Chapitre VIII - Les conclusions

À la fin de ce mémoire nous introduisons nos observations concernant l'évolution du travail, les problèmes rencontrés et les résultats obtenus. Nous indiquons également comment les travaux réalisés tout au long de la thèse pourraient être améliorés, et donnons la perspective de nouvelles directions de recherche.

Annexes

Les annexes sont composées de quatre parties. La première partie introduit des concepts techniques utilisés pour la conception de systèmes multi-agents. La deuxième partie présente la grammaire développée pour l'analyse syntaxique des expressions en anglais, utilisée par le système de dialogue du chapitre III. La troisième partie présente une petite ontologie, développée en MOSS, utilisée par la fonctionnalité d'aide en langage naturel du système de capitalisation des connaissances du Chapitre IV. Le langage de contenu échangé par les agents assistants personnels du système MAIS est présenté dans la quatrième partie des annexes.

2 Les agents assistants personnels

Dans ce chapitre nous étudions les éléments principaux généralement rencontrés dans les architectures d'agents assistants personnels. Tout d'abord, nous mentionnons quelques concepts généraux définissant les SMAs. Nous précisons ensuite les domaines de recherche concernés par le développement d'agents assistants personnels et détaillons une architecture d'agent assistant personnel proposée par Ramos (2000), capable de modéliser la plupart des applications contenant des agents assistants et qui nous servira de point de départ. Ensuite nous discutons les relations entre agents autonomes, agents autonomes adaptatifs et agents assistants personnels, concluant que les agents assistants personnels sont à la fois autonomes et adaptatifs. En fin de chapitre, nous analysons les insuffisances du modèle d'agent assistant personnel proposé par Ramos et des méthodes d'adaptation connues, quand on veut les employer pour construire des agents autonomes adaptatifs.

2.1. L'intelligence artificielle distribuée

Depuis les années 50, les chercheurs se sont intéressés à la construction de systèmes intelligents. Au début, pour être considéré intelligent, un système devait être capable de réaliser une tâche complexe, normalement réservée aux êtres humains, comme prouver des théorèmes, jouer aux échecs ou faire un diagnostic médical. Pour résoudre ces problèmes, les systèmes étaient dotés d'une grande connaissance dans le domaine d'application, représentée dans un formalisme donné (par exemple des règles de production). Ces applications ont été à l'origine des premiers systèmes experts.

Après les premières années d'excitation, l'Intelligence Artificielle (IA) s'est calmée à cause des limitations concernant les logiciels et le matériel informatique disponible à l'époque. À la fin des années 70, avec le recrutement de nouveaux chercheurs par des universités américaines comme MIT, Stanford et Carnegie Mellon, et, par conséquent, l'apparition de nouvelles techniques fondées sur la programmation logique, le raisonnement qualitatif, les modèles d'apprentissage neuronaux et symboliques, l'IA est redevenue un domaine de recherche très actif. C'est à cette époque que certains chercheurs comme Erman et Lesser (1975), Hewitt (1977) et Smith (1979) ont mis l'accent sur le besoin de fabriquer des entités intelligentes distribuées capables de communiquer et de résoudre des problèmes qui ne peuvent pas facilement être résolus de façon centralisée.

Erman et Lesser ont développé une architecture de coordination entre entités spécialisées qui partagaient des informations à partir d'un espace d'information commun. Chaque expert (nommé *base de connaissance*) était spécialisé dans une tâche spécifique concernant la reconnaissance de la parole et ses connaissances étaient codées sous forme de règles. Les hypothèses produites par chaque expert étaient stockées dans une mémorie commune nommée *blackboard*. Selon Erman et Lesser l'architecture blackboard constituait une méthode de résolution distribuée de problème car les agents partageaient des connaissances et pouvaient s'aider mutuellement.

Hewitt a proposé le modèle *Actor*, composé d'entités nommées *actors* capables de communiquer par transmission de messages et de travailler en parallèle. Les acteurs ont une durée de vie limitée à l'accomplissement de leurs actions codées sous forme de scripts. Pendant l'échange de messages, de nouveaux acteurs peuvent être créés et ainsi diminuer la complexité du problème global. Les acteurs peuvent également migrer vers d'autres machines et donc équilibrer la charge de travail dans le système. Selon Hewitt, la théorie des acteurs pouvait contourner les limitations concernant le logiciel et le matériel informatique disponible à l'époque de façon à résoudre des problèmes complexes en Intelligence Artificielle.

Smith a proposé l'un des premiers protocoles d'allocation de tâches utilisés par des agents, le protocole *Contract Net*. Dans le Contract Net un agent (le gérant) qui doit résoudre un problème de réponse inconnue doit d'abord choisir un contractant. Si le contractant est capable de donner une réponse il le fait, sinon il envoie en broadcast une annonce de contrat et attend les propositions des candidats. Ensuite, il utilise une stratégie pour choisir ceux qui peuvent donner la meilleure solution et confirme les contrats. La résolution d'un problème

peut impliquer plusieurs contrats. Finalement le contractant donne la réponse au gérant. Ce protocole fut à l'origine de nombreux travaux concernant les protocoles d'interaction entre agents.

Ces travaux ont créé une nouvelle perspective de recherche : les recherches concernant l'*Intelligence Artificielle Distribuée*. Parmi les motivations qui ont poussé les recherches dans le domaine de l'Intelligence Artificielle Distribuée nous pouvons citer : la résolution de problèmes devient plus rapide, car les agents exécutent des *calculs en parallèle*, le système devient plus *flexible*, car l'information ne dépend plus d'une seule source et peut être entretenue de façon modulaire, et la *fiabilité*, car des agents peuvent redistribuer le travail lorsqu'un agent tombe en panne. Par ailleurs, certaines applications sont naturellement distribuées, comme l'indiquent Moulin et Chaib-draa (1996), comme le contrôle de groupes de robots ou l'interprétation de données sensorielles distribuées.

Les recherches en Intelligence Artificielle Distribuée peuvent être classées en deux groupes : la *résolution distribuée de problèmes* et les *systèmes multi-agents*.

La résolution distribuée de problème concerne les techniques qui permettent à un ensemble de modules (agents) de diviser et de partager des connaissances concernant l'état courant et la solution d'un problème particulier. Une technique bien connue est celle de l'*éco résolution* proposée par Ferber (1995). Dans cette technique la solution d'un problème est obtenue lorsque tous les agents du système sont satisfaits. Si un agent n'est pas satisfait, il agresse les agents qui l'empêchent d'être satisfait. La propagation d'agressions sous forme de contraintes constitue ainsi un processus de recherche dans l'espace des solutions du problème. Cette technique est généralement employée lorsque la solution du problème est inconnue ou trop complexe pour être mise en œuvre dans un système centralisé, comme l'emploi du temps. Cependant, Ferber montre comment l'éco-résolution peut être utilisée dans des problèmes classiques comme le monde des cubes, les tours de Hanoï ou le Taquin.

La recherche sur les systèmes multi-agents (SMA) concerne les mécanismes qui permettent la mise en place d'entités distribuées pour la résolution d'un problème donné. Cela implique plusieurs mécanismes comme : le modèle de comportement des agents (autonomes, sociaux, communicants), les méthodes de raisonnement (réactifs, délibératifs), les protocoles d'interaction (coordination, coopération, négociation), l'infrastructure de communication ou encore les méthodes et environnements de développement. Ainsi, du point de vue de la

conception, un agent doit contenir plusieurs de ces mécanismes. Même si certains auteurs comme Huhns et Singh (1998) se sont plutôt intéressés à l'aspect opérationnel d'un agent (construction de mécanismes), nous pensons qu'il est important de donner une définition conceptuelle du terme « agent » du point de vue de la communauté SMA. Cette définition aidera le lecteur non spécialiste des SMA à comprendre la suite de ce mémoire.

2.2. Qu'est ce qu'un agent ?

Dans la discussion précédente, nous avons vu comment des agents sont mis en place pour résoudre des problèmes de façon intelligente. Cependant, aucune définition conceptuelle d'un agent n'a été donnée. La raison principale est que chaque nouvelle définition d'agent est influencée par les applications d'un certain domaine et, par voie de conséquence, ne s'applique pas à tous les cas.

Bien qu'une définition générique acceptée par toute la communauté SMA semble improbable (voire utopique), nous proposons une définition générale qui peut s'adapter à n'importe quel type de problème. Le lecteur intéressé par d'autres définitions peut consulter les travaux de Ferber (1995), Russell et Norvig (1995), Scalabrin (1996) ou Jennings (1998).

Un agent est une entité logicielle qui joue un rôle et qui peut avoir plusieurs caractéristiques (autonomie, adaptabilité, perception, sociabilité) et modèles internes (de soi-même, des autres, de communication ou de l'environnement). Le rôle de l'agent et les ressources informatiques disponibles définissent quelles caractéristiques et modèles l'agent doit contenir dans le cadre d'une application spécifique.

À partir de cette définition très générale, nous introduisons dans la section suivante une définition des agents qui nous intéressent plus particulièrement : les *agents assistants personnels*.

2.3. Qu'est ce qu'un agent assistant personnel?

Depuis des décennies, les techniques informatiques ont évolué énormément et nous sommes confrontés jour après jour à des environnements informatiques plus puissants et plus complexes. Souvent, les utilisateurs sont dépassés et incapables de gérer leurs activités quotidiennes personnelles et professionnelles, qui dépendent de plus en plus de ces environnements. Dans une telle situation, nous pouvons identifier deux attitudes : ou l'utilisateur agit seul, cherchant à apprendre les concepts manquants pour gérer son

environnement informatique, ou il fait confiance à un assistant informatique capable de le guider vers la résolution de certains problèmes et de le remplacer pendant l'exécution de tâches ennuyeuses. Ce type d'assistant est connu sous le terme d'*Agent Assistant Personnel*.

Du point de vue informatique, un agent assistant personnel est un agent qui joue le rôle d'assistant et dont l'objectif est de diminuer la charge de travail de son *maître* pendant la réalisation des activités profissionnelles qui demandent une interaction avec l'ordinateur ou un autre moyen informatique.

L'agent assistant personnel peut avoir une certaine *autonomie* pour « remplacer » son maître dans la prise de certaines décisions et raisonner de façon *réactive* lorsque son fonctionnement interne est piloté par les sensations perçues à partir de l'environnement, ou de façon *délibérative* quand il utilise un mécanisme de délibération (planification). Il doit être *adaptatif*, pour améliorer ses performances en fonction du temps.

Dans ce contexte, nous pouvons identifier au moins trois domaines de recherche distincts concernant au développement des agents assistants (Figure 1):

- Les agents intelligents ;
- L'apprentissage automatique ;
- La modélisation de l'utilisateur.

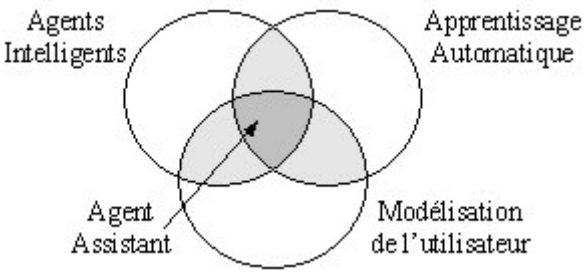

Figure 1 L'agent assistant et les domaines de recherche.

2.3.1. Agent intelligent et agent assistant personnel

L'utilisation du terme « agent assistant » est souvent mise en cause, car dans certaines applications le système n'est pas autonome, ne contient pas de connaissances et n'est pas non plus adaptatif (voir par exemple les agents compagnons de Microsoft Office). Nous pouvons donc nous poser la question suivante : un tel système est-il un agent ?

Certains concepts du domaine des systèmes multi-agents ont leur origine dans les études du comportement des entités naturelles, notamment ceux qui concernent les aspects sociologiques et philosophiques qui apparaissent lorsque ces entités collaborent pour accomplir des tâches. Parmi ces concepts, nous pouvons souligner le partage de connaissances, la communication, les ressources ou encore, les compétences. Ces entités (agents) sont donc souvent mises en œuvre dans la reproduction de situations réelles du monde et peuvent représenter n'importe quelle entité naturelle ou artificielle, notamment des êtres humains. Ainsi, pour certains chercheurs, un agent doit faire partie d'un monde complexe habité par d'autres agents. L'ensemble d'agents forme une communauté.

Ferber (1995) explique par exemple qu'un individu tout seul a moins de chance d'apprendre convenablement une nouvelle langue s'il n'interagit pas avec d'autres individus. Par conséquent, l'intelligence d'un agent ne dépend pas seulement de ses capacités individuelles, mais aussi de la vie en communauté. Pourtant, dans plusieurs applications d'agents assistants, notamment les assistants web, les agents sont isolés et n'ont pas de contact avec d'autres individus. Ils ressemblent plutôt à des systèmes experts, spécialisés dans des tâches bien précises, comme la recherche d'information ou la navigation sur le web. Ils communiquent seulement avec l'utilisateur. Par conséquent, les concepts de communauté ou de collectivité n'existent pas. Nous pouvons citer par exemple la communauté de chercheurs intéressés par les *agents interface* (Etzioni et Weld, 1998) (Lieberman, 1997), qui soutient qu'un agent peut être autonome et n'a pas besoin d'autres agents, car il interagit seulement avec l'utilisateur et d'autres ressources informatiques. Par ailleurs, c'est dans cette catégorie d'agents que se retrouvent la plupart des applications d'agents assistants (voir par exemple les applications discutées par Middleton (2001)).

Dans ce mémoire nous mettons l'accent plutôt sur les aspects pratiques liés à la conception d'un agent assistant personnel et ne discutons que superficiellement les questions philosophiques et souvent polémiques posées par la conception d'un assistant logiciel (i.e., s'il est intelligent ou pas, s'il est un agent ou pas ou s'il est isolé ou pas).

2.3.2. Agent intelligent et apprentissage automatique

Dans les chapitres suivants nous verrons en détails les techniques d'apprentissage automatique qui peuvent être employées pour la conception et le développement d'un agent autonome. L'apprentissage est utile pour l'évolution des modèles internes de l'agent. Cela

permet à l'agent d'agir dans des situations imprévues. L'évolution des modèles de l'agent favorise aussi l'optimisation de ses actions lui permettant de mieux accomplir ses objectifs.

Souvent, l'apprentissage automatique est mis en œuvre dans des agents autonomes pour améliorer leurs mécanismes d'action. En effet, un algorithme d'apprentissage doit acquérir des connaissances sur les états du monde et les effets causés par les actions de l'agent. Ainsi, l'agent peut choisir les actions qui rapprochent l'état du monde de celui qui satisfait au mieux ses intérêts.

Puisque pour nous un agent assistant personnel est normalement un agent autonome (dans le sens où il peut agir sans avoir la permission du maître), une approche semblable à celle du paragraphe précédent peut être utilisée pour améliorer sa performance. L'apprentissage se fait donc en observant les actions de l'utilisateur et les actions de l'agent qui contribuent à la satisfaction du maître.

Étant donné que le comportement de l'utilisateur n'est pas figé, (il peut changer de centres d'intérêts et ses compétences évoluent), l'apprentissage constitue donc un processus continu qui permet à l'assistant d'accompagner ces changements.

2.3.3. Modélisation de l'utilisateur et apprentissage automatique

L'une des caractéristiques les plus importantes d'un agent assistant personnel est l'évolution du modèle de son maître. Modéliser l'utilisateur consiste à fabriquer des structures capables de représenter l'état mental de celui-ci. *Ces modèles varient en fonction des différentes applications*. Pour la gestion documentaire, par exemple, l'agent peut modéliser les centres d'intérêts de l'utilisateur à travers des vecteurs de termes. Pour le cas d'aides spécialisées, l'agent peut utiliser des règles de comportement et des niveaux d'expertise. Ainsi, l'apprentissage automatique est souvent ajouté au modèle de l'utilisateur afin d'améliorer la représentation du maître dans le système.

Dans cette thèse nous nous sommes intéressés principalement à l'évolution dynamique des modèles existants dans un agent assistant personnel (notamment celui de l'utilisateur).

Cette approche permet de minimiser l'effort initial de modélisation de l'utilisateur, car le système peut démarrer avec un modèle minimal (voire inexistant). À chaque interaction avec le maître, le système ajoute une nouvelle observation, normalement représentée comme un

vecteur de variables, à une représentation conceptuelle. Ensuite, cette représentation peut être utilisée pour simuler l'état comportemental et les croyances du maître.

2.4. Structure générale d'un agent assistant personnel

Nous avons adopté la définition d'assistant personnel présentée par Negroponte (1997) et développée par Ramos (2000). Cela veut dire que l'assistant est spécialisé dans l'interaction avec l'utilisateur. Ramos (2000) a proposé une architecture générale d'agent assistant personnel ayant les composantes suivantes : *actions*, *tâches et projets*, *ontologies*, modèle du *monde*, modèle de *soi-même* et modèle du *maître*.

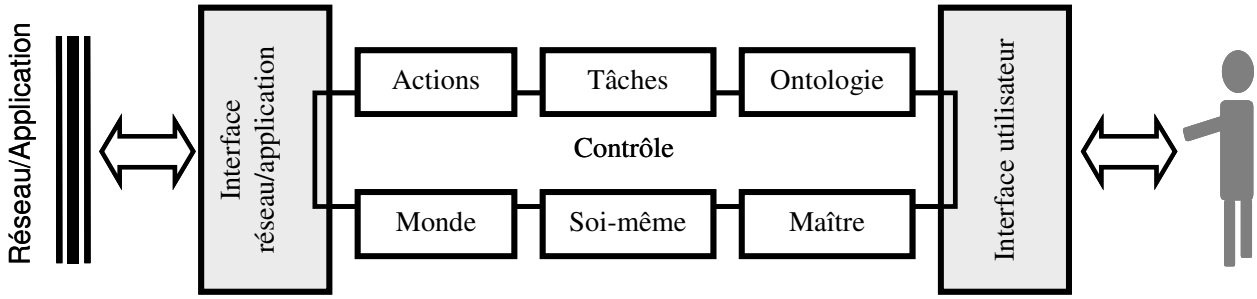

Figure 2 L'architecture générique d'un agent assistant personnel.

En raison de la complexité d'une telle architecture, une application l'utilisera rarement en totalité. Par ailleurs, cette architecture présente une vision structurelle de l'agent, c'est-à-dire, ne contient que les informations à représenter et non les mécanismes de raisonnement cognitif nécessaires. Pour conserver une certaine cohérence avec les architectures d'agents autonomes étudiés dans la section 2.6.1, nous supposons l'existence d'un mécanisme de *sélection d'action* composé par les modules suivants : actions, tâches, monde et maître. Dans les sections suivantes, nous allons étudier comment chacun des modules présentés sur la Figure 2 est représenté dans les applications existantes.

2.4.1. Les actions

Les actions sont normalement celles demandées par le maître ou d'autres agents et correspondent aux compétences de l'agent. Elles sont liées à des procédures informatiques qui permettent à l'agent d'atteindre ses objectifs. La nature des actions dépend, évidemment, de l'utilité de l'agent. Dans une application du style compagnon, par exemple, l'action consiste à *présenter une certaine information x*. Payne et al. (1995) ont créé une application pour classer des messages d'e-mail. À partir d'un message, le système choisit un dossier dans lequel garder le message. Les actions possibles sont : classe le message comme *cfp* (call for paper),

agents, *mbox*, *jobs*, *kdd*, *dai,* etc. Lieberman (1997) a développé un agent assistant pour l'aide à la navigation qui avait l'action suivante : donne une liste de sites liés aux sujets *x*, *y* et *z*.

Le nombre d'actions est toujours limité, selon Maes (1994). L'une des stratégies d'adaptation les plus courantes, consiste à apprendre le modèle des actions. Dans le cas d'agents réactifs comme ceux du paragraphe précédent la question consiste à choisir une action parmi l'ensemble des actions possibles. Dans le cas d'agents délibératifs la question consiste à apprendre une séquence d'actions qui permette à l'agent de d'atteindre ses objectifs, comme par exemple, organiser une réunion ou encore concevoir un artefact électro-mécanique.

2.4.2. Le modèle des tâches

Les tâches correspondent normalement à la partie délibérative de l'agent. Elles contiennent des stratégies de fonctionnement qui permettent à l'agent d'atteindre ses objectifs. Normalement ces stratégies sont composées de plans qui représentent les actions à long terme, nécessaires pour accomplir une certaine activité. Nous discutons en profondeur l'utilité des plans dans la section 2.6.3 (les agents autonomes adaptatifs délibératifs). Dans une application proposée par Silva et Demazeau (2002) les plans et tâches sont des structures statiques qui indiquent à l'agent comment une réunion doit être organisée. Dans cette application les objectifs des agents sont représentés sous forme de plans hiérarchisés dans de graphes AND (Figure 3a). Par exemple, pour organiser une réunion l'agent doit vérifier le calendrier, confirmer la réunion et ajouter la réunion dans l'agenda. D'autres relations peuvent être représentées comme des contraintes (liens *inhibit*) et précédence (liens *enables*). Les tâches sont représentées sous forme d'un graphe qui met en relation les objectifs, les actions à prendre et les ressources (Figure 3b). Dans la Figure 3 (b) la tâche de vérification du calendrier est décomposée en trois sous tâches (*verify free slot*, *change status* et *release resource*) et liens *mutex* mettent en relation la ressource *calendar* avec ces sous-tâches.

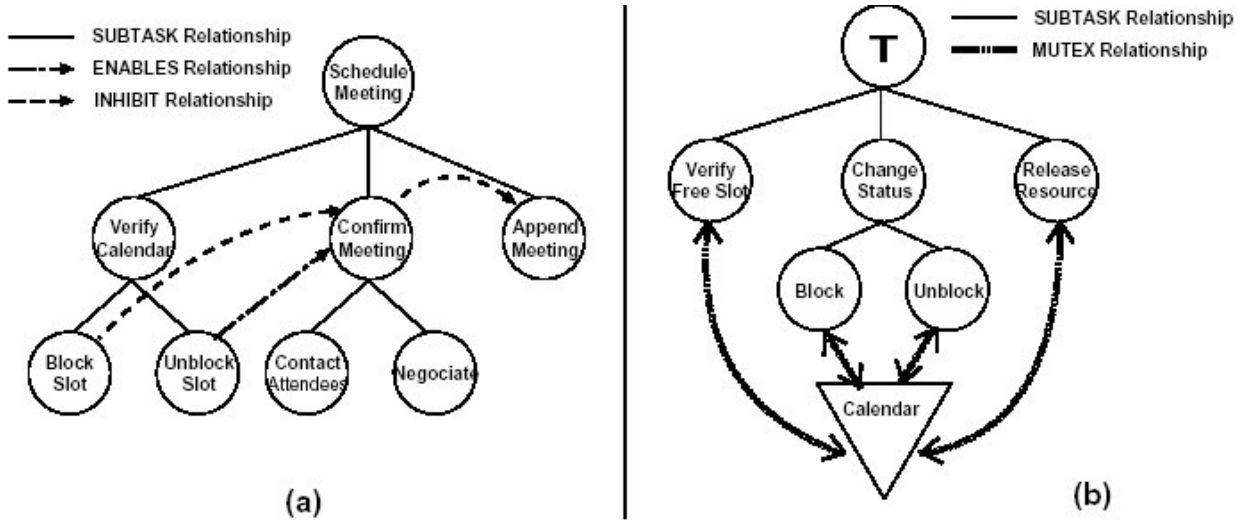

Figure 3 (a) Représentation de plan. (b) Représentation de tâche[3].

2.4.3. Les ontologies

Les ontologies sont souvent mises en œuvre dans des SMAs pour assurer une sémantique commune à la terminologie utilisée dans les messages échangés entre les agents.

Gruber (1993) a mis en évidence le besoin d'utiliser un vocabulaire commun pendant l'échange des connaissances entre des entités. Dans un article il écrit :

> *« La spécification d'un vocabulaire de représentation pour un domaine partagé, contenant par exemple, des définitions de classes, des relations, des fonctions, et d'autres objets, est appelée ontologie ».*

Par conséquent, un ensemble d'agents communicants doit partager un ensemble de symboles ayant une même signification. FIPA (Foundations for Intelligent Physical Agents) (FIPAa) (FIPAd) décrit un modèle standard de communication d'informations à partir d'ontologies. Dans FIPA, un agent nommé Ontology Agent (OA) est responsable du stockage des différentes ontologies du système. Selon FIPA, l'OA doit fournir les services suivants :

- Permettre l'accès à des ontologies ;
- Assurer l'entretien d'un ensemble d'ontologies ;
- Traduire des expressions appartenant à différentes ontologies et apparaissant dans le contenu des messages ;

[3] Cette Figure a été extraite de Silva et Demazeau (2002).

- Répondre à des requêtes sur des relations entre termes et entre ontologies ;
- Faciliter l'identification d'une ontologie partagée pour la communication entre deux agents.

Pour permettre l'échange entre bases des connaissances de chaque agent, FIPA propose un modèle de représentation d'information nommé OKBC[4] Knowledge Model. Ce modèle contient les structures standards de représentation des systèmes de *frames*, comme des classes, des objets, des relations, des slots, des facettes et des valeurs. Ensuite, FIPA a développé la FIPA-Meta-Ontology contenant de nombreux symboles, notamment les prédicats les plus courants pour la manipulation des *frames*. Par exemple, dans le message de la Figure 4 l'agent « ontology-agent@foo.com » informe à l'agent « client-agent@foo.com » que baleine est une sous-classe de mammifères, où « subclass-of » est un service disponible dans l'ontologie FIPA. L'ontologie courante est, évidemment, « animal-ontology ».

```
(inform
  :sender
    (agent-identifier
      :name ontology-agent@foo.com
      :addresses (sequence iiop://foo.com/acc))
  :receiver (set
    (agent-identifier
      :name client-agent@foo.com
      :addresses (sequence iiop://foo.com/acc)))
  :language FIPA-SL2
  :ontology (set FIPA-Ontol-Service-Ontology animal-ontology)
  :content
    (subclass-of baleine mammifère))
```

Figure 4 Un message FIPA avec des échanges entre connaissances d'une ontologie.

Pour les agents assistants, l'utilisation des ontologies peut être très utile, notamment pour assurer l'échange d'informations entre des utilisateurs qui n'ont pas le même niveau de connaissances sur un domaine déterminé. Lacher (2000) (2001) a travaillé dans cette direction. Dans (Lacher, 2000) l'auteur utilise une ontologie pour conserver la cohérence entre les messages échangés dans un système d'agents pour la capitalisation des connaissances. Dans son système, chaque groupe d'utilisateurs contient des agents qui accompagnent les utilisateurs individuellement. Les agents fabriquent le profil de l'utilisateur

[4] Open Knowledge Base Connectivity

au fil de l'eau prenant en compte ses activités. Ces agents partagent des informations pour découvrir quels utilisateurs ont des profils voisins. L'ontologie permet aux agents d'interpréter les messages en fournissant les noms des services et les termes nécessaires à la construction de communautés d'utilisateurs. Dans (Lacher, 2001) l'auteur soutient que chaque utilisateur possède des connaissances particulières. Le système utilise des ontologies privées et collectives pour faciliter l'échange de documents entre des groupes de personnes. Les termes des ontologies sont extraits de documents personnels par des techniques automatiques de traitement du texte. Ensuite l'auteur applique certaines techniques d'apprentissage automatique pour grouper les documents et découvrir quels utilisateurs ont des vocabulaires et des centres d'intérêts communs. Bien que l'importance des ontologies soit évidente pour un agent assistant personnel, Ramos (2000) ne discute aucun mécanisme formel spécifique dans sont architecture.

2.4.4. Le modèle du monde

La plupart des agents assistants personnels sont développés de façon isolée, l'agent étant spécialisé dans une tâche bien précise. Dans ce cas, il est difficile de justifier un modèle du monde. Cependant, certaines applications sont composées de plusieurs sous-applications. Dans ce cas, une représentation individuelle de l'état de chaque application est envisageable. Cette technique a été utilisée par Crabtree (1998) dans la construction d'un ensemble d'agents personnalisés. L'auteur a développé quatre agents spécialisés dans les services suivants : (i) recherche de personnes ayant des intérêts similaires vis-à-vis du web, (ii) proposition de nouveaux centres d'intérêts basés sur un filtrage collaboratif, (iii) recherche de pages web intéressantes, (iv) recherche de pages web à partir d'événements déclenchés dans un éditeur de texte. Au niveau le plus haut l'agent *Profiler* observe les événements déclenchés et les actions de l'utilisateur pour fabriquer un profil qui peut être partagé par toutes les autres applications. L'auteur a intitulé son architecture *Personal Agent Framework*. Ainsi le monde peut être modélisé comme l'ensemble des états possibles de chaque application. En examinant ce modèle l'agent peut décider de mettre à jour le profil de l'utilisateur (par exemple, après une recherche sur le web). Ramos soutient que le modèle du monde peut être construit dynamiquement à partir de l'analyse des messages transmis. Son approche rappelle celle des agents *matchmakers* de Sycara et collaborateurs (2001) qui fabriquent la représentation des agents et des services dynamiquement. Ensuite ces agents fonctionnent comme agences de

services. Toutefois, le modèle du monde de Ramos est composé simplement d'un log de messages et aucune représentation de plus haut niveau des autres agents n'est mise en place.

2.4.5. Le modèle de soi-même

Le modèle de soi-même selon Ramos (2000) est composé de deux parties principales : Une représentation des *informations personnelles* de l'agent (l'identification, les compétences, des meta-modèles, etc.) et une *mémoire ou modèle des expériences* passées. Dans le premier cas, les informations sont par exemple, *l'identification de l'agent* dans le cas des informations transmises à l'AMS[5] (système de gestion d'agents) et la *description de services* dans le cas de DF (Directory Facilitator) (voir FIPAc). Pour les expériences passées, rien n'empêche l'agent d'utiliser un mécanisme d'apprentissage qui lui permette d'optimiser ses communications avec d'autres agents. Ainsi, la plupart des modèles d'apprentissage automatique, notamment ceux de la classification automatique, peuvent être utilisés pour accomplir cette tâche. Nous faisons un bilan des techniques d'apprentissage dans la section 2.6.4. Un mécanisme d'apprentissage peut construire des généralisations à partir d'un ensemble de messages. Ces généralisations représentent des modèles généraux trouvés parmi les messages. Ces modèles peuvent alors être utilisés pour découvrir, par exemple, à qui l'agent doit demander un certain service ou encore prédire quel sera le résultat d'un service donné.

2.4.6. Le modèle du maître

Le modèle le plus important pour un agent assistant personnel est, sans aucun doute, celui de son maître. La complexité du modèle de l'utilisateur peut varier beaucoup en fonction des différents objectifs de l'agent. En général, le modèle de l'utilisateur est composé d'une partie *statique*, d'une partie *semi-dynamique* et d'une autre partie *dynamique*.

- La partie statique est celle qui concerne les informations personnelles de l'utilisateur, comme par exemple, le nom, la date de naissance, l'adresse, la profession, etc. Ces informations, sont évidemment fournies par l'utilisateur et peuvent être utilisées pour la personnalisation de mécanismes standards d'interaction, comme par exemple, des compliments personnalisés ou des propositions de valeurs standards ;

[5] Agent Management System dans le cadre de l'approche FIPA

- La partie semi-dynamique est celle qui est modifiée par l'utilisateur au fil du temps. Par exemple, ses créneaux horaires, les activités à faire ou le modèle de présentation d'information ;

- La partie dynamique est celle acquise par l'agent au fur et à mesure que l'agent interagit avec l'utilisateur. Cette partie est notamment divisée en deux sous-parties : le *modèle de comportement* et le *modèle des préférences*. Le modèle de comportement est utilisé dans des applications multi-tâches où le but est de modéliser la façon dont l'utilisateur travaille pour essayer d'anticiper les actions à prendre. Le modèle des préférences contient normalement les centres d'intérêts déterminés à partir des documents personnels.

La modélisation de l'utilisateur est le centre d'intérêt principal de cette thèse et sera discutée en détail dans les sections suivantes.

2.5. La plate-forme OMAS

Dans la section précédente nous avons discuté les différents modèles qui constituent la structure générale d'un agent assistant personnel. Cette struture générale a été inspirée en grande partie par l'organisation interne des agents de la plate-forme OMAS (Open Multi-Agent System) (Barthès, 2002). Par ailleurs, tout au long de cette thèse nous présenterons une série d'applications qu'ont été dévéloppées avec cette plate-forme d'agent. Dans cette section, nous discutons brèvement l'organisation interne des agents OMAS et l'infrastructure technique fournie par cette plate-forme pour le dévéloppement de SMA. Le lecteur non spécialiste de la terminologie technique utilisée dans la conception de SMA est invité à consulter l'Annexe I.

2.5.1. Le modèle interne des agents OMAS

Nous utilisons souvent des agents OMAS comme des agents qui fournissent des services à d'autres agents (dans ce cas l'agent est nommé *agent service*) ou à des agents du type assistant. Un agent service OMAS est une application LISP qui permet la mise en œuvre de la plupart des mécanismes discutés dans la thèse de Ramos (2000) et dans la section 2.4. Cette architecture a inspiré l'architecture générale d'agent assistant personnel présenté par Ramos. L'architecture interne d'un agent service OMAS est présentée dans la Figure 5.

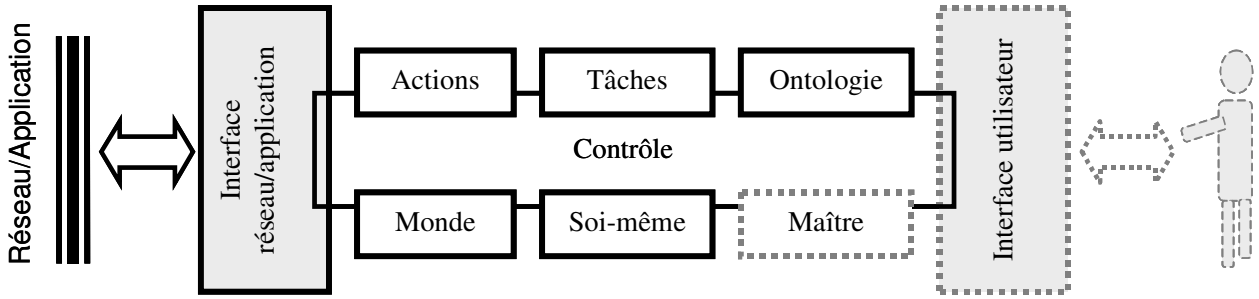

Figure 5 L'architecture d'un agent OMAS.

La différence la plus perceptible par rapport à l'architecture discutée dans la section 2.4 est l'absence de mécanismes pour représenter la communication avec l'utilisateur. Par conséquent un agent OMAS n'est pas capable de dialoguer avec un être humain et ne communique qu'avec d'autres agents. La définition de chaque composant de cette architecture a été discutée dans la section 2.4.

2.5.2. Communication

Les agents OMAS évoluent dans un environnement ouvert où les agents peuvent apparaître et disparaître au fil de l'eau. La communication entre les agents est toujours faite en mode broadcast à l'aide de sockets UDP qui partagent la même porte de communication. Pour communiquer entre des plates-formes à distance (dans des réseaux différents) les plates-formes échangent des messages HTTP. L'interaction entre des agents OMAS se fait normalement à travers le protocole Contract-Net sous forme de messages de type KQML. Le contenu des messages échangés correspond à des structures du type LISP.

Dans nos applications OMAS nous utilisons souvent le concept de *coterie*. Une coterie est composée par un ensemble d'agents placés physiquement proches les uns des autres (par exemple, dans le même réseau, dans le même labo, etc.) où ils partagent le même canal de communication. Par ailleurs, les agents d'une coterie reçoivent tous les messages. En général, dans une coterie, les agents utilisent un protocole de transport qui n'est pas connecté, comme par exemple UDP.

2.5.3. OMAS et FIPA

L'interopérabilité entre les agents OMAS et des agents développés dans d'autres plates-formes compatibles avec FIPA (FIPAa) est mise en place avec un agent nommée l'*Agent de Transfert* (AT) d'une façon transparente. L'AT met en place des services élémentaires de communication FIPA comme l'*Agent Manager System*, le *Directory Facilitator* et l'*Agent*

Communication Channel. L'AT est capable de traduire des messages OMAS dans messages FIPA SL-0 et assurer la cohérence entre les protocoles OMAS et des protocoles FIPA. Par conséquent, l'AT joue le role de passerelle entre OMAS et d'autres plates-formes compatibles avec FIPA. La Figure 6 montre l'architecture générale du deploiement d'une application OMAS. Nous discutons en détail le développement de l'AT dans le rapport technique (Enembreck et Barthès, 2001).

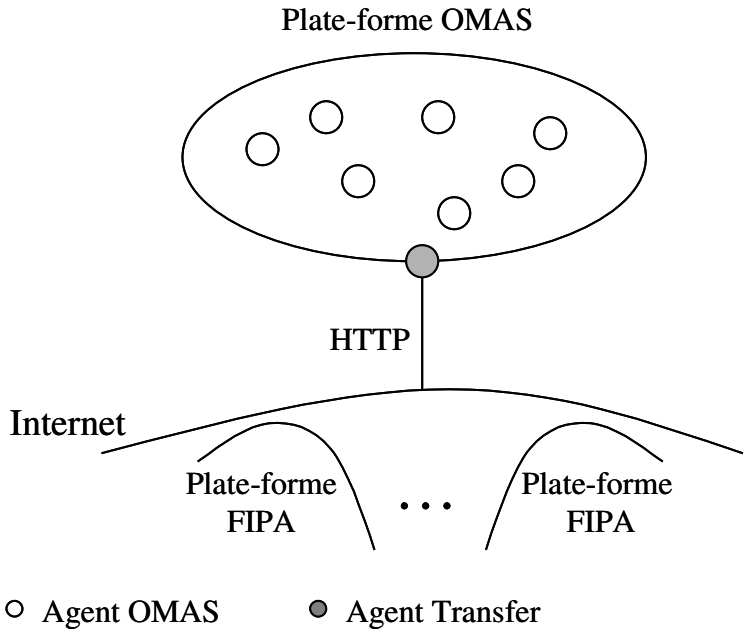

Figure 6 L'architecture générale d'une application OMAS.

2.6. Les assistants personnels et les agents autonomes adaptatifs

Dans cette section nous allons étudier deux catégories d'agents : les agents *autonomes* et les agents *autonomes adaptatifs*. Cette étude est nécessaire car la conception d'un agent assistant personnel normalement emploie des techniques rencontrées dans ces deux types d'agents. Ces techniques mettent en œuvre deux aspects fondamentaux pour la fabrication réussie d'un agent assistant personnel : l'*autonomie* et l'*adaptabilité*. L'autonomie permet à un agent assistant personnel de prendre en charge certaines activités avec le but de diminuer le travail de son maître. L'adaptabilité permet à l'agent d'améliorer l'interaction avec son maître. Cette section s'achève par une discussion sur comment situer l'agent assistant personnel parmi les catégories d'agents étudiées.

2.6.1. Les agents autonomes

Selon Boissier (2001), si l'on se réfère à leur comportement, on peut classer les agents en trois catégories principales : les agents *autonomes*, les agents *communicants* et les agents *sociaux*.

Les agents autonomes *vivent dans un environnement dynamique* et *doivent agir de façon autonome pour accomplir leurs objectifs*. Normalement, les agents autonomes perçoivent les événements de l'environnement et, à partir de cette perception, déclenchent des actions qui causent d'autres événements comme indiqué sur la Figure 7. Ainsi un agent autonome est en général isolé et n'interagit pas de façon directe avec d'autres agents.

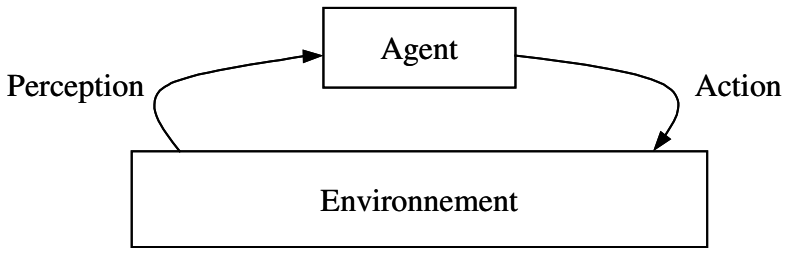

Figure 7 Comportement d'un agent autonome.

Les agents communicants en revanche sont capables de collaborer. Pour cela ils échangent des messages en utilisant un protocole de communication, des protocoles d'interaction et de coordination, un langage de communication d'agents (par exemple KQML) et un langage de contenu (par exemple FIPA SL ou KIF).

Enfin, les agents sociaux sont dotés de mécanismes de coordination à différents niveaux. Ces agents peuvent former des coalitions et constituer des entités ou groupes qui peuvent changer de structure au fur et à mesure de l'arrivée de nouveaux événements. Ces systèmes sont connus comme les systèmes multi-agents adaptatifs (Briffault et al., 2002).

Les études menées sur les agents assistants personnels nous ont permis de les identifier clairement comme des agents autonomes. Parmi les centaines de systèmes et applications utilisant des agents assistants personnels, la majorité peut être rangée dans la catégorie d'agents autonomes mentionnée précédemment. Même si certains agents assistants personnels possèdent en partie un comportement d'agent communicant, leurs mécanismes internes correspondent à une architecture d'agent autonome. Par conséquent, les échanges avec d'autres agents sont de simples demandes de services, comme dans une communication entre objets. Les agents assistants personnels ne sont pas en plus des agents sociaux, car ils ne sont

pas capables d'intervenir dans la vie ou dans le comportement d'autres agents. Par conséquent, les architectures d'agents autonomes peuvent être utilisées pour donner à l'agent assistant personnel un comportement intelligent.

Franklin (1997) propose une architecture générique d'agent autonome. Selon lui, un agent autonome doit avoir au moins les composantes suivantes : des capteurs, des actuateurs, un déclencheur, et un mécanisme de sélection d'action. L'agent perçoit l'environnement à travers des capteurs. Les capteurs fournissent des informations à un module de sélection d'actions qui choisit l'action la plus intéressante. Ensuite, un déclencheur lance l'action choisie. Ce modèle générique est illustré sur la Figure 8.

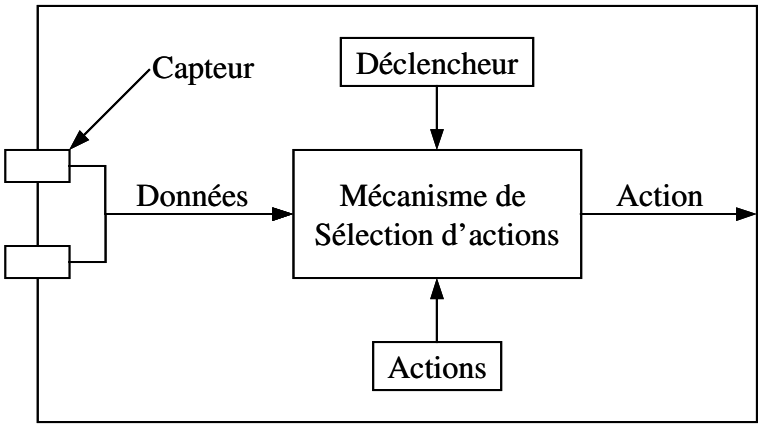

Figure 8 Architecture générale d'un agent autonome.

Il est clair sur la Figure 8 que le cœur de l'agent est le mécanisme de sélection d'action. Ce mécanisme représente un modèle cognitif du comportement de l'agent. Ce processus peut accéder à la mémoire à court terme (les expériences récentes) et à long terme (les expériences anciennes), exécuter un raisonnement, faire une classification, une planification, mettre en œuvre un apprentissage, etc. Ces techniques sont utilisées par deux courants de recherche qui s'opposent depuis longtemps sur la définition exacte d'un agent autonome. Le premier considère que les agents autonomes n'ont pas besoin d'avoir un modèle complexe de raisonnement pour agir, et, dans ce cas, les agents sont dits *réactifs*. Le deuxième argumente que, sans de tels modèles, l'agent ne peut pas avoir un comportement intelligent. L'agent doit pouvoir planifier ses actions avant d'agir. Dans ce dernier cas les agents sont dits *délibératifs*. Les paragraphes suivants vont détailler les différences entre ces deux approches et indiquer les principales architectures utilisées pour la réalisation de tels types d'agents.

2.6.2. Les agents autonomes réactifs

Le principal partisan de l'idée des agents réactifs est Brooks (1986) (1991). L'un des objectifs de Brooks était de développer des robots autonomes capables de se déplacer dans des salles, tout en arrivant à un endroit spécifique en évitant des obstacles placés aléatoirement. Les agents de Brooks n'ont pas besoin de communiquer directement, ils observent simplement les changements dans l'environnement. Les agents n'ont pas de représentation du monde. Selon Brooks, le monde lui-même est sa meilleure représentation. Il est plus simple d'utiliser le monde lui-même comme modèle et d'identifier uniquement les aspects qui ont un intérêt pour l'application. Ainsi, les agents n'ont pas besoin de modèles ni de méthodes cognitives complexes comme ceux mentionnés dans la section précédente. Dans (Brooks, 1986) l'auteur présente une architecture d'agents autonomes réactifs nommée *subsumption*. La subsumption est une méthode de représentation de comportement d'agents réactifs. Les agents n'ont pas de connaissances partagées. La subsumption est mise en place sous forme de couches composées par des machines à états finis qui relient les perceptions, les états de l'agent, les actions et des structures internes. Les couches inférieures représentent les actions atomiques de l'agent, comme par exemple, aller tout droit, ou tourner. Les couches supérieures représentent des comportements plus complexes et interviennent sur les structures des couches inférieures.

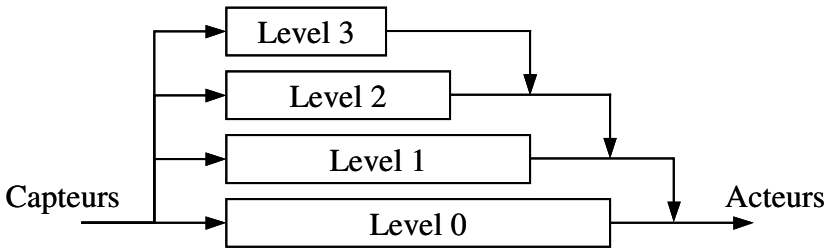

Figure 9 Le modèle de subsumption.

L'idée générale de la subsumption est de donner une priorité aux couches supérieures. Dans (Brooks, 1991) l'auteur donne les détails à propos du robot Allen construit au MIT. Par exemple, dans la première couche (celle des actions atomiques) le robot utilise des capteurs pour éviter les collisions et choisir la direction qu'il doit suivre. La deuxième couche choisit aléatoirement toutes les 10 secondes une nouvelle direction (qui se substitue à la première). La troisième couche observe des endroits distants et impose des nouvelles directions (qui se substituent à la deuxième).

Malgré la simplicité du modèle de Brooks, un effort considérable est nécessaire pour prévoir tous les conflits possibles. Cette architecture ne permet pas de réaliser des tâches en

collaboration facilement, car les agents ne partagent pas d'informations. Par ailleurs, un agent est codé de façon rigide, ce qui rendre la réutilisation de code difficile dans des applications différentes. Enfin, les agents ne peuvent pas améliorer leurs performances car ils n'ont ni mémoire ni mécanisme d'apprentissage.

2.6.3. Les agents autonomes délibératifs

Les agents autonomes délibératifs utilisent un modèle complexe de l'environnement dans lequel ils fonctionnent. Ce modèle est souvent utilisé pour la construction de structures cognitives qui permettent à l'agent de planifier et d'apprendre à partir des actions déclenchées. De façon plus précise, ce modèle permet de simuler les effets de l'exécution d'une action déterminée et d'évaluer la qualité de cette action dans une situation donnée. Le principal problème est que les modèles utilisés, malgré leur complexité, n'arrivent pas à donner une représentation suffisamment précise du problème, ce qui peut conduire à des erreurs. En revanche, la représentation même simplifiée de l'état du monde permet à l'agent d'identifier plus facilement les informations qui l'intéressent. La section suivante présente quelques techniques employées dans la construction d'agents délibératifs.

Modèle d'actions par planification

L'une des techniques les plus connues dans le domaine des agents délibératifs consiste selon Ferber (1995), à représenter l'environnement comme un état composé par un ensemble de formules logiques atomiques dont la conjonction est absolument vraie. Aujourd'hui encore, plusieurs systèmes utilisent l'ancien langage STRIPS (**S**tanford **R**esearch **I**nstitute **P**roblem **S**olver) développé par Fikes et Nilsson (1971) au début des années 70 pour représenter les états du monde. STRIPS est un planificateur utilisé dans le développement du robot *Shakey* dont le but était de se promener dans des salles, de pousser des boîtes et d'émettre des signaux lumineux. Un opérateur STRIPS représente une action et est composé par un *nom*, une liste de *pré-conditions*, une liste de faits à *ajouter* et une liste de faits à *supprimer*. Les pré-conditions sont les faits vrais nécessaires à l'exécution de l'opérateur. *Ajouter* est une liste de nouveaux faits produits à partir de l'exécution de l'opérateur et *supprimer* est une liste de faits qui disparaissent avec l'exécution de l'opérateur. Ainsi le problème de planification consiste à partir d'un état initial, à trouver une séquence d'opérateurs (actions) qui permette à l'agent de passer d'un état du monde à un autre état acceptable. Chaque séquence produite correspond à un plan. Ensuite, un sélectionneur va choisir le plan adéquat (voir la Figure 10).

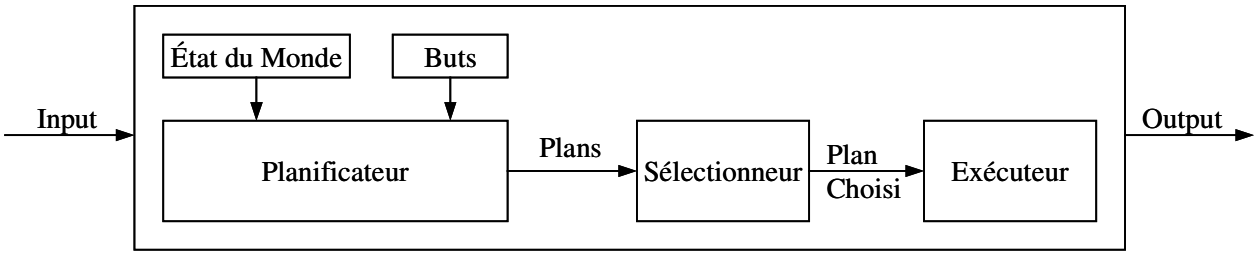

Figure 10 L'Architecture d'un agent STRIPS.

Cette approche a été améliorée par la suite par Nilson (1994) qui a développé un langage de contrôle d'agents autonomes nommé *Teleo*. Les programmes Teleo sont capables de gérer les plans dynamiquement en prenant en compte les changements de l'environnement, à la différence de STRIPS où les plans sont gérés avant l'exécution par l'agent. Les états de l'environnement sont représentés dans des structures semblables aux opérateurs STRIPS nommés TOPs (**T**eleo-**OP**erator).

Le modèle BDI

Il est clair que, d'après le paragraphe précédent, la planification pour les agents autonomes ne peut fonctionner que dans des domaines de complexité relativement limitée. Dans des domaines complexes, incertains et imprécis, notamment des domaines faisant intervenir des données quantitatives, la représentation symbolique en logique du premier ordre n'est pas suffisante pour représenter tous les états du système. Pour résoudre ce type de problème une autre architecture d'agents est souvent utilisée : l'architecture BDI (Belief Desire Intentions) (voir (Bratman, 1988), (Rao et Georgeff, 1995) et (d'Inverno et al., 1998)). Le comportement de l'agent est guidé par trois attitudes mentales : les *croyances*, les *désirs* et les *intentions*. Dans l'architecture BDI, chaque agent doit ainsi avoir une vision partielle de l'environnement, représenté de façon interne, par exemple, sous forme de formules logiques ou d'une base de données. Ces informations correspondent aux croyances de l'agent et sont normalement mises à jour au fur et à mesure que l'environnement change. De plus, il faut que l'agent ait une représentation des objectifs à atteindre, des priorités et des enjeux attachés à chaque objectif. Ces informations correspondent aux désirs de l'agent. Les intentions représentent l'aspect délibératif de l'agent où il doit mesurer la qualité des actions par rapport l'état courant, les objectifs à atteindre et la planification établie au préalable, c'est-à-dire, évaluer ce qu'il va faire effectivement. Dans d'Inverno (1998) l'auteur présente une architecture d'agent utilisée par le système dMARS. Cette architecture, nommée PRS

(Procedural Reasoning System), a été proposée initialement par Georgeff (1986). Elle est présentée sur la Figure 11.

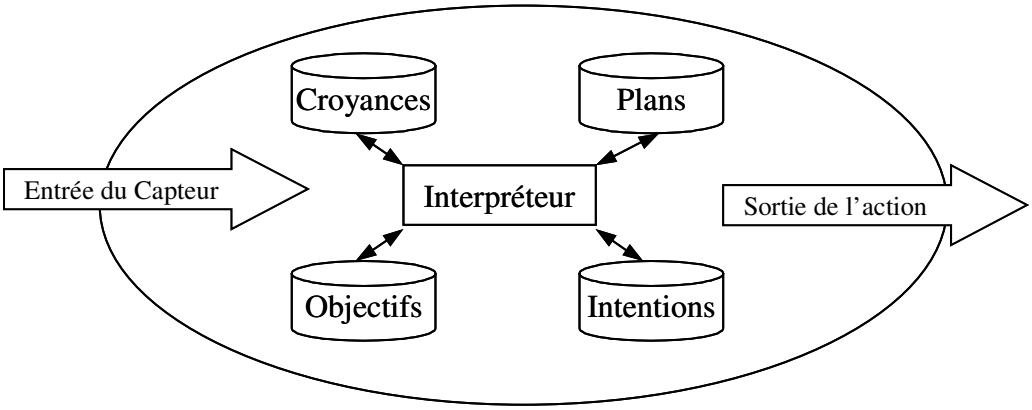

Figure 11 L'architecture de dMARS (PRS).

Chaque agent possède une bibliothèque de plans. Ces plans sont utilisés dans le choix de la tâche à accomplir et détaillent les actions à déclencher pour aboutir aux objectifs. Les objectifs choisis sont les intentions de l'agent. Nous pouvons décrire ce processus sous une forme algorithmique de la manière suivante, en nous plaçant du point de vue de l'agent :

- Regarder le monde et l'état interne de l'agent et ajouter les événements observés dans une queue d'événements ;
- Trouver les plans qui peuvent être déclenchés par les événements de la queue (les désirs);
- Choisir parmi les plans sélectionnés celui qui sera exécuté (les intentions);
- Si le plan est un sous-but, choisir la pile d'intentions adéquate et ajouter le plan courant ; sinon fabriquer une nouvelle pile de plans et ajouter le plan ;
- Choisir une pile d'intentions, prendre le plan du dessus et exécuter le pas suivant de ce plan ; si le pas est une action l'exécuter, sinon (s'il est un sous-but) mettre ce sous-but dans la queue d'événements.

De nombreuses autres architectures de type BDI ont été proposées ultérieurement. Cependant, nous nous limitons ici à la présentation des travaux principaux qui ont servi de base à ces systèmes. La comparaison entre les architectures BDI est hors du cadre de cette thèse car ces architectures sont rarement employées pour construir des agents assistants.

2.6.4. Les agents autonomes adaptatifs

Au-delà de l'autonomie, l'un des aspects les plus importants pour un agent assistant personnel est l'adaptation. Ainsi, dans cette section nous allons étudier les techniques utilisées dans la fabrication d'agents autonomes adaptatifs.

L'un des défis de l'Intelligence Artificielle consiste à concevoir des systèmes capables d'améliorer leurs performances à partir de leurs expériences. Cette capacité d'adaptation semble fondamentale pour les systèmes dont le comportement est autonome, car elle peut entraîner, au-delà de l'amélioration de leur performance, d'autres effets bénéfiques comme une économie de ressources et une meilleure robustesse. Les architectures d'agents autonomes qui présentent ce comportement sont dites *adaptatives*.

Si nous prenons l'exemple d'un robot similaire à celui de Fikes & Nilsson (section 2.6.3) nous percevons tout de suite l'utilité de l'apprentissage pour les agents autonomes. Disons que le robot doit parcourir 10 salles en permanence à la recherche de boîtes. Il s'aperçoit après quelques itérations que les salles de 1 à 3 sont toujours vides. Il peut alors décider de diminuer la fréquence de visite de ces salles. Ainsi, le robot augmente la vitesse d'exécution de sa tâche et dépense moins d'énergie en visitant les autres salles plus souvent.

Dans certaines applications, l'environnement n'évolue pas simplement au niveau des événements, mais aussi au niveau des structures. Dans ce cas, soit l'agent est codé à nouveau pour pouvoir continuer à fonctionner dans l'environnement, soit un mécanisme d'adaptation lui permmet d'évoluer. L'une des premières études approfondies mettant en évidence l'impact de l'apprentissage sur les agents autonomes a été proposée par Maes (1994). À cette époque, l'auteur a remarqué que les chercheurs en IA qui voulaient développer des agents autonomes travaillaient toujours dans les conditions suivantes :

- Ils travaillaient dans des domaines fermés où ils disposaient de toute la connaissance nécessaire ;
- Ils travaillaient sur des applications traditionnelles pour lesquelles l'environnement ne changeait pas pendant le fonctionnement du système ;
- Ils ne savaient pas comment faire évoluer les connaissances internes des agents.

Un agent autonome n'est normalement pas capable d'avoir une représentation très complexe du monde, il évolue dans un environnement dynamique et imprévisible et, surtout, il doit faire évoluer ses compétences. Pourtant, certains agents dotés d'une capacité d'apprentissage et d'adaptation peuvent améliorer leur performance malgré ces contraintes.

Dans les architectures étudiées précédemment une forme primitive d'adaptation est réalisée : le traitement des situations inattendues. Cependant, dans certains cas, une autre forme d'adaptation peut-être envisagée : *l'apprentissage à partir de l'expérience*. Dans ce dernier cas, l'agent doit s'améliorer au fur et à mesure que le temps passe. Cela signifie que l'agent doit apprendre à choisir les bonnes actions aux bons moments, c'est-à-dire qu'il doit y avoir une amélioration constante dans le mécanisme de sélection d'actions par l'agent.

Au début des années 90, un certain scepticisme régnait sur l'utilisation de mécanismes d'apprentissage dans des systèmes dynamiques comme ceux des agents autonomes, en raison de la précocité des recherches menées dans le domaine de l'apprentissage automatique. En effet, les chercheurs pensaient qu'un algorithme d'apprentissage nécessitait un effort de calcul énorme et une quantité également énorme de données, comme pour les réseaux de neurones par exemple. Heureusement, les techniques d'apprentissage ont évolué. Toutefois, tous les algorithmes d'apprentissage ne sont pas envisageables dans un agent autonome adaptatif car ils doivent avoir les caractéristiques suivantes :

- L'apprentissage doit être incrémental ;
- Il doit prendre en compte le bruit ;
- L'apprentissage doit être non-supervisé ;
- Éventuellement, il faut que l'algorithme puisse permettre d'utiliser des connaissances fournies par l'utilisateur et/ou par le développeur du système.

Un agent autonome n'a pas de temps à perdre avec des algorithmes qui prennent trop de temps pour apprendre. En effet, il y a deux catégories principales d'algorithmes d'apprentissage : (i) les *algorithmes non-incrémentaux* et (ii) les *algorithmes incrémentaux*. Dans le cas (i) l'algorithme travaille sur une base de données de taille connue et fabrique une série de généralisations pour représenter les données. Ensuite, ces généralisations peuvent être utilisées dans le raisonnement sur des données inconnues. Dans certains cas, cela peut

demander beaucoup de temps. Dans le cas (ii) l'algorithme fabrique des généralisations au fur et à mesure que les données arrivent. Au début l'algorithme fonctionne avec peu de données. Ainsi l'apprentissage se fait au fil de l'eau. Dans un environnement dynamique, un agent autonome reçoit des informations peu à peu et dispose rarement d'une grosse base de données, ce qui interdit l'utilisation d'algorithmes non-incrémentaux.

Les informations reçues par les agents autonomes viennent, normalement, des capteurs. Il est difficile de garantir la fiabilité totale des données provenant d'un environnement distribué, rempli d'entités différentes qui font des tâches différentes. Ainsi, une méthode qui essaie de prendre en compte toutes les nuances des données (notamment le bruit) ne donnera pas de bons résultats.

Deux autres catégories d'algorithmes d'apprentissage sont les algorithmes *supervisés* et les algorithmes *non-supervisés*. Les algorithmes supervisés reçoivent en entrée des données déjà classées et fabriquent ensuite des classifieurs capables de prévoir la classe d'autres exemples. Pour un agent autonome, par exemple, ils peuvent prévoir l'action à exécuter dans une certaine situation. Les algorithmes non-supervisés reçoivent des données non classées. Leur but est de grouper des exemples similaires dans des classes différentes en utilisant une mesure de similarité. Ainsi, pour être autonome, un agent doit lui-même être capable d'attribuer des classes aux observations qui arrivent.

Des algorithmes supervisés ou non-supervisés peuvent faire usage des connaissances fournies par l'utilisateur. Normalement, ces connaissances sont utilisées par les algorithmes pour résoudre des conflits ou encore pour faciliter la recherche de généralisations intéressantes pendant le processus d'apprentissage. Mitchell discute cette question dans (Mitchell, 1990).

Dans ces perspectives, selon Maes (1994) et Brenner (1998), trois techniques d'apprentissage sont utilisables pour développer des agents autonomes adaptatifs : *l'apprentissage par renforcement*, *les systèmes classificateurs* et *l'apprentissage de modèles*.

L'Apprentissage par renforcement (AR)

Cette méthode a été créée par Watkins en 1989 (Watkins, 1989). L'idée générale est d'améliorer la séquence d'actions prises par l'agent en attribuant des récompenses à des actions qui contribuent à la résolution du problème. Supposons que l'agent soit dans un environnement dans un état *s*. Au début, les poids des actions sont inconnus. À partir de

l'exécution d'une action *a* l'environnement évolue vers un état s_1. L'agent calcule alors la récompense $Q(s, a)$. La solution du problème consiste à déterminer dans chaque état *s*, la valeur $Q(s, a)$ qui maximise la récompense accumulée. Watkins utilise la fonction suivante pour mettre à jour les récompenses des actions :

$$Q_{n+1}(s,a) = (1 - \alpha) \times Q_n(s,a) + \alpha \times \gamma \left(r_{s'} + \max_{a' \in Actions} Q_n(s',a') \right)$$

où $0 < \alpha < 1$ et $0 < \gamma < 1$. α est un taux d'apprentissage (lié à l'importance de chaque pas vers la solution) et γ est un escompte constant. $r_{s'}$ est la récompense gagnée dans l'état s'.

Si le choix des actions est fait avec une probabilité uniforme pour chaque état du système et de façon continue, la fonction précédente assure une politique optimale de choix.

Maes (1994) indique toutefois quelques limites de la méthode d'apprentissage par renforcement :

- La méthode ne prend pas en compte des objectifs qui changent dans le temps : dans certains problèmes les objectifs de l'agent peuvent changer. Par conséquent, la priorité dans le processus de sélection des actions devra changer aussi. Le problème est que l'agent a raffiné son modèle d'une façon qui n'est plus adaptée aux nouveaux objectifs ;

- Si jamais les objectifs changent, le processus d'apprentissage doit recommencer à partir de zéro ;

- Dans certains domaines, l'espace de recherche donné par le produit état × action peut être très grand. Ainsi l'algorithme aura besoin de beaucoup de temps pour apprendre, ce qui risque de diminuer la performance de l'agent de façon significative ;

- Le calcul des récompenses peut également nécessiter beaucoup de temps ;

- L'algorithme a toujours besoin de connaître l'état courant du monde. Certaines imperfections dans les capteurs peuvent empêcher ou brouiller la vision que l'agent a du monde.

De nombreux travaux ont été entrepris avec l'espoir de résoudre ces problèmes, car de nombreuses applications utilisent cette méthode. Oliveira (2001) utilise l'apprentissage par

renforcement pour améliorer la coordination entre agents autonomes. Dans son approche, les agents apprennent le meilleur modèle de négociation. Chapelle, Simonin et Ferber (2002) utilisent une technique de récompense qui prend en charge la satisfaction des agents voisins dans une application de coordination d'agents pour le transport d'objets. Dans le cadre de la Cinquième Conférence Internationale sur les Agents Autonomes 2001, le lecteur pourra trouver trois autres applications d'apprentissage par renforcement : Makar (2001) utilise l'apprentissage par renforcement (AR) dans l'ordonnancement de tâches, Minut (2001) présente une application de reconnaissance visuelle d'objets et Shapiro (2001) discute comment améliorer la performance de l'AR avec des connaissances du domaine.

La deuxième technique d'apprentissage utilisée par les agents adaptatifs (les classificateurs) est discutée dans la section suivante.

Les systèmes classificateurs

La classification est la forme la plus courante de l'apprentissage automatique. Souvent, les agents autonomes utilisent des systèmes de classification pour découvrir à partir d'un certain état, la meilleure action à exécuter. Ces techniques sont généralement statistiques. Nous ne considérons dans ce travail que des approches incrémentales pour les raisons évoquées précédemment. Les algorithmes incrémentaux n'ont pas de données au début du processus d'apprentissage. L'algorithme doit mettre à jour ses modèles internes au fur et à mesure que les données arrivent (typiquement, un nouvel état ou un événement de l'environnement). Ainsi, l'algorithme prend continuellement en compte les caractéristiques et les changements du monde dans lequel il évolue.

Nous pouvons distinguer deux types principaux de systèmes classificateurs :

- Les systèmes à base de règles ;
- Le raisonnement à partir de la mémoire.

Des systèmes à base de règles

Les premiers systèmes classificateurs utilisés dans les agents autonomes sont les systèmes à base de règles, par exemple dans l'approche de Holland (1986). Dans ces systèmes, le concepteur doit fabriquer un ensemble de règles qui décrit le comportement de l'agent. Chaque règle met en relation un ensemble de conditions ou prémisses et une action. Pour

qu'une règle s'applique, les prémisses doivent être vraies dans l'état courant de l'environnement. Pour apprendre, l'agent choisit un ensemble de règles applicables dans l'état courant. Ensuite, il utilise une mesure de qualité attachée à chaque règle pour choisir la meilleure. Normalement, l'agent doit avoir une récompense en fonction de son action, comme dans les systèmes d'apprentissage par renforcement. Dans ce cas, il augmente la qualité de toutes les règles sélectionnées précédemment. Par conséquent, au fur et à mesure que le temps passe, la priorité des règles change. Des nombreuses approches ont été proposées pour améliorer cette technique, notamment celle qui concerne l'utilisation d'algorithmes génétiques et du calcul parallèle (voir (Booker, 1989) et (Holmes, 2002)).

Cette approche hérite malheureusement de plusieurs défauts de l'approche d'apprentissage par renforcement, comme le temps de calcul des récompenses ou le temps nécessaire pour apprendre. L'agent a besoin d'un modèle initial (un ensemble de règles) pour pourvoir agir, ce qui nécessite un langage de représentation et beaucoup de connaissances du domaine. Malgré l'élégance des algorithmes génétiques et du calcul parallèle pour l'exploitation de l'espace de recherche, le temps d'adaptation restreint beaucoup son utilisation pratique.

Raisonnement utilisant la mémoire

Les premières applications utilisant la mémoire pour améliorer la performance des agents autonomes ont été développées au début des années 90, parmi celles-ci, les applications de Kozierok & Maes (1993) et Lashkari & Maes (1994). Ces applications ont utilisé l'approche du raisonnement par mémoire, plus connue sous le vocable Memory-Based Reasoning (MBR). Dans le domaine de l'apprentissage automatique, cette approche est connue comme l'apprentissage fondé sur des instances, ou Instance-Based Learning (IBL).

L'idée générale du MBR est que si une certaine action *a* prise dans le passé dans une certaine situation *s* a donné de bons résultats, elle sera utile dans une nouvelle situation *s'* similaire à *s*. Ainsi, la question clé dans le MBR consiste à trouver une mesure de similarité pour comparer les observations. Les méthodes traditionnelles de MBR utilisent le fameux algorithme K-NN (Aha, 1991) avec des mesures classiques de similarité comme la Distance Euclidienne ou la mesure du cosinus. L'approche de MBR est la même que celle du raisonnement à base de cas (CBR) (Riesbeck et Schank, 1989). Cependant, le CBR est un processus plus complexe qui met en place des mécanismes d'analyse et d'adaptation des cas. Cunningham et des collaborateurs discutent les limitations de MBR et le rapport entre MBR et CBR (1994).

Selon eux, MBR est différent du CBR parce que MBR n'utilise pas des connaissances du domaine. Dans CBR, les connaissances du domaine sont souvent utilisées pour la définition des mesures de similarité et pour l'adaptation des cas. En revanche, MBR est purement empirique, fondé sur une grande quantité d'informations, nécessite un grand effort de calcul, et beaucoup d'effort dans la structuration des cas. L'auteur argumente que MBR doit être utilisé dans des activités où la sémantique n'est pas nécessaire. Le problème est que la structuration des cas demande souvent autant d'efforts que la modélisation sémantique du domaine. En ce qui concernent les agents autonomes, les cas représentent les états de l'environnement, et donc, une certaine connaissance du domaine sera nécessaire pour décrire ces cas, le plus souvent sous forme des vecteurs du type attribut-valeur.

Faltings soutient que les mesures de similarité pour des attributs et des cas sont généralement imprécises et demandent un grand effort de calcul (Faltings, 1997). Dans une tentative pour minimiser ces problèmes, l'auteur propose l'utilisation d'outils de modélisation qualitative (de connaissances du domaine). Avec ces outils l'utilisateur/développeur peut fournir au système des modèles capables de diminuer l'ambiguïté des mesures de MBR. Kozierok et Maes (1993) et Lieberman (1997) utilisent une approche similaire dans des interfaces intelligentes. Kozierok considère le feedback de l'utilisateur pour augmenter/diminuer le degré d'adéquation des prédictions faites par un système de MBR dans une application pour l'organisation de réunions. L'agent interface de Lieberman utilise l'interaction avec l'utilisateur pour définir différents degrés d'importance pour les attributs utilisés pour la récupération d'information pour des problèmes de location d'appartements.

La comparaison entre cas est faite souvent avec des mesures pondérées. Le problème majeur est donc le temps de calcul nécessaire pour mettre à jour ces poids au fur et à mesure que les nouvelles informations arrivent. Une des approches utilisées est alors de mettre à jour les informations, non pas à chaque fois qu'une nouvelle information arrive, mais dans des intervalles de temps moins fréquents. C'est le cas par exemple dans le travail de Kozierok et Maes (1993), où l'agent met à jour son modèle une fois par jour pendant la nuit. Dans ce cas, l'agent travaille toujours en décalage par rapport au modèle courant du monde.

La troisième technique d'apprentissage automatique utilisée dans la construction des agents adaptatifs est l'apprentissage par modèles. Celle-ci est discutée dans la section suivante.

L'apprentissage par modèles

Les agents qui utilisent l'apprentissage par modèle essaient de trouver des relations de cause à effet entre leurs actions et les événements de l'environnement. En général, ils utilisent soit des modèles probabilistes soit des modèles logiques dans cette approche.

Maes a proposé un réseau de comportements pour modéliser la procédure de sélection des actions d'un agent autonome dans un environnement dynamique et imprévisible (Maes, 1992). Ses idées originales ont été reprises par la suite dans nombreux travaux, voir (Dorer, 1999). Dans le réseau de comportement, chaque nœud est un comportement (action) possible représenté avec une structure similaire à celle des opérateurs STRIPS (voir section 2.6.3). Chaque nœud est composé par des pré-conditions, post-conditions, une liste *add* de prédicats, une liste *delete* de prédicats (comme dans les opérateurs STRIPS) et une énergie d'activation. À partir de chaque nœud trois types de liens peuvent se dégager:

- Prédécesseur : d'un nœud vers d'autres nœuds qui satisfont des pré-conditions du nœud d'origine. Ex : L'action « mettre la balle dans la boîte » doit avoir au moins deux prédécesseurs : « ouvrir la boîte » et « prendre la balle ».

- Conflictuel : d'un nœud vers d'autres nœuds qui rendrent impossible la satisfaction des pré-conditions (les contraintes). Ex : L'action « mettre la balle dans la boîte » doit avoir un lien conflictuel avec « laisser tomber la balle ».

- Successeur : un lien dans la direction contraire au lien *prédécesseur*.

Le but du réseau est d'accumuler une énergie dans les nœuds qui représentent les actions adéquates. À chaque cycle de temps le processus de transmission d'énergie est réalisé de la manière suivante : (i) tous les nœuds exécutables (pré-conditions vraies) transmettent une activation aux successeurs ; (ii) tous les nœuds non-exécutables transmettent une activation aux nœuds prédécesseurs qui rendent une pré-condition fausse, vraie ; (iii) tous les nœuds (exécutables ou non) diminuent l'activation des nœuds connectés par de liens conflictuels.

Le réseau garde aussi un seuil minimal que les nœuds doivent respecter. Lorsqu'un nœud dépasse le seuil il est supprimé. Au bout d'un certain temps le réseau doit garder seulement des actions qui normalement sont dépendantes les unes des autres. Les avantages principaux de cette approche, selon Maes, sont que le réseau peut modéliser des changements d'objectifs

de l'agent. Elle considère aussi des environnements quantitatifs et des actions multiples. Cependant, en raison de la propagation de valeurs d'activations, la prise de décision tend à demander beaucoup de temps de calcul.

Une autre technique d'évolution des agents autonomes consiste à utiliser la programmation logique inductive (ILP). Benson présente un algorithme capable d'apprendre les modèles d'action de l'agent (Benson, 1995). Ces modèles sont utilisés ensuite pour élaborer une planification. L'algorithme représente les actions sous forme de TOPs (voir la section 2.6.3) composées de conditions, de variables, et d'effets. L'agent réalise des lectures périodiques dans l'environnement. Pour chaque lecture, l'agent produit un ensemble d'exemples positifs ou négatifs en fonction des actions dont les conditions sont satisfiables ou non. Lorsque les conditions d'un TOP sont vraies, une opération de mise à jour est mise en place en utilisant des post-conditions modélisées au préalable. Ensuite, un algorithme ILP extrait, à partir de cette base de données, les relations entre les formules propositionnelles. Ces relations forment alors les plans de l'agent pour lui permettre d'atteindre ses objectifs plus rapidement. Il semble que le processus d'adaptation de l'agent soit très lent, car il nécessite beaucoup d'observations. Du coup, l'apprentissage ne se fait pas en permanence, mais de temps en temps.

2.6.5. Les agents assistants adaptatifs

Dans les sections précédentes, nous avons étudié deux catégories d'agents : les agents *autonomes* et les agents *adaptatifs*. Nous allons voir dans cette section comment les agents dits assistants se placent parmi ces catégories.

Un agent assistant personnel est un agent qui assiste son maître dans la réalisation des tâches quotidiennes. Nous croyons qu'un agent assistant personnel doit, par définition, être autonome. En effet, si l'agent assistant personnel a pour but de diminuer la charge de travail de son maître, il doit donc agir de façon autonome et si possible prendre en charge certaines des activités humaines. Tout d'abord, un agent autonome doit exister dans un environnement dynamique et agir en fonction des changements de l'environnement. Ainsi nous pouvons dire qu'un agent assistant personnel est un agent autonome qui fonctionne dans un environnement dynamique composé principalement d'outils informatiques (généralement distribués) et dont les états changent selon le comportement de son maître et en fonction de l'arrivée de nouvelles informations. Ce raisonnement est présenté dans la Figure 12. Deuxièmement, un

agent autonome doit avoir l'autonomie d'action pour accomplir des objectifs. Dans ce dernier cas, nous pouvons dire également qu'un agent assistant personnel est un agent autonome dont les objectifs sont ceux qui conduisent à la satisfaction de son maître.

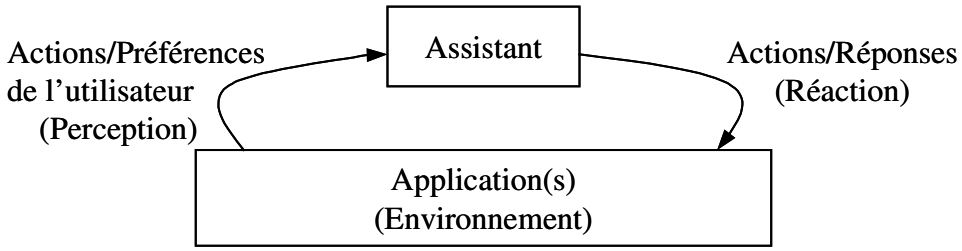

Figure 12 L'assistant comme un agent autonome.

Certains pourront argumenter que dans certaines applications un agent assistant personnel peut éventuellement communiquer avec d'autres agents et ainsi appartenir à la catégorie des agents communicants (voir section 2.6.1). Néanmoins pour nous, le plus important n'est pas le fait de communiquer, mais d'avoir une intention indépendante de l'avis des autres agents. C'est-à-dire, l'autonomie prime sur la communication.

Nous savons tous que dans n'importe quel domaine d'activité humaine les différents niveaux hiérarchiques de la société sont distingués par le pouvoir de délégation. La délégation, à son tour, est (ou devait être) directement proportionnelle à la compétence d'un individu pour réaliser les activités qui lui sont déléguées par son supérieur (si nous faisons abstraction des aspects sociaux comme la rivalité, la méfiance, le mépris, etc). Toutefois, deux individus de différents niveaux hiérarchiques ne peuvent avoir de bonnes relations, s'ils n'ont pas partagé un certain nombre d'expériences, voir (Negroponte, 1997). Ceci dit, l'un et l'autre doivent se connaître mutuellement, et surtout, faire évoluer cette connaissance.

Dans notre problème, l'agent assistant personnel et son maître sont des individus placés à des niveaux hiérarchiques différents. Nous considérons que l'adaptation des connaissances doit se faire de l'assistant vers son maître. Par conséquent, l'assistant doit avoir des techniques d'apprentissage pour apprendre à connaître son maître à partir des expériences partagées. Dans ce contexte, un agent assistant personnel est un agent autonome adaptatif.

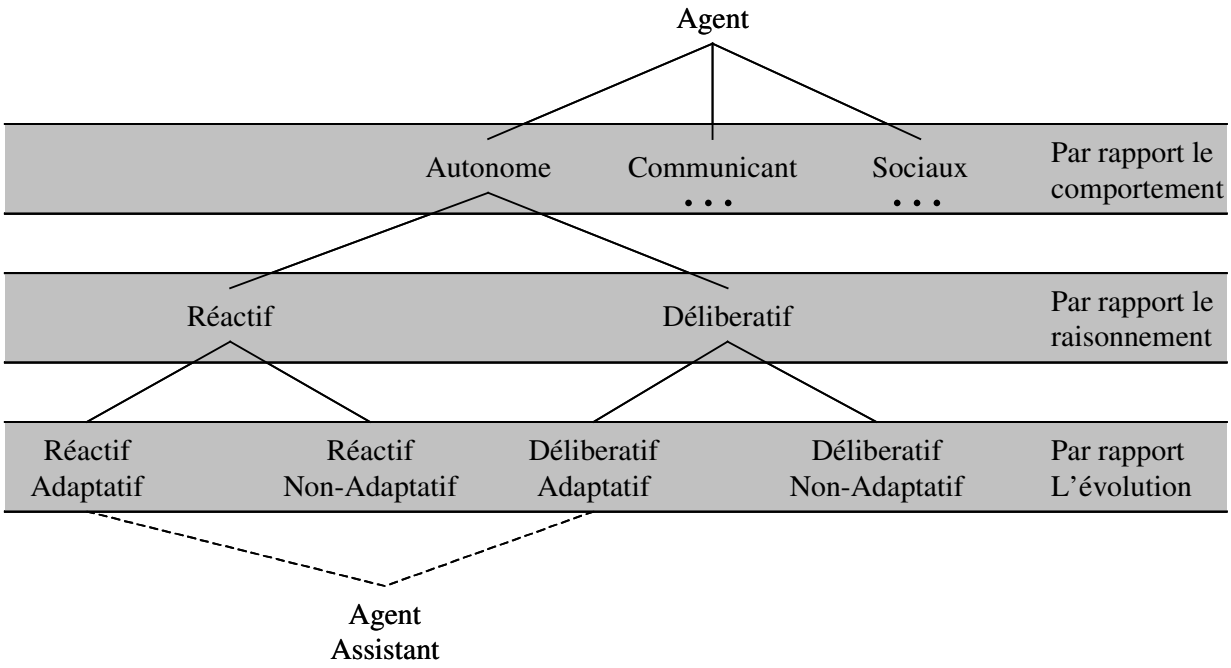

Figure 13 Classification de l'Agent Assistant Personnel.

Dans la Figure 13 nous avons placé l'assistant à la fois comme réactif adaptatif et délibératif adaptatif. Il est vrai que dans la littérature il n'y a pas beaucoup d'architectures d'agent assistant personnel délibératif. L'application de Silva et Demazeau en est une exception (2002). Dans cette application, chaque utilisateur possède un agent agenda qui va essayer d'organiser son créneau horaire (marquer des réunions, répondre à des demandes de réunions). Cette approche est notamment différente de la nôtre, car chez nous l'assistant est spécialisé dans l'interaction avec son maître et non dans la résolution directe de problèmes. En effet, les tâches qui permettent la résolution du problème posé par l'utilisateur sont réalisées par des agents service, ce qui suggère que les stratégies de résolution du problème doivent être dans les agents service et non dans l'assistant. Malgré cela, rien n'empêche qu'un agent assistant personnel délibératif soit mis en oeuvre dans des domaines complexes, où des tâches nécessitent des stratégies variées selon la situation.

Nous avons déjà vu dans ce chapitre une grande variété de méthodes d'apprentissage utilisées dans la construction d'agents adaptatifs. Ces méthodes sont utilisées dans le choix de l'action à faire et dans l'amélioration du mécanisme de choix. Dans l'univers des agents assistants nous distinguons deux types d'applications où ces approches peuvent être mises en oeuvre : les applications *mono-tâche* et les applications *multi-tâches*.

Les applications mono-tâche

Dans ces applications le but du système consiste à se spécialiser dans une seule tâche clairement définie. Le principal modèle à représenter et à faire évoluer est celui des *préférences* de l'utilisateur, même si d'autres modèles (par exemple, celui du monde, celui du domaine, etc.) peuvent exister. Dans ce mémoire nous avons déjà vu comment ces modèles peuvent co-exister. Des applications de ce genre sont, par exemple, l'organisation des e-mails personnels, l'organisation des documents personnels, la navigation sur le web, le filtrage de documents, les agents experts, la recherche d'information. Cependant, dans toutes ces applications l'agent continue à avoir besoin de modèles, notamment celui de son maître. Par conséquent, l'assistant peut utiliser la plupart des méthodes d'apprentissage vues précédemment pour faire évoluer ces modèles et optimiser sa performance au fur et à mesure que le temps passe.

Les applications multi-tâches

Dans les applications multi-tâches l'assistant peut travailler à deux niveaux : il peut s'occuper du *comportement* de l'utilisateur en essayant de fabriquer un modèle de sélection des actions prises. Ensuite, ce modèle d'actions sera utilisé pour gérer l'interaction entre les différentes applications ; la deuxième perspective est celle vue précédemment où l'assistant doit se spécialiser dans la modélisation des *préférences* de l'utilisateur pour chaque application.

Des applications de ce genre sont notamment moins nombreuses que celles des applications mono-tâche. Des exemples classiques sont ceux des assistants personnels pour la gestion d'ordinateur (fichiers, documents, communication, etc.) et les applications des assistants personnels mobiles (téléphones portables et palm tops). Cela est dû principalement au manque de mécanismes standards capables de faire dialoguer les différentes applications. Aujourd'hui un effort considérable est fait dans le but de développer des protocoles et des langages de représentation qui permettent la communication entre différentes applications, voir par exemple CORBA (Common Object Request Broker Architecture), UML (Unified Modeling Language), XML (eXtended Markup Language) de l'OMG (voir référence OMG). Certains domaines, comme par exemple celui des outils collecticiels, regroupent les outils dans un même environnement de travail. Malgré l'application potentielle des assistants dans ces outils, le développement d'agents assistants pour améliorer l'efficacité des collecticiels est

encore un terrain inexploré. Nous avons proposé d'avancer dans cette direction (Enembreck et Barthès, 2002).

2.7. Les problématiques

À partir des études réalisées dans ce chapitre nous avons identifié deux problématiques auxquelles nous essayerons d'apporter une solution dans ce mémoire :

- L'absence de certains mécanismes liés à l'architecture proposée par Ramos pour la construction d'un agent assistant personnel ;
- Le manque de méthodes d'apprentissage adaptées à la conception d'agents assistants personnels adaptatifs.

2.7.1. L'architecture d'agent assistant personnel

Malgré la cohérence des modèles proposés par Ramos discutés dans la section 2.4, cette architecture demande à être complétées car

- Les modules sont décrits seulement au niveau conceptuel ;
- L'interaction avec l'utilisateur n'est pas clairement définie ;
- L'architecture peut rencontrer des problèmes lors d'un passage à l'échelle.

La description de haut niveau donnée par Ramos pour les éléments de l'assistant ne favorise pas un mécanisme de contrôle précis, ce qui rend le fonctionnement de l'agent difficilement compréhensible. Nous discuterons dans ce mémoire plusieurs mécanismes et représentations qui permettent de remplir les boîtes proposées par Ramos. Nous avons conçu dans le Chapitre III, par exemple, le module d'interface (voir Figure 2) sous forme d'un système de dialogue qui recquiert la représentation de trois autres modules (Ontologie, Tâches et Maître) ainsi qu'un moteur de dialogue qui décrit nettement l'interaction entre ces modules (le contrôle).

Ramos discute un mécanisme de dialogue avec l'utilisateur fondé sur des graphes d'états. Cependant, chaque état du dialogue est codé en dur dans le système. Cette technique rend l'assistant difficilement utilisable lorsqu'une nouvelle compétence est rajoutée car le dialogue doit être recodé mannuellement. Par ailleurs, cette technique rend le dialogue moins ouvert et moins naturel pour l'utilisateur. Nous proposons donc dans le Chapitre III un modèle

générique de dialogue capable de s'adapter à de nouvelles tâches et de dialoguer de façon plus naturelle.

Finalement, le troisième problème avec l'architecture de Ramos est qu'elle ne prend pas en compte la complexité de développement d'un assistant capable de fournir plusieurs services simultanément, c'est-à-dire, possèdant plusieurs compétences. Nous pouvons constater dans les chapitres IV et VI que même des services simples peuvent nécessiter des mécanismes complexes et lourds en calculs. Ainsi nous soutenons qu'un assistant sera difficilement capable d'exécuter lui-même des services, c'est-à-dire, avoir des *compétences*. *A contrario*, il devra déléguer ses tâches à d'autres agents. Nous discutons donc dans le chapitre VII la création de Systèmes Multi-Agent Personnalisés (SMAP) et introduisons une définition formelle de SMAP.

2.7.2. Les agents assistants personnels adaptatifs

Nous avons passé en revue dans ce chapitre un nombre de techniques destinées à la mise en place des mécanismes d'autonomie et d'adaptation des agents autonomes. Parmi celles-ci nous retrouvons les techniques des agents réactifs, délibératifs, l'apprentissage par renforcement, les systèmes classificateurs (règles et MBR) et l'apprentissage par modèles (réseaux de comportements et la programmation logique inductive). Nous avons indiqué un certain nombre de problèmes concernant les techniques d'adaptation, comme par exemple, la complexité, le temps nécessaire à l'apprentissage et le mauvais usage des ressources (temps de calcul et mémoire). Dans le cadre d'un agent assistant personnel ces problèmes peuvent s'avérer dramatiques :

La complexité : le dévéloppement de l'agent ne doit pas demander beaucoup d'effort de modélisation *a priori* (comme dans les systèmes à base de règles, l'apprentissage par modèles, la programmation logique inductive et la définition des mesures de similarité spécifiques dans MBR) car cela implique que l'agent est spécialisé et ne peut être mis en place facilement dans une nouvelle application ;

Le temps nécessaire à l'apprentissage : l'agent ne doit pas passer beaucoup de temps à apprendre (comme dans l'apprentissage par renforcement et les réseaux de comportements), car il doit être capable d'identifier les préférences du maître rapidement sans avoir besoin de nombreux feedbacks ;

Le mauvais usage des ressources : l'agent ne doit pas avoir un temps de réponse qui se dégrade au fil du temps ni avoir besoin d'une mémoire énorme (comme dans MBR) pour stocker les informations dynamiques qui arrivent.

Dans ce mémoire nous avons développé une nouvelle méthode d'apprentissage qui tient compte de ces problèmes. Avant de présenter la méthode dans le chapitre V nous présentons dans les chapitres III et IV une étude sur les modèles internes d'un agent assistant susceptibles d'un processus d'évolution et l'effet que ce processus peut avoir sur le comportement de l'agent. Pour cela nous discutons les structures internes des assistants dans des applications qui ont été développées au laboratoire.

2.8. Conclusions

Dans ce chapitre nous avons mentioné plusieurs domaines de recherche qui fournissent les concepts de base pour le développement d'un agent assistant personnel adaptatif, notamment celui des agents intelligents, de la modélisation de l'utilisateur et de l'apprentissage automatique. Nous avons vu quelles sont les structures internes qui peuvent doter l'assistant d'un comportement intelligent.

Nous avons essayé de situer les agents assistants parmi les catégories les plus courantes rencontrées dans la littérature des systèmes d'agents : les agents *autonomes* et les agents *autonomes adaptatifs*.

Nous nous sommes appuyés sur l'idée qu'un agent assistant doit être autonome et adaptatif. Ensuite, nous avons présenté les mécanismes les plus couramment utilisés dans la conception des agents qui appartiennent à ces catégories. Enfin, nous avons conclu le chapitre avec une analyse sur les problèmes liés à la conception d'un agent assistant adaptatif.

3 Adaptation et personnalisation de l'interaction

Dans un certain nombre d'applications associées aux agents assistants l'aspect évolutif et adaptatif est quasiment négligé. Dans ces applications, l'agent doit généralement avoir une bonne représentation des connaissances du domaine pour assurer une interaction convaincante. Cela est particulièrement vrai, pour des assistants dotés d'interfaces intelligentes. Comme dans ces applications les connaissances sont plutôt statiques et l'assistant est isolé, la seule possibilité d'adaptation concerne l'interaction avec le maître.

Par ailleurs, la complexité des mécanismes d'interaction et de représentation des connaissances du domaine rend l'adaptation du système extrêmement compliquée. Par conséquent, nous ne trouvons pas dans la littérature beaucoup d'agents assistants adaptatifs capables de dialoguer avec l'utilisateur ou de mettre en œuvre des interactions complexes.

Dans ce chapitre nous étudions la relation entre complexité d'interaction et difficulté d'adaptation pour la conception d'un agent assistant.

3.1. Les assistants, les interfaces adaptatives et les systèmes hypermédias adaptatifs

Dans un certain nombre de domaines, notamment le domaine pédagogique, nous trouvons de nombreux travaux concernant les interfaces adaptatives. De Bra et collaborateurs (1999) ont publié une étude générale sur ce sujet. Dans cette section nous essayons d'analyser un certain nombre de points communs entre ces travaux et surtout nous critiquons l'utilisation du terme

« adaptative » dans certaines applications dites « interfaces adaptatives » ou « systèmes hypermédias adaptatifs ».

De nombreuses applications ont été proposées dotées d'interfaces qui s'affichent différemment selon le profil de l'utilisateur (des interfaces adaptatives). Ces applications conservent un modèle d'utilisateur acquis au préalable, voire même dynamiquement, pour aider l'utilisateur à accomplir plus facilement sa tâche en réduisant le travail d'analyse des informations jugées peu importantes. Ces modèles peuvent varier énormément selon leur contenu. Par exemple, la plupart des systèmes bancaires en ligne mettent en œuvre un petit formulaire avec certaines questions pour définir le niveau socio-économique des clients pour ensuite proposer des produits d'investissement personnalisés. Dans le domaine pédagogique, de nombreuses applications ont été proposées pour personnaliser la présentation du matériel pédagogique aux étudiants (Brusilovsky et Su, 2002).

Akoulchina (1998) soutient que deux techniques peuvent être utilisées pour l'adaptation des interfaces : *pré-programmée* ou *expérimentale*.

Les applications *pré-programmées* ont normalement en commun un modèle de comportement câblé du système qui met en relation certaines caractéristiques du comportement de l'utilisateur et certaines informations ou formes de présentation. Ce modèle de comportement peut être réalisé à l'aide de contraintes ou encore de règles décrivant le comportement des utilisateurs, buts ou niveaux d'expertise, mais il est souvent statique. Ainsi, tous ces éléments doivent être prévus au préalable et ne changent pas. Par conséquent, l'utilisation du terme « adaptation » est inapproprié dans ce genre d'application, car la définition de l'adaptation ne désigne pas, dans son essence, un processus bien déterminé au sens de l'Encyclopedia Universalis (1989). Étant donné que l'aspect cognitif du système, c'est-à-dire, le modèle de comportement des utilisateurs, est rigide et que le système n'est pas capable de le mettre à jour, aucune adaptation n'est obtenue au niveau agent. Même si les informations sur les utilisateurs sont modifiées et la présentation des informations change, le modèle cognitif qui fait la liaison entre ces deux éléments ne change pas.

Il est donc difficile d'imaginer comment une application peut être adaptative sans disposer d'un algorithme d'apprentissage.

L'adaptation *expérimentale* est souvent mise en forme par un agent assistant intelligent qui accompagne l'utilisateur au long de ses activités et qui est capable d'assimiler au fil de l'eau

ses préférences, ses compétences et ses objectifs. Ces informations sont organisées sous forme d'un modèle informatique qui peut être utilisé par l'agent pour guider l'utilisateur vers la solution des problèmes et présenter des informations de façon personnalisée. L'apprentissage de ces modèles se fait normalement avec des méthodes originaires des domaines d'*apprentissage automatique* et d'*agents autonomes adaptatifs*, car l'agent doit être capable d'améliorer ses connaissances à mesure qu'il intéragit avec l'utilisateur et doit être le plus autonome possible pour réduire au minimun le besoin de feedback. Puisque l'adaptation expérimentale concerne la création d'un modèle dynamique de l'utilisateur, on peut affirmer qu'elle est une application des techniques d'un autre domaine de recherche : la *modélisation de l'utilisateur*. Nous discutons ce domaine en détails dans le Chapitre IV.

Dans la suite de ce mémoire nous considérons que les applications pré-programmées sont hors contexte et nous ne considérons que l'*adaptation expérimentale*. Pour garder une coherence terminologique nous faisons l'hypothèse que le terme *adaptation* concerne l'*adaptation expérimentale*.

Nous verrons dans la suite de ce chapitre comment l'adaptation peut être mise en œuvre pour améliorer l'interaction entre un agent assistant et l'utilisateur.

3.2. L'interaction par dialogue et l'adaptation

Les agents assistants dotés d'une interface en langage naturel sont de plus en plus utilisés dans une multitude de domaines, par exemple, dans des applications de traduction speech-to-speech (Kipp et al, 1999), la vente de titres de transport (Allen et al., 1996) (Seneff et Polifroni, 2000), dans les systèmes tuteurs (Rickel, 2000), la navigation sur le web, les assistants mobiles, etc. Ces systèmes sont souvent conçus sous forme de systèmes de dialogue. En général, dans ces applications, le domaine est représenté dans une ontologie sous forme d'un réseau sémantique ou d'un système de frames. Le noyau du système est un mécanisme de coordination de dialogue générique qui travaille sur une base de connaissances spécifique du domaine. Cette approche a été utilisée par Rich et ses collaborateurs (Rich et al, 2001) qui utilisent le noyau du système de dialogue COLLAGEN, voir (Rich et Sidner, 1997), dans quatre applications complètement différentes : l'aide à l'installation/configuration d'un magnétoscope, l'aide à l'édition d'une base d'objets quelconques, l'aide à l'opération du moteur d'une turbine à gaz dans un simulateur, et l'aide au contrôle d'un chauffage domestique électronique.

Par ailleurs, Flycht-Eriksson (1999) a classé les systèmes de dialogue en deux catégories : les systèmes de question/réponse et les systèmes orientés tâches. Les premiers concernent généralement les systèmes de consultation par exemple pour obtenir des informations touristiques ou des information de voyage. Les seconds travaillent comme un guide qui conduit un dialogue avec l'utilisateur en vue d'exécuter une tâche précise. Ces tâches vont de la recherche de documents jusqu'à des activités beaucoup plus complexes comme le contrôle d'un agenda personnel.

Généralement, l'adaptation d'un système de dialogue consiste à mettre à jour le modèle de l'utilisateur de façon que l'interaction avec le système soit personnalisée. Dans ce cas, la personnalisation peut être accomplie à différents niveaux de complexité. Zukerman et Litman (2001) soutiennent que la manière la plus simple d'adaptater le système consiste à définir différents niveaux d'expérience et de stéréotypes d'utilisateurs. Par exemple, le dialogue avec un expert demande un niveau différent d'explications qu'un dialogue avec un débutant.

Une autre approche consiste à modéliser intensivement des bases de connaissances complexes qui représentent les plans, les croyances, les compétences, et les buts des utilisateurs comme dans les travaux de Kass et Finin (1988), Sarner et Carberry (1992) et Kobsa (2001). Ces informations sont ensuite utilisées pour adapter le dialogue. Une telle technique présente les mêmes problèmes que les interfaces adaptatives pré-programmées discutées dans la section précedente, car la connaissance du système est figée et par conséquent l'application n'est pas adaptative. Par ailleurs, un effort considérable est necéssaire pour modéliser toutes ces connaissances.

Dans un autre niveau, un système de dialogue peut incorporer un mécanisme d'apprentissage automatique pour acquérir des règles de comportement vis-à-vis des utilisateurs et produire les stéréotypes dynamiquement.

Les modèles de l'utilisateur dans les systèmes de dialogue peuvent être utiles principalement dans les cas suivants :

- *Pour optimiser le dialogue* : le dialogue pour l'exécution de tâches peut être très ennuyeux, car l'utilisateur doit fournir beaucoup d'informations. Le système doit donc être capable de prédire certaines valeurs pour diminuer le travail de l'utilisateur ;

- *Pour augmenter la précision du système* : les informations données par les utilisateurs sont souvent ambiguës et imprécises. Un mécanisme de mémoire peut donc être utile pour la résolution de conflits et l'acquisition de feedback de la part de l'utilisateur et peut améliorer les résultats du système.

Un exemple d'architecture de génération des préférences de l'utilisateur dans un système de dialogue a été présenté par Elzer et collaborateurs (1994). Les auteurs prétendent qu'un système de consultation doit prendre en compte les préférences de l'utilisateur. Les préférences sont utilisées pour proposer de nouvelles solutions dans le système. Le modèle de l'utilisateur peut être acquis, selon Elzer, dans deux cas de figure : (i) à partir des phrases et (ii) à partir du contexte de la conversation. Dans le premier cas, les préférences sont *explicites* (par exemple, « j'aime le cours d'IA »), *indirectes* (par exemple, « je veux suivre le cours de **Math** ») ou *partielles* (par exemple, « j'aime peut-être le cours d'IA »). Dans le deuxième cas (ii) le système identifie les préférences au fil de l'eau selon quatre possibilités : l'utilisateur rejette une solution proposée (par exemple, « je n'ai pas aimé le cours d'IA »), l'utilisateur donne des préférences dans la spécification du problème (par exemple, « je veux un cours pas trop long »), l'utilisateur donne des préférences sans que le système ne le lui ait demandé ou, finalement, l'utilisateur donne une réponse à une question. Le système représente les préférences identifiées sous forme de formules atomiques. Ensuite, il ajoute des niveaux de confiance aux solutions (très fort, fort, modéré, faible et très faible) et fait alors des propositions personnalisées en fonction des poids des actions possibles.

Zukerman et Litman (2001) soutiennent que l'utilisation de techniques d'apprentissage dans des systèmes de dialogue est un domaine récent qui doit encore être approfondi. Malgré quelques applications de réseaux de Bayes dans la reconnaissance de plans, la génération d'arguments et la modélisation du dialogue, ce domaine reste encore peu exploré car nous ne connaissons pas les limites ni la potentialité de l'apprentissage automatique dans les systèmes de dialogue.

Toutefois, nous croyons que l'apprentissage basé sur les expériences passées peut être utilisé dans la construction des préférences de l'utilisateur. Ces préférences permettent ensuite d'améliorer le dialogue avec le système.

Nous ne comparons pas ici les nombreux assistants capables de dialoguer car la plupart de ces systèmes ne présentent pas un comportement évolutif ni adaptatif.. En revanche, dans la

section suivante nous présentons un prototype que nous avons développé au cours de la thèse, qui nous a permis d'identifier les éléments nécessaires à la conception d'un agent de dialogue mais aussi d'analyser l'aspect évolutif (s'il existe) de chaque élément.

3.3. La conception d'un agent assistant capable de dialoguer

Nous prétendons que pour envisager une interaction adéquate, les techniques de Langage Naturel et IHM (Interface Homme-Machine) doivent être combinées.

En effet, les interfaces classiques sont efficaces pour des problèmes simples où des formulaires peuvent afficher toutes les rubriques à remplir. En revanche, le Langage Naturel est envisageable dans le cas d'interactions complexes. Par exemple, considérons la tâche d'envoi de courriels. L'utilisateur demande au système : *envoyez un e-mail à carlos@utc.fr*. À l'évidence, ceci n'est pas une bonne utilisation du Langage Naturel. Le Langage Naturel peut être plus intéressant dans le cas suivant : *envoyez un e-mail au père de Cesar*. Nous pouvons constater que le Langage Naturel est envisageable dans des applications où des connaissances supplémentaires peuvent être utiles à la solution du problème.

Dans cette section nous étudions comment les modèles internes d'un agent assistant peuvent être organisés pour la conception d'un module de dialogue et nous mettons également en évidence la possibilité de faire évoluer ces modèles. Pour cela nous avons développé un module de communication fondé sur le Langage Naturel. Ce module a été conçu comme un système de dialogue ayant les composants suivants : un modèle de dialogue, un modèle de l'utilisateur, un modèle du domaine et un modèle de tâches.

Les modèles du domaine et de tâches ont été représentées sous forme d'ontologies dérivées du projet AACC[6]. En revanche, le modèle de l'utilisateur est spécifique du dialogue. Le module de dialogue peut être utilisé par l'utilisateur pour exécuter des tâches et demander des informations. Les paragraphes suivants détaillent l'architecture et le fonctionnement du système que nous avons développé.

[6] Le projet AACC (Agents d'Aide à la Conception Coopérative) est realisé en collaboration avec le laboratoire HEUDIASYC et le laboratoire LaRIA de l'Université Picardie Jules Verne.

3.3.1. Le système de dialogue

Nous avons suivi la même approche générale que Flycht-Eriksson (1999). L'auteur décrit un système de dialogue comme une application modulaire où un moteur de dialogue coordonne plusieurs modèles :

- le modèle de dialogue ;
- le modèle de tâches ;
- le modèle du profil d'utilisateur ;
- le modèle du domaine.

Dans cette section nous discutons comment nous avons mis en place tels modèles dans notre système.

3.3.2. Le modèle de dialogue

Le modèle de dialogue sert à coordonner le dialogue et à identifier les questions à poser à l'utilisateur. La tâche correspondante est appelée *coordination de modèle de dialogue*, *gestion de dialogue* ou encore *coordination* de *dialogue*. Toutefois, il n'y a pas de consensus sur la façon dont la gestion du dialogue doit être mise en œuvre. Habituellement, un automate à états finis est employé pour modéliser les états du dialogue. Chaque nœud du réseau représente une question normalement liée à une propriété particulière d'une tâche (Chan Lam, 1979). Cependant, les transitions changent d'un système à un autre selon le comportement désiré. Par exemple, certains systèmes laissent l'utilisateur entrer l'information sur plusieurs propriétés en même temps, d'autres lui permettent de naviguer dans les états passés pour corriger des erreurs, etc. Deux techniques principales sont employées pour réaliser les transitions : les actes de langage et l'utilisation de connaissances.

1 User: Send a mail to Marco for me.
2 System: Who are the Carbon Copy receivers?
3 User: What is Leila's address?
4 System: The address of Leila is "25 rue de Paris".
5 System: Who are the Carbon Copy receivers?
6 User: none
7 System: What is the subject?
8 User: Ohh I'm sorry I made mistake!
9 System: Who are the Carbon Copy receivers?
10 User: Mary's husband.
11 System: What is the subject?
...

Figure 14 Un dialogue ouvert.

Les transitions fondées sur des actes de langage (Searle, 1969) attachent des classes aux expressions entrées par les utilisateurs. De telles classes (inform, demand, confirm, answer, etc.) ont une sémantique prédéfinie et indiquent comment chaque expression doit être interprétée. Allen et collègues (1996) ont employé une technique où une expression peut produire plusieurs actes de langage. Par exemple, l'expression « Okay, maintenant je prends le dernier train vers Albany » peut être décomposée en trois actes du langage : CONFIRM/ACKNOWLEDGE ("Okay"); TELL ("maintenant je prends le dernier train ") et REQUEST ("vers Albany").

Les transitions fondées sur les connaissances n'utilisent pas d'actes de langage. Elles sont directement mises en place sous forme de règles ou de scripts qui indiquent le chemin à suivre pendant le dialogue (voir par exemple (Chan Lam, 1979)). Dans le système Mercury de Seneff et Polifroni (2000) le langage Genesis est utilisé dans la construction d'une table de règles ordonnées. Chaque état de dialogue possède un sous-ensemble de règles adaptées aux possibilités de l'utilisateur.

Nous pouvons observer que les systèmes de dialogue fondés sur des actes de langage demandent moins d'efforts de développement. En effet, puisque la sémantique des actes du langage est prédéfinie, le dialogue peut être réutilisé pour différentes tâches et dans différents systèmes, simplement en changeant le modèle de tâche et la connaissance du domaine. En revanche, à la différence des systèmes d'actes de langage où une bonne partie de la sémantique est définie dans les actes de langage, dans les systèmes de dialogue fondé sur les connaissances, tous les besoins sémantiques doivent être définis pour chaque état du dialogue, ce qui peut devenir extrêmement complexe (200 règles pour le système Mercury (Seneff et

Polifroni, 2000)). En effet, il est très difficile de prévoir toutes les actions possibles des utilisateurs afin de les coder. En conséquence, les systèmes de dialogue basés sur les connaissances sont strictement dépendants du domaine et sont difficiles à réutiliser pour différentes tâches et dans différents domaines.

Notre approche emploie un système d'actes de langage pour les communications entre l'utilisateur et l'agent assistant personnel. L'utilisateur entre des questions et des requêtes, ensuite l'agent assistant personnel entame un dialogue pour atteindre un état dans lequel une action sera déclenchée. Les états de dialogue sont des nœuds d'un graphe de dialogue et la plupart des actes de langage sont disponibles pour n'importe quel nœud. Examinons par exemple le dialogue de la Figure 14. Dans les lignes 1-5, l'utilisateur demande l'envoi d'un message et le système demande des informations complémentaires. L'utilisateur pose une nouvelle question pendant le dialogue, le système lui répond et revient au niveau précédent. Pour accomplir ceci, le système garde une pile d'états. Quand une nouvelle tâche est demandée le système conserve dans une pile un certain nombre d'états correspondants au nombre de propriétés demandées pour accomplir la tâche. Quand une propriété est renseignée avec succès, le système la marque comme dépilée. Cette stratégie permet également à l'utilisateur de revenir à des états précédents (Figure 14 lignes 5-10).

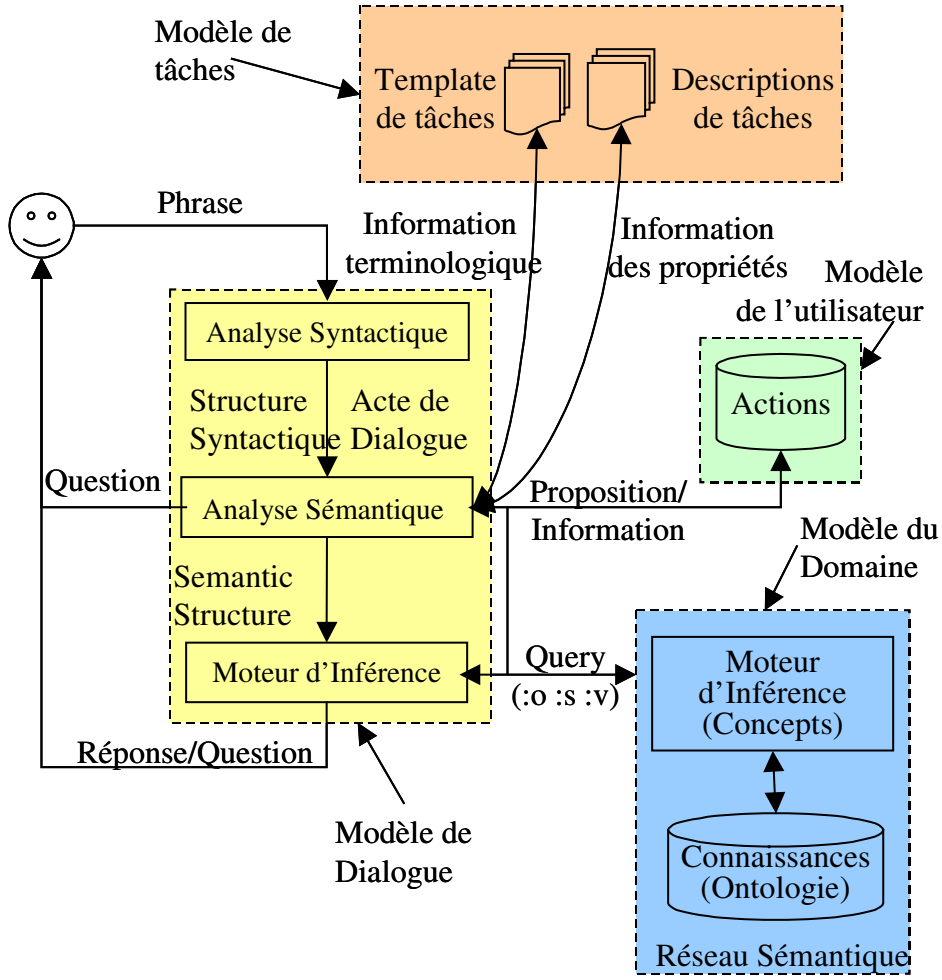

Figure 15 Architecture du système.

Les dialogues sont utilisés pour lancer l'exécution de tâches, comme par exemple, l'envoi de courrier et la localisation de documents ou encore l'obtention d'aide. Ainsi, nous pouvons traiter les deux formes de dialogue : le dialogue orienté tâches et le dialogue de type question/réponse. La Figure 15 illustre l'architecture du système. Quand le système reçoit une question ou une information simple, l'analyseur syntactique produit une représentation syntactique. Cette représentation indique la structure grammaticale de la phrase (expression verbale, expression nominale, locution prépositive, etc.). L'analyseur sémantique utilise cette structure pour établir des demandes. Le rôle de l'analyseur sémantique est d'identifier des objets, des propriétés, des valeurs et les actions dans la structure syntactique en utilisant la hiérarchie et les relations entre objets définies dans le modèle du domaine. Cette information est utilisée pour créer une requête formelle. Finalement, le moteur d'inférence utilise la requête formelle résultante pour rechercher l'information demandée et la présenter à l'utilisateur. La Figure 16 montre comment les expressions sont transformées par le système.

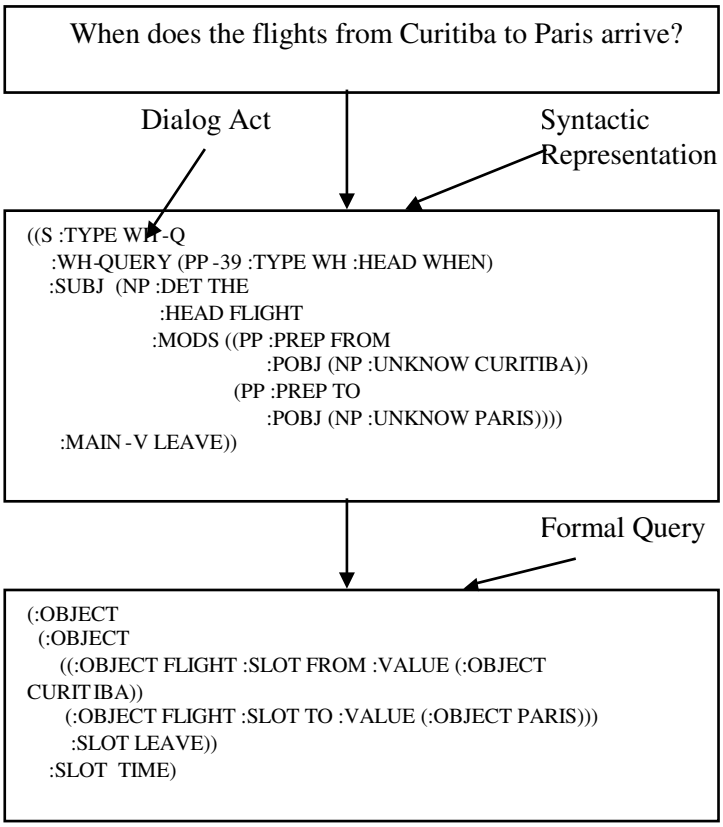

Figure 16 Les structures internes.

Toutes les fois qu'un dialogue orienté vers des tâches démarre, l'analyseur sémantique essaie d'abord de déterminer si une tâche connue est concernée. Si c'est le cas, il vérifie les propriétés fournies dans la requête initiale et poursuit le dialogue pour acquérir les informations nécessaires pour exécuter la tâche. Pour identifier la tâche et les propriétés, l'analyseur recherche l'information dans les modèles de tâches. La stratégie récursive de pile permet à l'utilisateur d'employer à tout moment des relations et des concepts définis dans le modèle du domaine (voir la Figure 14, ligne 10).

Puisque dans notre système la sémantique des actes de dialogue est rigide nous pouvons dire que le modèle de dialogue n'est pas adaptatif. Cependant, le déroulement du dialogue ne dépend pas seulement du modèle de dialogue mais il doit prendre en compte également les informations concernant le modèle de l'utilisateur. Nous discutons comment cela peut être accompli avec une approche très simple dans la section 3.3.6.

3.3.3. Comment interpréter les expressions de l'utilisateur !

Ramos (2000) a employé une technique de langage libre pour analyser les expressions de l'utilisateur. Dans son approche, il n'y a aucune structure linguistique et l'identification est

faite en utilisant des mots-clés pris dans une ontologie. Pour des expressions plus complexes cette technique présente certaines limites parce que des mots sont employés comme index de concepts, mais un concept peut être exprimé de manières très différentes. Par exemple « Joseph » peut être « le père de Helena » ou « le patron » ou d'autres choses encore. De façon plus courante, les systèmes de dialogue s'appuient souvent sur des grammaires. En utilisant des grammaires, les expressions sont traduites en structures syntactiques et peuvent être classées selon des actes de langage. Abel (2000) utilise ces actes de langage pour construire une application traditionnelle avec fenêtres et menus sans employer le langage naturel. Des actes de langage sont directement intégrés à l'interface et sont employés implicitement par l'utilisateur.

Dans notre approche, nous employons une grammaire anglaise simple. Nous avons développé une base de règles grammaticales étendue à partir du travail d'Allen (1986) (voir l'Annexe II), où chaque règle se rapporte à un seul acte de dialogue (par conséquent, nous ne pouvons pas interpréter des expressions composées comme dans (Allen et al., 1996)). Allen et ses collaborateurs (1996) attachent une règle sémantique à chaque règle syntactique et le résultat de l'évaluation des expressions fournies par l'utilisateur provient de l'évaluation de ces structures. Nous avons divisé les transformations syntactique et sémantique en deux étapes suivant en cela McRoy et Ali (1999).

Ici, les analyses syntactique et sémantique ont des caractéristiques particulières qui diffèrent des systèmes plus traditionnels. En effet, les systèmes de dialogue traditionnels travaillent dans des domaines fermés, signifiant que toutes les entités, tous les objets et dispositifs sont déjà représentés et ne changent pas aux étapes ultérieures. Cependant, si l'on considère la tâche « Recherche de Documents », où l'utilisateur peut rechercher des documents par le nom d'auteur, le nom du rédacteur, la date de publication, cette tâche utilise en général les services fournis par un agent de gestion de documents ou un outil de collaboration qui contrôle les documents d'un groupe de personnes. Chaque personne peut stocker ses propres documents et autoriser un accès public. Dans ce scénario, le nom des auteurs n'appartient pas à un ensemble limité et fixe de valeurs. Par voie de conséquence, le système de dialogue doit fonctionner même si on lui demande des documents d'auteurs inconnus. Pour résoudre ce problème pendant l'étape d'analyse syntactique, nous avons introduit un cas particulier d'expression nominale qui appartient à la classe « UNKNOWN ». Cette classe est attachée aux éléments qui n'appartiennent pas à la grammaire. Nous nous servons de l'hypothèse de monde fermé

pour identifier les éléments de cette classe: si un élément ne peut pas être classé dans une classe grammaticale précise il sera classé « UNKNOWN ».

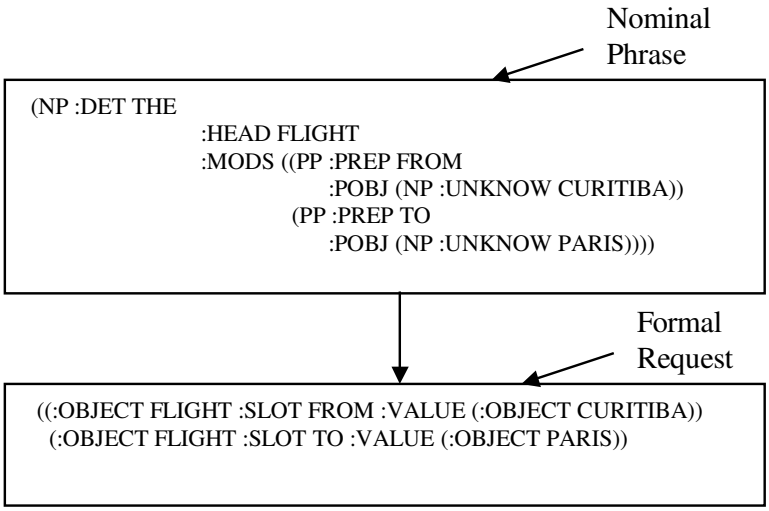

Figure 17 Une requête formelle.

À la différence d'Allen (1986) ou de Mcroy et d'Ali (1999) nous n'employons pas de règles sémantiques prédéfinies explicites d'interprétation. En effet, ceci peut être un processus très fastidieux, parce que de nombreuses règles sont nécessaires. En revanche, nous avons mis en application un algorithme qui analyse la représentation syntactique et l'ontologie du domaine et produit des requêtes bien formées. L'algorithme emploie des locutions nominales et prépositives pour localiser les objets et les propriétés connus. Pendant l'analyse sémantique le système peut demander à l'utilisateur une confirmation ou demander des informations complémentaires si des conflits se produisent. Sur la Figure 17 nous pouvons voir un exemple d'une requête formelle créée à partir d'une structure syntactique. L'idée consiste à créer une liste de conditions utilisées pour filtrer des objets. Le procédé de filtrage renverra des objets satisfaisant toutes les conditions présentes dans la requête. Chaque expression nominale se compose d'un élément cible, par exemple « Flight » dans la Figure 17, et d'une liste de modifieurs. La situation la plus simple se produit quand les modifieurs sont liés aux propriétés de la cible (la Figure 17). Dans ce cas, nous évaluons chaque modifieur pour construire une condition (Figure 18 (a)). Puisqu'une locution prépositive contient normalement une expression nominale, la fonction *eval-PP* dans la Figure 18 (b) emploie la fonction initiale *eval-NP* de la Figure 18 (a). D'autres conditions ne sont pas incluses dans *eval-PP*. Si l'on considère par exemple la phrase nominale « the weight of the luggage », la structure syntactique produite est :

« (NP :DET THE:HEAD WEIGHT:MODS ((PP :PREP OF :POBJ (NP :NOUN LUGGAGE)))) »

pour laquelle la cible est « WEIGHT». La cible est alors une propriété de l'objet dénoté par la locution prépositive. Toutes ces conditions sont prévues dans *eval-PP*.

```
Function eval-NP (NP)
  L = nil  // conditions list
  Target = target object of NP
  If NP has no modifiers Then
        return (:object Target)
  else  // NP have modifiers (properties)
        foreach Modifier M
            insert eval-PP(Target, m) to L
        return L
  End if
End Function
```

(a)

```
Function eval-PP (Target, PP)
  POBJ = eval-NP(PP)
  PREP = target preposition of PP
  If PREP is a slot of Target Then
     return (:object Target :slot Prep :value POBJ)
  else . . .
  End if
End Function
```

(b)

Figure 18 Les fonctions utilisées pour l'analyse sémantique.

L'analyse sémantique est complétée par une analyse linguistique de l'expression, où nous essayons d'identifier si une action (par exemple « leave ») ou un certain modifieur général (par exemple « time » (quand) ou « quantity ») est demandé. Une liste de verbes dénotant des actions et des modifieurs est employée pour accomplir cette tâche.

Enfin, le moteur d'inférence prend la requête formelle résultante et opère le filtrage. La question est une conjonction de requêtes atomiques. Le format de chaque requête peut être « (:Object O :slot S :value V) » pour la recherche d'un objet ou « (:Object O :slot S) » pour la recherche de la valeur d'une propriété. « O » et « V » peuvent être des structures récursives complexes comme indiqué sur la Figure 16. Le résultat pour la question représentée sur la Figure 16 est une liste de valeurs à présenter à l'utilisateur.

3.3.4. Coordination du dialogue

La coordination du dialogue dépend du type d'expressions dénotées par des actes de langage. Schank et Abelson (1977) ont proposé une catégorisation des messages. Pour nous, les catégories suivantes sont importantes :

- Assertive : le message qui affirme quelque chose ou donne une réponse (par exemple « Paul est professeur d'IA à l'UTC », « le mari de Marie ») ;
- Directive : donne une directive (par exemple, "Trouve-moi un document.») ;
- Explicative : demande une explication (par exemple, "Pourquoi ?») ;
- Interrogative: nécessite une solution (par exemple, "Où Paul travaille-t-il?").

Les actes de langage sont employés pour classer des nœuds du graphe d'états du dialogue. Le graphe du dialogue représente une discussion entre l'utilisateur et le système où les nœuds sont les expressions de l'utilisateur et les arcs sont la classification donnée au nœud. Pour améliorer les échanges nous avons introduit de nouveaux actes de langage spécialisés :

- Go-back : l'utilisateur veut aller au nœud précédent, par exemple, quand il a commis une erreur ;
- Abort : l'utilisateur veut terminer le dialogue ;
- Confirm : le système a besoin d'une confirmation de l'utilisateur ;
- Propose : le système propose une valeur. Cet acte de dialogue peut être suivi d'un Confirm.

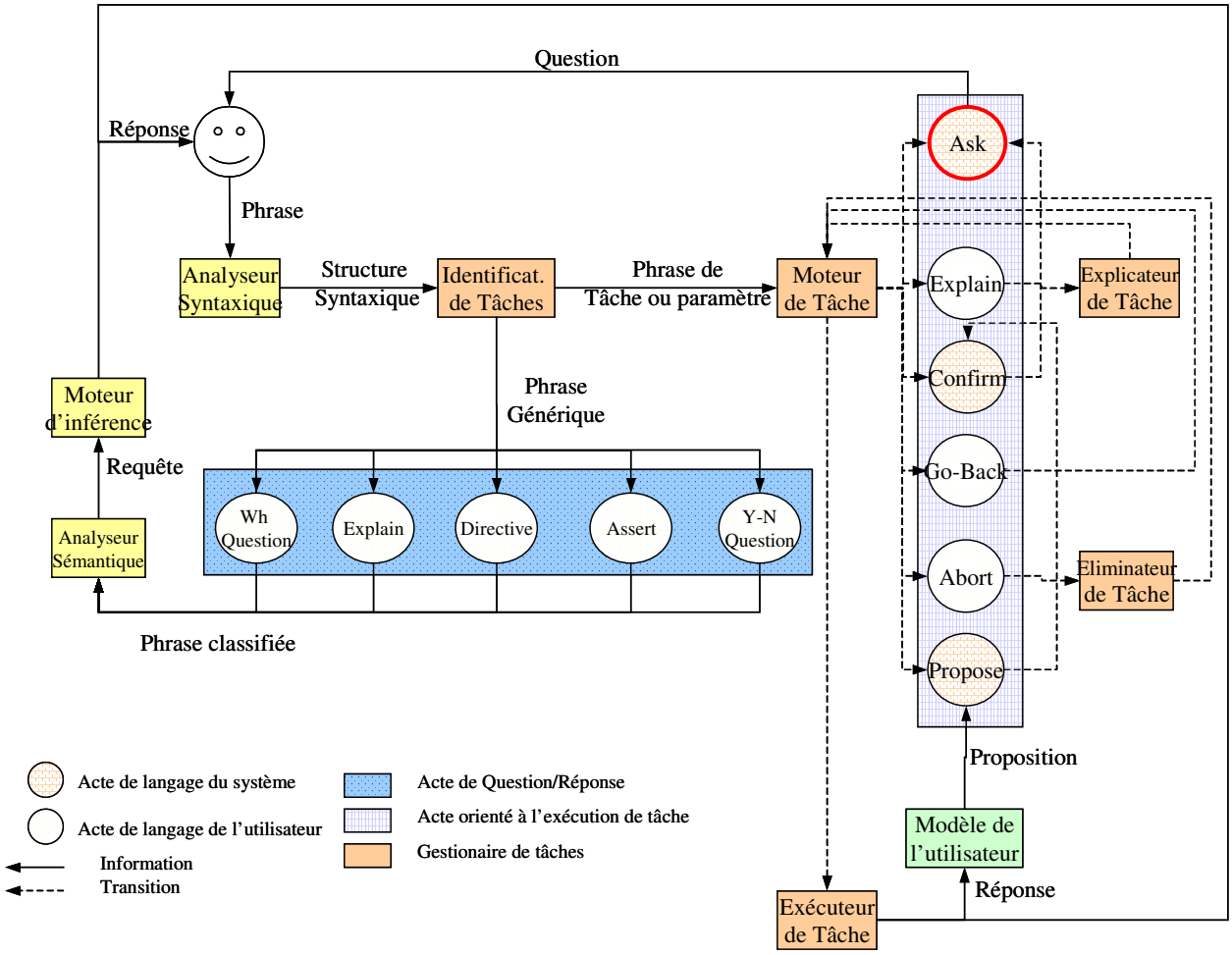

Figure 19 Le contrôle du dialogue.

La Figure 19 présente le mécanisme de coordination du dialogue. Elle illustre comment nous avons conçu l'interprétation de chaque acte de langage. L'interaction avec l'utilisateur démarre toujours avec un acte « Ask » de la part du système. La question standard est "What can I do for you?". Ensuite l'utilisateur peut demander des informations ou déclencher une tâche. À partir de la phrase tapée par l'utilisateur, l'identificateur de tâche la classifie comme « Phrase Générique » ou « Phrase de tâche ou paramètre ».

Une phrase *générique* est analysée par l'analyseur sémantique qui prend en compte le type d'acte de langage correspondant à la phrase. Quatre types d'actes de langage sont possibles : Assertive (acte assert), Explicative (acte explain), Directive (act directive) ou Interrogative (actes wh/question ou y-n/question). Finalement le moteur d'inférence utilise la base des connaissances pour trouver une réponse.

L'interprétation d'une phrase du type *tâche ou paramètre* est plus complexe. D'abord l'identificateur de tâches reconnaît la tâche concernée par la phase de l'utilisateur à travers

des termes trouvés dans les phrases nominales utilisant la représentation terminologique des tâches (template de tâches sur la Figure 15 et la Figure 20). Ensuite il fait le matching entre les modificateurs des phrases nominales et l'information sur les paramètres pour remplir les propriétés citées dans la phrase. Le moteur de tâche démarre donc un processus séquentiel d'interrogation de l'utilisateur pour saisir les informations manquantes pour l'exécution de la tâche. Pour chaque paramètre, un acte *Ask* est exécuté par le système. Lorsqu'une question est posée par le système l'utilisateur peut réaliser les actions suivantes :

- Répondre à la question (dans ce cas le moteur de tâche saisit la propriété et passe à la propriété suivante) ;
- Demander une explication (acte *Explanation*) ;
- Retourner à la dernière question (acte *Go-back*) ;
- Annuler le dialogue (acte *Abort*) ;
- Démarrer un nouveau dialogue.

Lorsque l'utilisateur demande une explication l'explicateur de tâche lui présente l'information codée dans la structure de tâche concernant le paramètre courrant (*params-explain* sur la Figure 20) et le moteur de tâche continue le dialogue à partir du paramètre courrant. L'acte *Go-back* simplement recule le dialogue vers le dernier paramètre, ensuite le dialogue continue normalement. Lorsque qu'un acte *Abort* est identifié, l'éliminateur de tâche met à jour quelques variables concernant le dialogue courrant et le système revient au dialogue sur la pile de dialogues ou à l'état initial du système avec la question standard. Si jamais l'utilisateur démarre un nouveau dialogue, le système mantient le dialogue précédent dans la pile de dialogues et rajoute à la pile le nouveau dialogue.

À chaque interaction le système peut demander une confirmation ou proposer des valeurs avec les actes *Confirm* et *Propose* respectivement. Pour confirmer une valeur, le système présente une question standard comme « Confirm the value? » et attend la réponse. Si une réponse positive est donnée, le système enregistre la valeur et le dialogue continue. Sinon, le moteur de tâche pose à nouveau une question concernant le paramètre courant. L'acte *Propose* est exécuté avant l'acte « Confirm ». Le gestionnaire du modèle de l'utilisateur cherche une valeur pour la proposer à l'utilisateur. Si une valeur est retrouvée elle est

présentée et le système interroge l'utilisateur sur la confirmation de la valeur utilisant un acte « Confirm ».

Finalement, lorsque plus aucune information n'est nécessaire, l'exécuteur de tâche déclenche la tâche et montre la solution à l'utilisateur. L'exécuteur envoie aussi des informations au gestionnaire du modèle de l'utilisateur qui enregistre les informations concernant le profil de l'utilisateur.

3.3.5. Le modèle de tâches

Le résultat d'un dialogue orienté tâche sera l'exécution d'une tâche particulière. Nous avons donc besoin du modèle de la tâche. Les tâches dans le travail de collaboration peuvent être complexes et prendre beaucoup de temps (des jours, des semaines, des mois voire des années). Dans ce cas, il est difficile de créer un modèle spécifique de dialogue pour modéliser plusieurs tâches. Les systèmes de dialogue génériques emploient les modèles explicites de tâches pour guider le dialogue avec l'utilisateur. De ce fait, de nouvelles tâches peuvent être ajoutées au système (AP) sans que l'on ait à changer la structure de dialogue.

Template de la tâche : « Recherche de document »

```
:verbs-synonyms (locate find search give look)
:nouns-synonyms (document documents paper file article)
author (author researcher of)
year (year date when)
title (subject title theme)
subject (about)
```

Structure de la tâche : « Recherche de document »

```
:name 'find-document
:params '(author year title subject)
:params-values '(nil nil nil nil)
:semantic-value '(name nil nil nil)
:params-confirm '(nil nil nil nil)
:params-labels '("Who is the author of the document?"
                 "When the document was written?"
                 "What is the title of the Document?"
                 "What is the subject?")
:params-save '(t t t t)
:params-explain '("The person who wrote the document."
                  "The parameter FROM represents who goes
                   to receive a Carbon Copy of the message."
                  "All the messages have a subject for identification."
                  "This is the body of the message, the text that will
                   be sent.")
:global-confirm "Confirm search?")
```

Figure 20 La définition d'une tâche avant son exécution.

Kölzer (1999) discute un générateur générique de système de dialogue. Le système présente un éditeur de tâches pour créer la structure de dialogue. Le développement d'une application est décomposé en plusieurs étapes :

- définition de tâches ;
- définition de la structure des tâches
- les paramètres d'application ;
- les messages de sollicitation de tâche.

Étant donné que nous ne sommes pas directement intéressés par un générateur d'applications mais par un moteur de dialogue générique, les tâches sont codées sous forme de structures LISP directement dans le modèle de tâches, ce qui simplifie la mise en place du système. Ainsi nous ne disposons pas d'un éditeur de hiérarchie ou d'un éditeur de tâches comme dans le travail de Kölzer et les quatre étapes sont réduites à un seul pas. Dans notre cas, une tâche est divisée en seulement deux parties : *template* et *structure*. La Figure 20 présente un modèle pour la tâche « Send Message ».

Pour identifier la tâche demandée par l'utilisateur et l'information sur les paramètres, l'analyseur sémantique utilise le *template* de la tâche. Le template contient des termes

linguistiques. La structure de la tâche contient toute l'information nécessaire pour exécuter la tâche. Les données nécessaires dans la définition de la structure de la tâche sont présentées dans le Tableau 1.

Paramètre	Définition
params	les paramètres de la tâche
params-values	les valeurs données aux paramètres
semantic-value	une fonction qui doit être exécutée sur une valeur donnée par l'utilisateur. Par exemple, la fonction « e-mail » doit donner la valeur carvalho@utc.fr pour le terme « carlos »
params-confirm	si la valeur est « true », une confirmation doit être fournie par l'utilisateur
params-labels	les questions à présenter à l'utilisateur
params-save	si les valeurs des paramètres doivent ou non être utilisées pour la formation du modèle de l'utilisateur (discuté dans la section suivante)
params-explains	si « true » (pour un paramètre) une explication doit être donnée par le système
global-confirm	si « true » une confirmation générale est demandée avant l'exécution de la tâche

Tableau 1 Définition des éléments de la structure de tâche.

3.3.6. Utilisation du profil de l'utilisateur

Nous avons vu dans la section 3.2 comment la modélisation de l'utilisateur peut être mise en place dans des systèmes de dialogue. Basiquement nous identifions deux approches : *modélisation a priori* et *modélisation dynamique*.

Dans la *modélisation a priori* des stéréotypes de bases de connaissances concernant les plans, les compétences, les buts et le comportement des utilisateurs sont mis en forme. La *modélisation dynamique* est accomplie au fil de l'eau et ne demande pas d'effort préalable de modélisation. Nous sommes intéressés plus particulièrement à ce deuxième type de modélisation de l'utilisateur.

Certains auteurs (comme Elzer et collaborateurs (1994)) soutiennent que le modèle de l'utilisateur peut être acquis à partir de l'interaction en utilisant des stratégies heuristiques codées au préalable, comme par exemple, la détection des concepts employés, la demande de feedback, etc. D'autres utilisent algorithmes d'apprentissage automatique (Okamoto et

collaborateurs, 2001). Notre approche utilise également un algorithme d'apprentissage simple. Cette technique permet la création de modèles qui ne sont pas dépendants du domaine d'application et tendent à réduire le besoin de feedback.

En effet, les dialogues pour l'exécution d'une tâche peuvent demander beaucoup d'effort de la part de l'utilisateur. De ce fait, le système doit prévoir les valeurs données par l'utilisateur afin de réduire le besoin de feedback. Par ailleurs, l'utilisation d'un profil permet de conserver et de réutiliser les résultats obtenus auparavant. Le profil peut aider à obtenir de meilleurs résultats en réduisant l'imprécision liée aux questions de l'utilisateur comparant celles-ci aux questions déjà posées.

Les modèles d'utilisateur (MU) sont fréquemment construits à partir de modèles statistiques (cf. par exemple (Flycht-Eriksson, 1999) (Webb et Pazzani, 2001) (Bueno et David, 2001) (Kiss et Quinqueton, 2001), (Okamoto et collaborateurs, 2001) ou (Zukerman et Albrecht, 1999)). Dans la plupart des applications, la modélisation de l'utilisateur est traitée comme un problème de classification automatique. En revanche, l'optimisation d'un dialogue peut utiliser encore d'autres techniques comme le groupement, l'apprentissage par exemple, le raisonnement basé sur des cas, un réseau neuronal non supervisé (BSB, ART) ou encore l'apprentissage à partir de réseaux de concepts.

Dans notre approche nous employons un procédé de génération dynamique d'un modèle d'utilisateur (MU) fondé sur des calculs statistiques. Toutes les tâches et inférences sont enregistrées dans le MU. Des valeurs sont prévues avec une technique basée sur la fréquence. Nous avons choisi d'implémenter cette technique simple car d'après certains auteurs, comme par exemple Zukerman et Litman (2001), les méthodes d'apprentissage actuelles ne sont pas suffisamment adaptées aux systèmes de dialogue et donc, de nouvelles approches doivent être expérimentées.

L'idée générale consiste à minimiser l'effort de l'utilisateur lors de l'exécution de tâches répétitives. Considérons l'exécution de la tâche « Envoi de Message ». L'utilisateur envoie plusieurs messages avec copie à « Jones ». Le système devrait être assez malin pour prévoir la valeur « Jones » pour une nouvelle exécution de la tâche « Envoi de Message » même si parmi les observations précédentes la valeur « Jones » n'était pas la plus fréquente. Ceci suggère que les observations récentes soient plus importantes que les plus anciennes. Pour

résoudre ce problème, nous calculons une fréquence pondérée en donnant un poids plus important aux valeurs des derniers enregistrements.

Le système doit minimiser les propositions incorrectes pour ne pas demander des efforts supplémentaires à l'utilisateur pour la correction d'erreurs. Ainsi, avant de faire une prévision l'agent assistant personnel décide si c'est une prévision admissible en utilisant la somme des fréquences pondérées. Le système calcule la fréquence pondérée pour toutes les valeurs d'une propriété particulière et propose celle pour laquelle la fréquence pondérée est plus importante que la somme de toutes les autres fréquences. Dans le chapitre suivant nous discutons de techniques plus élaborées de construction et d'adaptation dynamique du modèle de l'utilisateur.

3.3.7. Modèle du domaine

Pour permettre au système d'identifier les problèmes de l'utilisateur et de fournir des réponses aux questions particulières, il est nécessaire de conserver une base de connaissances dans l'agent assistant personnel. La connaissance de l'agent est utilisée pour identifier des objets et des relations demandées par l'utilisateur. Ces objets peuvent représenter des exemples d'objets de diverses classes (des personnes, des tâches, de conception, etc.) et posséder de nombreux synonymes. Par conséquent, il est tout à fait important d'utiliser des outils efficaces pour représenter une structure hiérarchique de concepts, d'objets, de synonymes, etc. Dans notre approche, nous utilisons le système de MOSS (Barthès, 1994) pour représenter la connaissance. Des concepts ont été organisés dans un réseau sémantique mettant en application une ontologie[7]. MOSS permet l'indexation d'objets par des synonymes. Plusieurs objets peuvent partager les mêmes index. La connaissance du domaine est d'abord utilisée dans l'analyse sémantique. Le moteur d'inférence trouve des relations demandées par l'analyseur pour créer la requête formelle. Puis, le moteur d'inférence de dialogue utilise la connaissance pour produire un résultat correspondant à la requête de l'utilisateur.

[7] Kassel et ses collègues ont discuté le développement d'une ontologie dans (Kassel et al., 2000).

1 User: Joe is a professor of Artificial Intelligence at UTC.
2 System: OK.
3 User: What does Joe do?
4 System: Joe is a professor of Artificial Intelligence at UTC.
5 User: What is Joe's profession?
6 System: Joe is a professor of Artificial Intelligence at UTC.
7 User: Where does Joe works?
8 System: At UTC
9 User: Is Joe a professor of Artificial Intelligence?
10 System: Yes
11 User: Is Joe a professor of artificial intelligence at UTC?
12 System: Yes
13 User: Is Joe a professor of artificial intelligence at PUC?
14 System: I don't know. None information about this.
15 User: What does Joe teaches?
16 System: Artificial Intelligence

Figure 21 Un dialogue intelligent.

La connaissance est importante parce qu'elle augmente la capacité du système à produire des réponses raisonnables. Considérons le dialogue reproduit sur la Figure 21. Au début le système n'a aucune information sur le métier de Joe. L'utilisateur entame le dialogue par un acte de dialogue « Assert » énonçant le métier de Joe. Puis, l'utilisateur pose plusieurs questions liées à l'expression initiale, le système peut quand même lui répondre. Pour réaliser le dialogue le système est capable de créer des objets et des liens sémantiques en fonction de l'expression d'entrée. Actuellement cependant, l'uniformité et les vérifications d'erreurs ne sont pas encore réalisées. Marcu et collaborateurs (2001) emploient un agent utilisant le langage naturel pour aider une personne qui n'est pas un expert en ingénierie cognitive à créer une base de connaissances. Toutes les fois qu'un problème est rencontré (des objets inconnus, des propriétés inconnues, des valeurs inconnues, des contradictions, etc.) l'utilisateur peut fournir des informations complémentaires pour le résoudre ou peut le conserver dans l'ordre du jour. Ici nous employons une approche similaire.

La Figure 22 illustre comment les concepts sont organisés. Chaque nœud représente un concept ayant des liens *is-a* ou *instance-of*. Chaque concept peut avoir un ensemble de points d'entrée (index) et de propriétés. Les points d'entrée peuvent être un synonyme ou un terme linguistique comme préposition ou verbe se rapportant au concept. Une propriété est un arc liant un concept d'origine (propriétaire de la propriété) à un concept de destination (valeur de la propriété). Dans la Figure 22 les propriétés sont les symboles ayant « : » à l'intérieur des concepts pour simplifier la Figure. Quand la première expression de la Figure 21 est

interprétée, l'objet « Prof58 » est créé. Il est alors employé pour répondre à des questions ultérieures. Les relations entre concepts permettent de répondre aux questions (la Figure 14, ligne 10). Ainsi, le dialogue mis en place permet à l'utilisateur d'explorer la connaissance représentée naturellement.

Le système peut identifier et interpréter correctement des questions très différentes liées aux même concepts (lignes 3 et 5) et répondre à différentes questions à leur sujet. Ceci est possible parce que la sémantique des propriétés est exploitée dans les questions, par exemple, le lien « :has-professor » est équivalent à « :has-profession », à « :has-occupation », etc. Ainsi, une propriété peut jouer un rôle qui est mis en évidence de différentes manières.

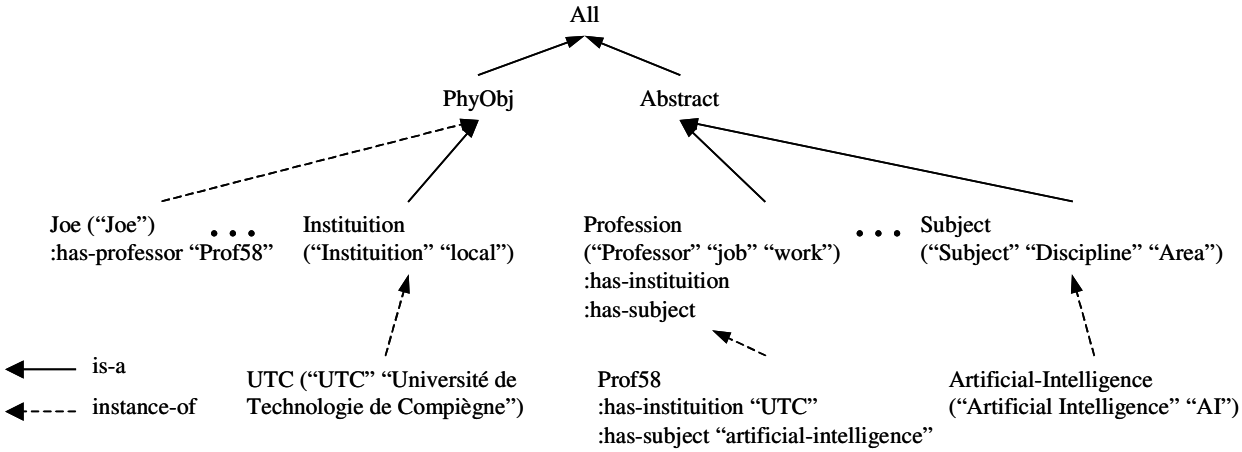

Figure 22 Base de connaissances.

3.3.8. Considérations sur les assistants de dialogue

Dans les sections précédentes nous avons abordé le problème de la communication entre l'utilisateur et son Assistant Personnel (AP) dans des situations où les utilisateurs doivent communiquer avec un AP pour effectuer un travail collaboratif. Nous avons insisté sur le fait que le langage naturel et l'IHM devraient être combinés pour fournir une meilleure interaction, ce qui nous paraît essentiel si nous envisageons effectivement de déployer des applications « intelligentes ». Un module de communication d'aide à l'utilisateur a été développé comme un système de dialogue comprenant un moteur de dialogue et utilisant plusieurs modèles : un modèle de dialogue, un modèle de tâche, un modèle d'utilisateur, et un modèle de domaine. Nous avons essayé de concevoir un système de dialogue qui puisse à la fois répondre à des questions et exécuter des services tout en utilisant des connaissances.

Pour exécuter des services et pour demander des connaissances, l'utilisateur dialogue avec son agent assistant personnel. Des tâches et des questions simples ont été utilisées pour démontrer l'efficacité du système et ses avantages par rapport aux interfaces des systèmes traditionnels. Le système a été développé en Common LISP.

3.4. Conclusions

Dans ce chapitre nous avons présenté des travaux concernant les assistants destinés à l'interaction avec l'utilisateur dotés de mécanismes complexes d'interaction et adaptation.

Nous avons étudié les relations existantes entre les interfaces adaptatives et hypermédias adaptatives, les systèmes de dialogue en langage naturel et les mécanismes d'adaptation connus.

Nous soutenons que les interfaces adaptatives et hypermédias adaptatives font partie d'un domaine de recherche plus général nommé modélisation de l'utilisateur qui sera étudié dans le chapitre suivant. Malheureusement, de nombreuses applications concernant les interfaces adaptatives et hypermédias adaptatives ne sont pas vraiment adaptatives car les connaissances du système sont statiques. C'est la raison pour laquelle nous avons étudié les mécanismes d'adaptation d'une autre forme d'interaction : *les systèmes de dialogue en langage naturel*.

Les systèmes de dialogue en langage naturel sont une alternative aux modes d'interaction classiques (fenêtres, boîtes de texte et boutons). Nous avons vu dans ce chapitre les éléments d'un système de dialogue devéloppé dans notre laboratoire. Nous avons pu constater qu'il est très difficile d'avoir un agent qui est adaptatif et qui dialogue. Dans ces applications les modèles principaux (tâches, domaine et dialogue) sont normalement figés et la seule possibilité d'adaptation du système correspond à l'interaction avec le maître. Par ailleurs, l'absence de mécanismes d'adaptation adaptés aux systèmes de dialogue est encore un problème à résoudre.

4 Adaptation et personnalisation des compétences

Dans le chapitre précédent nous avons vu comment des modèles plutôt statiques peuvent être mis en place dans des agents assistants capables d'avoir des interactions complexes avec l'utilisateur. Dans ces applications, l'aspect évolutif des modèles de l'agent est relégué au deuxième plan face à la complexité des modèles de connaissance et de communication.

Cependant, dans certaines applications, l'agent doit continuellement mettre à jour ses connaissances pour améliorer son efficacité à partir de connaissances *a priori* inexistantes ou limitées. L'assistant doit donc faire évoluer ses connaissances par lui-même pour s'adapter aux besoins de l'utilisateur de façon à fournir des services personnalisés. Pour cela, des méthodes d'apprentissage automatique doivent être mises en œuvre.

Dans ce chapitre nous étudions quelques méthodes d'apprentissage normalement mises en œuvre dans des agents assistants adaptatifs et nous présentons deux applications que nous avons développées. La première met l'accent sur comment apprendre le modèle de préférences des utilisateurs à partir de documents personnels et ensuite identifier des utilisateurs qui ont un intérêt commun sur des documents. La deuxième présente un SMA où des agents adaptatifs apprennent le modèle de préférences des utilisateurs à partir de leurs activités quotidiennes. Ces agents peuvent ensuite faire des recommandations pour aider un utilisateur à résoudre un problème donné.

4.1. Modélisation de l'utilisateur

La modélisation de l'utilisateur est un sous-domaine de l'Intelligence Artificielle. D'après Kobsa (2001), de nombreux systèmes ont été dévéloppés pour travailler différement en fonction du profil de l'utilisateur. Pour cela, ces systèmes utilisent une representation complexe de l'utilisateur composée par différentes stéreotypes, modèles d'inférence, niveaux de connaissances, croyances, préférences et buts. Ces systèmes demandent un effort considerable de modélisation *a priori* et sont généralement fortement connectés au domaine d'application.

Certains auteurs soutiennent que ces problèmes peuvent être minimisés lorsque quelques techniques de génération automatique de modèles sont mises en œuvre. Webb et Pazzani (2001) soutiennent que le problème de modélisation de l'utilisateur peut être traité comme une question classique d'un autre sous-domaine de l'IA : l'*apprentissage automatique*. La stratégie proposée consiste à stocker les actions de l'utilisateur dans une base de données et à utiliser ensuite un algorithme d'apprentissage pour fabriquer des modèles génériques. Nous suivons les idées de Webb et dans la suite de ce mémoire nous considérons la modélisation de l'utilisateur comme un problème d'apprentissage.

Cependant, d'après Webb, les limitations de l'apprentissage automatique posent encore beaucoup de problèmes dans le cas d'un agent assistant personnel, dont : *le besoin d'avoir grandes bases de données*, *la nécessité d'un classement préalable*, *la stabilité des données*, la *complexité de calcul*.

Généralement, les algorithmes d'apprentissage ont besoin de nombreux exemples pour acquérir des modèles fiables. *A contrario*, un assistant démarre son travail soit avec une base des données vide soit avec un modèle générique. Par conséquent, pour être rapidement mis en application l'agent doit utiliser une méthode d'apprentissage capable de gérer des nouveaux concepts avec le moins d'informations possible.

Pour pouvoir apprendre, les algorithmes on besoin d'un nombre d'observations déjà classées. Généralement, dans un agent assistant, le classement des informations est réalisé grâce au feedback de l'utilisateur. Par conséquent, si l'algorithme n'utilise pas une stratégie de classement automatique l'utilisateur sera surchargé.

Généralement, les modèles acquis par les algorithmes d'apprentissage sont dépendants de la structure des observations fournies en entrée. Du point de vue pratique, les variables qui représentent les cas doivent avoir des poids constants. Cependant, dans le cas d'un assistant, l'utilisateur peut, par exemple, changer souvent de centre d'intérêts, ce qui implique que l'importance des termes utilisés pour représenter le centre d'intérêts changera également. Par conséquent, une technique pour détecter ces changements et corriger le modèle acquis précédement par l'algorithme doit être mise en place.

Par ailleurs, les algorithmes d'apprentissage peuvent avoir besoin de beaucoup de temps pour apprendre les modèles à partir des données et pour classer des nouveaux cas. Par conséquent, les techniques d'apprentissage en ligne ne peuvent avoir qu'une complexité limitée.

Nous pouvons voir que les observations de Webb ont des points communs avec celles de Maes présentées dans la section 2.6.4 (Les agents assistants adaptatifs). Selon Zukerman et Albrecht (2001) et d'autres auteurs, normalement, ces modèles d'apprentissage sont utilisés dans deux stratégies différentes de modélisation de l'utilisateur:

- la modélisation basée sur le contenu ;
- la modélisation collaborative.

Dans la modélisation basée sur le contenu, les agents utilisent des techniques d'apprentissage automatique et de récupération d'information pour apprendre les modèles des items rencontrés par le passé (par exemple, des pages web, des e-mails, des news, des rendez-vous, etc.). Ensuite, ils utilisent de tels modèles pour trouver des items similaires qui sont alors proposés à l'utilisateur. Des exemples de telles applications sont le système Letizia (Lieberman, 1999) et WebWatcher (Armstrong et al, 1995) de Carnegie Mellon University. Ces deux applications sont utilisées pour proposer de nouvelles pages pendant la navigation sur le web.

Dans l'approche collaborative de modélisation de l'utilisateur les comparaisons ne sont pas faites au niveau des items mais au niveaux des modèles de groupes d'utilisateurs. Pour décider quelles informations proposer ou des actions à prendre, l'agent compare le profil de l'utilisateur avec le profil de groupes de personnes. Ainsi, il prendra les décisions qui sont normalement prises par un groupe d'utilisateurs partageant les mêmes centres d'intérêts. Dans les applications traditionnelles d'agents assistants collaboratifs on peut citer le filtrage de

« news » (Billsus et Pazzani, 1999), d'e-mail (Lashkari, 1994) et de documents (Magnini et Strapparara, 2001) (Enembreck et Barthès, 2003).

Selon Mladenic (1999) des approches hybrides (basées sur le contenu et sur la collaboration) sont parfois envisagées. Tout d'abord, des profils individuels sont créés à l'aide de techniques basées sur le contenu, puis, des techniques collaboratives sont utilisées pour trouver des profils similaires. Cette approche est utilisée par Cunningham dans (Cunnigham, 1994) et par Krulwich dans (Krulwich, 1997).

La modélisation de l'utilisateur (basée sur le contenu et collaborative) peut être utilisée pour la résolution de deux tâches différentes : la *modélisation du comportement* et la *modélisation d'intérêts*.

4.1.1. La modélisation de comportement

La modélisation du comportement consiste principalement à trouver des relations entre les actions de l'utilisateur. Des relations de précédence entre tâches et modèles de valeurs peuvent être trouvées avec des algorithmes d'apprentissage et des techniques de modélisation dynamique. Ces algorithmes fabriquent normalement un graphe d'états qui représentent les transitions entre les actions de l'utilisateur. Wu (2002) a proposé la modélisation dynamique des actions de l'utilisateur pour des activités de design avec des réseaux de Bayes. Les réseaux de Bayes ont été aussi utilisés par Ardissono et collaborateurs (2002) pour la personnalisation de configurations en-ligne de produits et services. Les poids initiaux des liens sont donnés par stéréotypes d'utilisateurs, ensuite les poids sont mis à jour en fonction des caractéristiques choisies par l'utilisateur.

Les modèles d'action sont souvent utilisés dans la fabrication d'interfaces « intelligentes » qui contrôlent des applications de domaines spécifiques. Hoyle (1997) et Lieberman (1998) ont utilisé ces techniques pour améliorer l'interaction entre l'utilisateur et le système d'exploitation Macintosh. L'assistant enregistre les actions déclenchées par l'utilisateur (ouverture des dossiers, ouverture des applications, etc.) et essaie de prévoir de nouveaux événements. Si le système arrive à faire une bonne prédiction, il diminue la charge de travail de l'utilisateur. Toutefois, ces applications ont en général une performance très faible car les utilisateurs ont tendance à travailler de façon aléatoire. Dans le domaine médical Jing et. al. (2002) ont développé une méthode basée sur un graphe pour modéliser les actions de l'utilisateur dans un système de diagnostic de maladies de la rétine. La performance maximale

du système, toutefois, ne dépasse pas 64,2% de bonnes suggestions. Une étude approfondie sur la personnalisation d'interfaces à partir de modèles de comportement a été faite par Weld et collaborateurs (2003).

4.1.2. La modélisation d'intérêts

La modélisation des centres d'intérêts est normalement utilisée pour raffiner et faciliter le travail de l'utilisateur dans certaines activités. Lieberman (1997) et Armstrong et al. (1995) ont montré comment un assistant peut rechercher des pages intéressantes sur le web. Tandis que l'utilisateur travaille, l'agent propose de nouveaux sites en regardant le contenu des pages visitées. Geutner et collaborateurs (1998) utilisent un assistant pour guider l'utilisateur qui travaille avec un système de navigation automobile. L'utilisateur peut demander des informations sur un restaurant au système à travers des commandes vocales et le système lui présente la route à suivre pour trouver le restaurant le plus proche tout en tenant en compte son profil (par exemple, le type de cuisine préférée). Lashkari et collaborateurs (1994) utilisent des assistants pour organiser les e-mails d'un groupe de personnes. Des profils individuels sont créés à partir de l'analyse du contenu des documents et des actions effectuées sur les documents (effacer, déplacer vers dossier « x », etc.). Quand un agent ne sait pas quoi faire, il demande à d'autres agents qui possèdent des profils similaires.

La modélisation d'intérêts est souvent accomplie à travers l'analyse des documents personnels de l'utilisateur. Dans un de nos articles (Enembreck et Barthès, 2002) nous détaillons comment l'agent peut apprendre le modèle de son maître à partir de documents. En effet, plusieurs techniques de classification de documents et de récupération d'information peuvent être utilisées. Le premier pas consiste à trouver des termes et les expressions importants dans la collection de documents personnels de l'utilisateur. Pour cela il existe une technique bien connue nommée TFIDF (Term Frequency - Inverse Document Frequency) proposée par Salton (1989) (voir section 4.3.3). L'idée générale de TFIDF est qu'un mot est important quand il est très fréquent dans un petit nombre de documents. Un nombre limité de termes est alors choisi pour représenter les documents sous forme d'un vecteur, où chaque valeur du vecteur est la valeur TFIDF d'un terme. Pour représenter l'utilisateur nous pouvons, par exemple, utiliser le centre de gravité de l'ensemble de documents. Dés lors, la représentation des intérêts de l'utilisateur est donnée sous forme d'un vecteur qui peut alors être comparé avec ceux d'autres utilisateurs ou d'autres ensembles de documents. La Figure 23 illustre un exemple de modèle d'intérêts extrait à partir d'une petite base de documents

dans le domaine de la recherche personnalisée sur le web. Normalement ce modèle contient une grande quantité de termes (par exemple, 1000). Dans les chapitres suivants nous discuterons plus précisément comment un agent peut apprendre le modèle d'intérêts d'un utilisateur.

```
(("USER" 287 6 6) ("QUERY" 151 4 4) ("AGENT" 116 3 3)
  ("PROFILE" 92 3 3) ("APARTMENT" 50 1 1) ("DOCUMENT" 88 3 3)
  ("RECOMMENDATION" 62 2 2) ("VECTOR" 37 1 1) ("REFINEMENT" 45 2 2)
  ("STATE" 43 2 2) ("GOOGLE" 30 1 1) ("DATA" 85 5 5)
  ("INFORMATION" 105 6 6) ("INFORMATIONAL" 28 1 1) ("YAHOO" 26 1 1)
  ("ASSOCIATION" 35 2 2) ("FEATURE" 33 2 2) ("NETWORK" 50 4 4)
  ("COVERAGE" 22 1 1) ("ACTION" 39 3 3) ("DECISION" 49 4 4)
  ("PERFORMANCE" 48 4 4) ("INTERVAL" 28 2 2) ("RELEVANCE" 35 3 3)
  ("BAYESIAN" 19 1 1) ("NEURAL" 19 1 1) ("PROBABILITY" 30 3 3)
  ("PERSONALIZATION" 23 2 2) ("COLLABORATIVE" 23 2 2) ("TASK" 47 5 5)
  ("MODEL" 37 4 4) ("ENGINE" 46 5 5) ("BROWSE" 29 3 3)
  ("TFIDF" 16 1 1) ("SIZE" 36 4 4) ("RELEVANT" 45 5 5) ...)
```

Figure 23 Un exemple de modèle d'intérêts.

4.2. Les assistants adaptatifs et le raisonnement utilisant la mémoire

Mladenic (1999) présente une étude sur 34 travaux concernant des agents intelligents destinés au traitement du texte et à la personnalisation de services disponibles sur web (personnalisation de journal électronique, recherche d'articles, recherche de personnes, navigation, et filtrage d'information). Tous ces agents utilisent de techniques similaires pour représenter les documents en utilisant un vecteur de termes (nous verrons dans les sections suivantes comment cela peut être accompli). Cependant, une énorme variété de méthodes d'apprentissage est utilisée pour classer ces documents. Le Tableau 2 présente les méthodes plus connues d'apprentissage utilisées par les travaux discutés par Mladenic et la fréquence avec laquelle ces méthodes apparaissent.

Méthode	Nombre d'utilisations
MBR et voisin plus proche	6
Naïve Bayes	6
TFIDF	5
Réseau de neurones	5
Règles de décision	4
Arbres de décision	2
Réseau de concepts	1
Réseau de Bayes	1
Théorie du raffinement	1
Programmation logique inductive	1

Tableau 2 Méthodes d'apprentissage citées par Mladenic (1999).

Dans une autre étude, Middleton a présenté plus de 80 travaux concernant des agents assistants dans les plus variés domaines (il compris le web). Nous avons analysé les travaux qui utilisent des méthodes d'apprentissage et nous présentons ces méthodes dans le Tableau 3.

Méthode	Nombre d'utilisations
MBR et voisin plus proche	15
TFIDF	6
Arbres de décision	6
Naïve Bayes	5
Clustering	4
Réseau de neurones	3
Programmation par l'utilisateur	3
Apprentissage par renforcement	3
Règles de décision	2
Algorithme génétique	2
Réseau de concepts	1
Réseau de Bayes	1
Règles d'association	1
CBR	1
Planning	1

Tableau 3 Méthodes d'apprentissage citées par Middleton (2001).

Les assistants ont besoin d'algorithmes incrémentaux d'apprentissage pour modéliser des informations.

Cela semble évident si l'on analyse le nombre d'agents assistants adaptatifs mettant en œuvre des algorithmes d'apprentissage basés sur la mémoire, comme nous pouvons constater dans les deux tableaux (Tableau 2 et Tableau 3). Par conséquent, nous avons décidé de consacrer

notre attention à cette technique. Les lecteurs intéressés par d'autres méthodes d'apprentissage peuvent consulter l'ouvrage de Mitchell (1997)[8].

Les assistants essaient toujours d'analyser le comportement de l'utilisateur à partir d'observations sur le comportement passé. Cette technique est connue comme raisonnement utilisant la mémoire (MBR Memory-Based Reasoning ou IBL Instance-Based Learning). En fonction de ces observations l'assistant peut apprendre les préférences du maître et par conséquent, personnaliser la présentation d'information, le calcul des résultats et agir de façon autonome.

Les premières approches faisant usage des techniques de MBR (voir la section 2.6.5) dans les agents assistants ont été proposées dans le début des années 90 par Kozierok et Maes (1993) pour l'organisation de réunions. Dans cette application les auteurs utilisaient une mesure de similarité pondérée pour trouver des cas de réunions précédentes dont les caractéristiques seraient identiques à celle de la réunion que l'utilisateur avait besoin d'organiser (par exemple, les ressources, les participants, le créneau horaire, etc.). Selon Maes, les mesures pondérées de similarité sont en général très coûteuses. Elles demandent beaucoup de calculs pour mettre à jour les poids qui correspondent au modèle courant des données. Pour résoudre le problème, les chercheurs ont décidé de sacrifier la caractéristique incrémentale de l'algorithme en ne mettant pas à jour le modèle en permanence (à chaque fois qu'une nouvelle activité est déclenchée) mais à des intervalles périodiques moins fréquents (par exemple, une fois par jour pendant la nuit). Cette technique est évidemment peu pratique, car le modèle des données est toujours en décalage avec le comportement réel de l'utilisateur et la méthode devient non-incrémentalle. Par conséquent, elle est peu utilisable dans des applications où l'utilisateur déclenche la tâche plusieurs fois par jour, par exemple, pour des recherches sur le web ou pour la gestion des messages du courrier électronique.

Le problème introduit par l'utilisation de MBR dans le développement des agents assistants constitue un point d'intérêt majeur de notre approche. En effet, nous cherchons une méthode d'apprentissage qui permette à l'agent d'apprendre sans nécessiter beaucoup de calculs. L'agent doit donc, toujours posséder une représentation correcte des expériences passées. Par ailleurs, cette méthode doit être suffisamment générique de façon que n'importe quel agent assistant puisse l'utiliser dans n'importe quel domaine. En fonction de ces besoins, nous

[8] D'autres ressources peuvent être retrouvés dans le *Machine Learning Journal* et *Data Mining and Knowledge Discovery Journal* (l'un et l'autre publiés par Kluwer).

avons dévéloppé une méthode qui sera présentée dans le chapitre V. Pour l'instant, nous discutons dans les sections suivantes comment l'assistant peut évoluer en interaction avec l'utilisateur en utilisant quelques techniques de MBR.

4.3. Les assistants personnels, la classification et le filtrage de documents

Dans cette section nous présentons une approche agent pour la classification personnalisée de documents. La classification de documents dans des environnements collaboratifs peut être utilisée pour le filtrage d'informations.

Le filtrage d'informations, nécessite en général des outils utilisant d'énormes bases de documents pour modéliser les préférences des utilisateurs. Nous soutenons que le filtrage collaboratif peut être accompli avec des systèmes décentralisés où les préférences sont acquises à partir de petites bases de documents individuelles. Une telle approche distribuée peut être plus souple dans la gestion des utilisateurs (addition et suppression) et distribue naturellement le calcul, conduisant à un meilleur usage des ressources. Par ailleurs, elle facilite l'évolution individuelle des préférences de chaque utilisateur.

Les systèmes multi-agents (SMAs) peuvent être mis en œuvre dans le développement des systèmes distribués génériques. Dans cette section nous utilisons un SMA dans la conception d'une application capable de personnaliser l'information dans un environnement collaboratif. Un assistant personnel est utilisé pour communiquer avec l'utilisateur et acquérir ses modèles de préférence. Ensuite un agent décision met en œuvre ces modèles pour décider à qui les nouveaux documents doivent être envoyés. L'application présentée dans cette section est générique et peut notamment diminuer le trafic des documents qui sont échangés entre les intranets et l'Internet.

Les agents sont capables de rechercher et analyser l'information de façon intelligente (Lieberman, 1999). Dans cette section nous utilisons quelques techniques de récupération et recherche d'information pour classer des documents selon les utilisateurs du système. En effet, le problème consiste à créer des modèles de préférences des utilisateurs et de les utiliser pour déterminer les utilisateurs qui sont intéressés par les nouveaux documents qui arrivent dans l'environnement. Dans notre cas, chaque utilisateur représente une classe et les techniques de classification nous permettant de trouver les classes les plus proches en fonction du contenu du document.

Le problème décrit dans le paragraphe précédent concerne notamment les serveurs d'information (Joachims, 1997) (Pannu et Sycara, 1996) et le filtrage d'e-mails (Rennie, 2000) (Lashkari et al, 1994), où le but consiste à réduire le nombre de messages et documents qui circulent dans le réseau. Avec ces techniques, seuls les messages personnalisés sont transmis. Dans les sections suivantes nous utiliserons souvent une base de données disponible à l'Université de la Californie Irvine (Blake et Merz, 1998) pour illustrer notre technique.

4.3.1. La classification et le filtrage de documents

Dans cette section chaque modèle d'utilisateur est traité comme une classe de documents. La plupart des applications collaboratives ne sont pas capables d'apprendre les modèles des centres d'intérêts des utilisateurs dans des environnements complexes comprenant des centaines d'usagés qui partagent documents et centres d'intérêts. En effet, dans les approches traditionnelles chaque utilisateur constitue une classe de documents et la classification devient très difficile quand le système doit apprendre avec de nombreuses classes. Dans ces environnements complexes, le système doit être le plus autonome possible et l'utilisateur ne doit pas avoir à faire du travail supplémentaire comme par exemple, fournir de nombreaux documents de référence (exemples positifs) et surtout des documents non représentatifs (exemples négatifs). Par conséquent, nous avons besoin d'une méthode de classification de documents qui travaille avec un grand nombre de classes contenant peu d'information.

La classification automatique de documents constitue aujourd'hui un sujet de recherche très actif. Cette activité fait partie normalement des études menées par les chercheurs intéressés dans l'apprentissage à partir de textes. L'idée est de concevoir un modèle représentatif d'une collection de documents pour organiser, récupérer et proposer des documents qui ont un contenu similaire. Normalement l'apprentissage à partir de documents est séparé en deux étapes : l'*apprentissage des modèles* et la *classification des documents*.

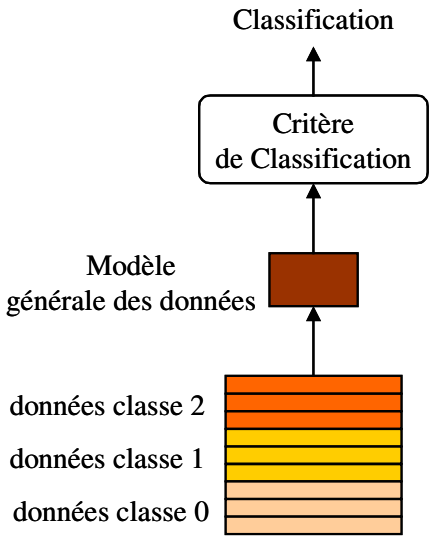

Figure 24 Approche traditionnelle de classification de document.

Dans la phase d'apprentissage, certaines caractéristiques sont choisies pour représenter les documents et un algorithme d'apprentissage est mis en route pour l'apprentissage des modèles des classes. Les documents sont normalement stockés dans une base de documents unique (Figure 24). Cependant, les algorithmes d'apprentissage ne sont pas capables de fabriquer des bons classificateurs pour beaucoup de classes pour deux raisons principales : (i) à mesure que le nombre de classes augmente, l'apprentissage devient très difficile. Dans ce cas, beaucoup de documents sont nécessaires pour distinguer les classes, ce qui représente une contrainte dans un environnement collaboratif ; (ii) la taille de l'espace est directement proportionnelle au nombre de classes. Pour bien séparer les classes, les algorithmes d'apprentissage automatique ont besoin d'un bon ensemble de caractéristiques. Dans le problème de classification automatique de documents chaque classe est normalement représentée par un grand vecteur de caractéristiques, par exemple, de taille 1000. Par conséquent, la dimension de l'espace va demander un effort considérable de calcul en fonction de l'algorithme utilisé. Goller et collaborateurs (2000) décrivent un processus de sélection de caractéristiques pour réduire la complexité des données et permettre ainsi l'usage d'algorithmes d'apprentissage automatique. Toutefois, cette approche demande beaucoup de temps de calcul.

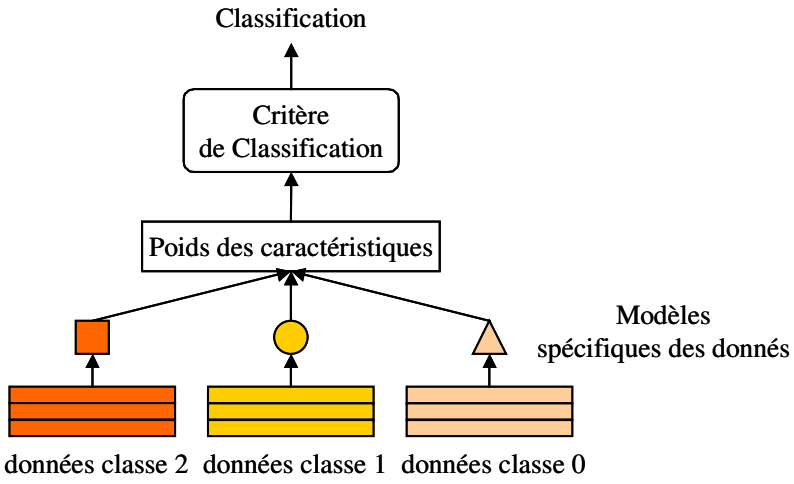

Figure 25 La technique du centre de gravité pondéré.

Le modèle de chaque classe peut être acquis individuellement, comme schématisé sur la Figure 25. Dans notre approche, chaque classe est construite comme une petite base de documents individuelle. Ensuite, l'agent assistant choisit un nombre de termes pour représenter chaque document. La représentation de chaque classe est donnée par le vecteur du centre de gravité des documents. Dans un deuxième temps, l'agent décision collecte les modèles des utilisateurs et fabrique un modèle général à partir de la mesure d'importance de chaque terme existant dans les différents modèles d'utilisateurs. Finalement, un agent décision peut mesurer l'importance d'un document pour chaque utilisateur et décider de les envoyer aux utilisateurs les mieux classés. La section suivante donne plus des détails sur la technique.

4.3.2. Les agents et le filtrage collaboratif

Dans un article (Enembreck et Barthès, 2002a) nous avons présenté une technique générale de classification de documents basée sur le centre de gravité des classes. Dans cette section nous utilisons la même technique avec des agents. Cela nous permet d'avoir un système décentralisé pour assurer un filtrage collaboratif. Dans notre approche, chaque utilisateur a un assistant personnel (AP). L'agent assistant personnel inspecte la base de documents personnels de l'utilisateur et prend en compte les documents marqués comme « intéressants » pour créer le modèle de préférences. Ainsi, seuls des exemples « positifs » sont considérés pour la construction des modèles. Le modèle de l'utilisateur est composé du vecteur

représentant le centre de gravité des documents. Chaque élément du vecteur correspond à la valeur TF-IDF des *N* meilleurs termes[9] trouvés dans la collection de documents.

Les modèles acquis par les APs sont transmis à un agent décision (AD). À partir des vecteurs centraux de chaque utilisateur l'AD calcule la qualité de chaque terme en fonction de sa capacité de discrimination. La mesure de discrimination de chaque terme prend en compte l'importance du terme pour chaque utilisateur et calcule un degré général d'importance du terme vis-à-vis l'ensemble des modèles. Cette mesure est calculée en utilisant l'index gini (Shankar et Karypis, 2000) (section 4.3.5). Pour classer un document, les modèles sont ordonnés selon une mesure pondérée de similarité qui prend en compte la qualité du terme, la valeur du terme dans le modèle et la valeur TF-IDF du terme selon le document (section 4.3.3). La Figure 26 montre l'architecture générale du système. Dans la Figure 26 l'AD décide d'envoyer le nouveau document seulement à l'agent B.

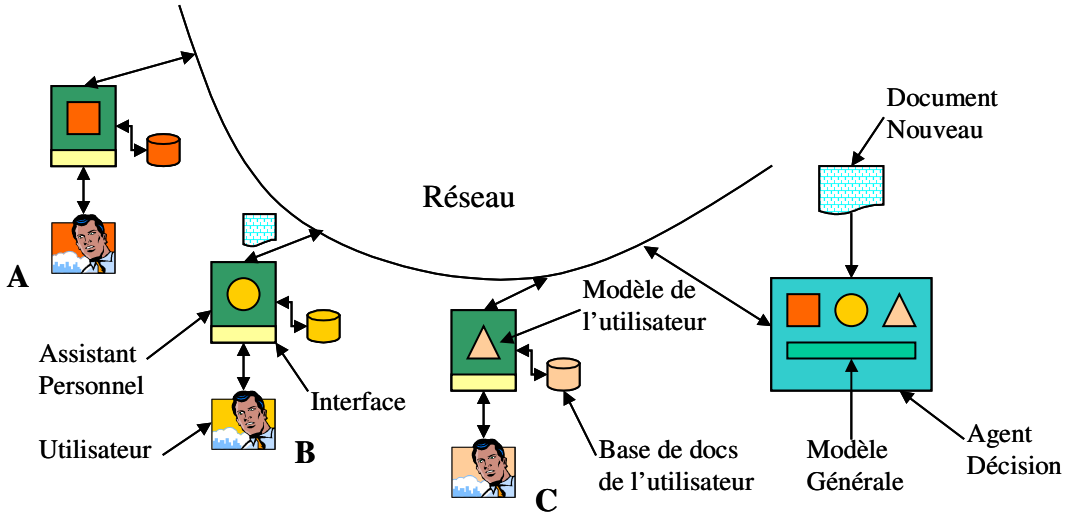

Figure 26 Architecture du système multi-agent.

Avec la méthode présentée dans cette section nous pouvons gérer un nombre illimité de classes (utilisateurs), car les modèles sont acquis séparément. Par ailleurs, la création de chaque classe n'exige pas beaucoup de documents.

4.3.3. L'apprentissage du profil de l'utilisateur

Dans ce travail les APs doivent fournir une interface pour le stockage des documents de l'utilisateur. L'utilisateur indique quels sont ses documents favoris et quand l'assistant doit

[9] Nous éliminons les verbes et l'antidictionaire avec un dictionaire de noms. Nous n'utilisons pas d'algorithme de lemmatisation.

mettre à jour sont modèle. La première étape dans la création du modèle de l'utilisateur consiste à trouver les bonnes caractéristiques à partir des documents. Pour cela nous utilisons la mesure TF-IDF (Salton, 1989).

TF-IDF considère un terme important s'il est très fréquent dans un petit nombre de documents. TF_{ter} est la fréquence du terme *ter* dans la collection de documents et IDF est la fréquence inverse par rapport le nombre de documents calculé avec $\log(N/DF_{ter})$ où N est le nombre total de documents dans la collection et DF_{ter} est le nombre de documents qui contiennent le terme *ter*. Ainsi, la valeur TF-IDF d'un terme quelconque *i* par rapport à une collection de documents est donnée par :

$$TFIDF(i) = TF_i \times \log\left(N / DF_i\right) \tag{1}$$

La représentation d'un certain document *d* prend en compte les fréquences des termes en *d*, et est donnée par le vecteur

$$\mathbf{d} = \left\{TF_1^d \times \log\left(N / DF_1\right), \ldots, TF_n^d \times \log\left(N / DF_n\right)\right\}$$

où TF_i^d est la fréquence du terme *i* dans le document *d*.

Nous avons limité empiriquement la dimension du vecteur de termes à 1000. La mesure TF-IDF pour la sélection de termes a été choisie, car celle-ci est souvent utilisée dans d'autres travaux. Par ailleurs, elle est très simple de calculer et sa sémantique semble robuste. Néanmoins d'autres méthodes peuvent être utilisées, comme celle présentée par Mather et Note (2000). Dans ce travail les auteurs ont choisi seulement les termes contenus dans les titres et les sous-titres des documents. Dans ce cas, les documents doivent forcément être bien structurés, ce qui peut s'avérer contraignant. TF-IDF en revanche est une approche plus générale qui prend en compte tout le texte du document et qui est moins sensible au bon choix des noms de sections des documents.

L'apprentissage du modèle des classes peut être accompli à partir d'une grande variété d'algorithmes d'apprentissage automatique, comme par exemple, les arbres de décision, le naïve Bayes, des réseaux de neurones, le KNN, etc. Dans ce travail nous utilisons une approche fondée sur le centre de gravité de la classe. En effet, le modèle de la classe est donné

par le vecteur moyen. Ainsi, à partir d'une collection de documents S qui appartiennent à la classe *c*, la formule suivante donne le vecteur central :

$$\mathbf{c} = \frac{1}{|S|} \times \sum_{d \in S} \mathbf{d} \tag{2}$$

Le principal avantage du centre de gravité sur d'autres méthodes est l'usage seulement des exemples positifs dans la construction du modèle. Par conséquent, l'utilisateur n'a pas besoin de fournir de contre-exemples. Par ailleurs, les calculs sont très simples.

4.3.4. Le centre de gravité pondéré et l'agent décision

Dans ce travail, l'agent décision (AD) est responsable du processus de classification des nouveaux documents. L'AD combine les nombreux modèles des utilisateurs pour identifier ceux qui ont des préférences semblables et sont intéressés par les documents. Dans cette technique, la classification se réduit à la comparaison entre vecteurs.

Des nombreuses mesures ont été proposées pour le calcul de la proximité entre deux vecteurs. Deux des techniques les plus connues sont la mesure du cosinus et la distance Euclidienne. Cependant, dans notre cas, nous ne pouvons pas mettre en œuvre ces techniques directement parce que les vecteurs centres de gravité sont composés de termes différents. Il ne faut pas oublier que le processus d'apprentissage mené dans la section précédente prend en compte chaque classe séparément. Par conséquent, les vecteurs contiennent des caractéristiques différentes.

L'AD reçoit les modèles des utilisateurs représentés sous forme des vecteurs centres de gravité. Ces vecteurs possèdent le même nombre de caractéristiques. Normalement, ces vecteurs peuvent partager certaines caractéristiques, notamment si les utilisateurs ont des centres d'intérêts en commun. Par exemple, supposons qu'un utilisateur s'intéresse à « religion » et un autre à « athéisme ». Vraisemblablement, ces utilisateurs partagent un certain « dictionnaire » commun. Par conséquent, nous pouvons dire que le processus d'apprentissage de la section précédente sert à découvrir comment chaque classe est composée mais il ne sert pas à découvrir comment discriminer l'une de l'autre.

Par ailleurs, un terme peut être très important pour représenter une certaine classe mais il peut être également très mauvais pour distinguer cette classe d'une autre. Si nous reprenons la base de documents des 20 groupes de discussion de l'UCI (Blake et Merz, 1998), par exemple,

nous avons découvert que le terme « article » a été considéré important pour la plupart des classes et donc il a une forte chance de ne pas contribuer au processus de classification.

Afin de combiner les différents modèles des classes, nous fabriquons un modèle générique qui prend en compte le pouvoir de discrimination de chaque terme (section 4.3.5). Le pouvoir de discrimination de chaque terme représente le modèle général de la Figure 26. Les poids de tous les termes sont mis en œuvre dans le processus de comparaison entre chaque classe et le nouveau document. Comme résultat, ce processus donne une liste de similarités normalisées. Ainsi les classes voisines peuvent être identifiées directement (section 4.3.6).

4.3.5. Le pouvoir de discrimination des caractéristiques

La mesure de la qualité de discrimination de chaque terme est faite avec la méthode dite « gini index ». Le seul critère mis en œuvre dans le choix de cette mesure a été la simplicité de calcul. Dans les paragraphes suivants nous détaillons la méthode.

Considérons que l'ensemble $\{C_1, C_2,..., C_m\}$ est composé par des centres de gravité calculés selon les formules précédentes et T_w le vecteur donné par les valeurs d'importance de w dans chacun des centres de gravité, soit $T_w = \{C_{1w}, C_{2w},\ldots, C_{mw}\}$. T'_w le vecteur normalisé, soit $T'_w = \{C_{1w} / \| T_w \|_1, C_{2w} / \| T_w \|_1, \ldots, C_{mw} / \| T_w \|_1 \}$. Le pouvoir de discrimination de w est calculé selon la formule suivante :

$$P_w = \sum_{j=1}^{m} T_{jw}^2 \tag{3}$$

P_w est égal au carré de la taille du vecteur T'_w. Par conséquent, P_w est toujours dans l'intervalle $[1/m, 1]$. P_w a la plus petite valeur quand $T'_{w1} = T'_{w2} = \ldots = T'_{wm}$, tandis que sa valeur augmente au fur et à mesure que de classes contiennent une valeur non nulle.

Exemple 1: Considérons l'ensemble de centres de gravité $C = \{C_1, C_2, C_3, C_4, C_5\}$ et deux termes *a*, *b* trouvés dans C. À partir de C nous récupérons les valeurs de *a* et *b* dans les vecteurs, soit $T_a = \{0.12, 0.56, 0.45, 0.22, 0.73\}$ et $T_b = \{0.0, 0.85, 0.0, 0.08, 0.1\}$, respectivement. Après normalisation $T'_a = \{0.06, 0.27, 0.22, 0.11, 0.35\}$ et $T'_b = \{0.0, 0.82, 0.0, 0.08, 0.09\}$. la formule (3) donne comme résultat $P_a = 0.2474$ et $P_b = 0.6869$. Les valeurs P_a et P_b sont celles utilisées pour la classification. Avec cette technique nous pouvons identifier que *b* est meilleur pour distinguer les classes que *a*. Cette conclusion semble

évidente quand nous analysons les valeurs de T'_a et T'_b. Tandis que dans T'_a il est très difficile de distinguer une classe des autres, dans T'_b la classe 2 semble être nettement la meilleure.

Le poids des caractéristiques agit comme un paramètre de normalisation qui permet ensuite de comparer les nombreux vecteurs centre de gravité. Toute caractéristique présente dans l'ensemble *C* est évaluée. L'application de ces poids est discutée dans la section suivante.

```
function Classification (document, C){
   S = set of terms and frequency inside document
   W = set of weights for all terms in C
   q = { }
   foreach centroid c of C{
      d  = Frequency vector for all terms of c in S
      w = weights in W for all terms in c
      q = insert-priority-queue (q, class of c, score (c,d,w))
      }
    return most important class from q
 }
```

Figure 27 Algorithme de Classification.

4.3.6. La classification des documents

La classification d'un document démarre avec l'extraction d'un certain nombre de termes et le calcul de leurs fréquences respectives (TF). Pour chaque classe *c*, nous construisons un vecteur *d* pour représenter le document. Chaque élément de *d* correspond à la fréquence *TF* d'un terme w_i multipliée par la valeur *IDF* de la classe c. Ensuite nous mesurons la similarité de *c* et *d* avec la formule suivante :

$$score(c,d,w)=\frac{\sum_{i=1}^{|d|} c_i \times d_i \times w_i}{|d|} \qquad (4)$$

Dans l'équation (4) *c* est le centre de gravité, *d* est le document et *w* est le vecteur des poids des termes. Pourtant, dans cette formule nous calculons la qualité moyenne des termes du document, en prenant en compte les valeurs de TF, IDF et le pouvoir de discrimination. Nous avons encore amélioré la mesure précédente en ajoutant un seuil minimal de discrimination. Ainsi nous ne considérons que les termes dont le pouvoir de discrimination excède 0,3. La Figure 27 présente l'algorithme général de classification qui met en œuvre notre critère.

Exemple 2: Dans cet exemple nous reprenons une petite partie de la base de documents UCI (Blake et Merz, 1998) choisie de façon aléatoire pour illustrer notre technique. Considérons l'ensemble de centres de gravité C = {$C_{atheism}$, C_{space}, $C_{religion}$} qui représente les classes *atheism*, *space* et *religion* respectivement. L'ensemble de termes existants dans *C* est groupé dans F, soit F = {atheism, religion, belief, space, NASA, universe, science}. *doc* est un vecteur avec les fréquences pour chacun des Fi. La taille des classes est limitée à 4 caractéristiques comme montre le Tableau 4. Le Tableau 4 montre la valeur TF-IDF moyenne pour les classes, soit, les centres de gravité. Les positions marquées par « X » représentent l'absence du terme dans la classe. Nous avons choisi seulement 7 caractéristiques pour simplifier l'exemple.

Class\Feature	Atheism	Religion	Belief	Space	Nasa	Universe	Science
Atheism	0.318	0.337	0.223	X	X	0.233	X
Space	X	X	X	0.523	0.35	0.383	0.166
Religion	X	0.493	0.322	X	X	0.191	0.079

Tableau 4 Les centres de gravité des classes**.**

Le pouvoir de discrimination de chaque caractéristique est calculé avec la mesure gini index, vue dans la section 4.3.5. Le résultat de ce premier processus est illustré dans le Tableau 5. La représentation de doc est donné dans le Tableau 5. Le pas suivant consiste à calculer le score de chaque classe selon la formule 4. Le score des classes *atheism*, *space* et *religion* est donné dans le Tableau 6. Comme nous pouvons constater dans le Tableau 6, le score le plus important est celui de la classe *atheism*. La deuxième classe est *religion* et la troisième *space*.

Feature:	Atheism	Religion	Belief	Space	Nasa	Universe	Science
P	1.0	0.52	0.52	1.0	1.0	0.36	0.56

Tableau 5 Pouvoir de discrimination des caractéristiques.

Feature:	Atheism	Religion	Belief	Space	Nasa	Universe	Science
Doc (Freq.)	6	3	4	0	0	2	1
$doc_{atheism}$	6.04	2.85	3.13			2.32	
doc_{space}				0	0	2.89	0.89
$doc_{religion}$		2.62	5.12			2.54	1.43

Tableau 6 La représentation de *doc*.

Score(C_I,doc,P)	Value
I = Atheism	(0.318*1.0*6.04 + 0.337*0.52*2.85 + 0.223*0.52*3.13 + 0.233*0.36*2.32) / 4 = **2.97**
I = Space	(0.523*1.0*0.0 + 0.350*1.0*0.0 + 0.383*0.36*2.89 + 0.166*0.56*0.89) / 4 = **0.480**
I = Religion	(0.493*0.52*2.62 + 0.322*0.52*5.12 + 0.191*0.36*2.54 + 0.079*0.56*1.43) / 4 = **1.768**

Le score des classes.

4.3.7. Performance

Au début de ce travail nous cherchions une méthode de classification de documents capable de manipuler plusieurs classes d'une façon distribuée puisque qu'un environnement collaboratif peut compter avec nombreux utilisateurs qui représentent les classes du système. Par ailleurs, dans un tel environnement, la technique de classification de documents doit être suffisamment souple pour permettre l'insertion et suppression d'utilisateurs (classes) sans avoir besoin de mettre à jour tous les modèles du système, ce qui n'est pas simple.

Nous savions également qu'une telle application distribuée risque d'augmenter le déséquilibre entre le nombre de classes et la précision du système et pourtant avoir une précision inférieure à celle d'un système de classification centralisé.

Pour mesurer la performance du système nous avons comparé la performance à celle de l'algorithme Rocchio sur la base de documents des 20 groupes de discussion mis à disposition par l'UCI (Blake et Merz, 1998). La performance de l'algorithme Rocchio a été extraite de l'article de Joachims (1997). Il est important de signaler que dans un outil de filtrage collaboratif la performance du système n'est pas un paramètre essentiel. Par exemple, si un utilisateur reçoit un document peu intéressant, il va tout simplement l'ignorer. En revanche, si l'utilisateur ne reçoit pas un document important (par exemple, un appel à communication) les conséquences peuvent être beaucoup plus importantes. Nous discutons ces aspects dans la section 4.3.8. Dans les expérimentations menées dans ce travail nous avons utilisé le même protocole d'évaluation que Joachims (1997) afin de pouvoir comparer directement les résultats. Dans ce protocole nous faisons la moyenne entre plusieurs itérations en choisissant 67% des exemples pour l'apprentissage et 33% pour le test. Nous avons observé des écarts-type négligeables. La Figure 28 présente nos résultats.

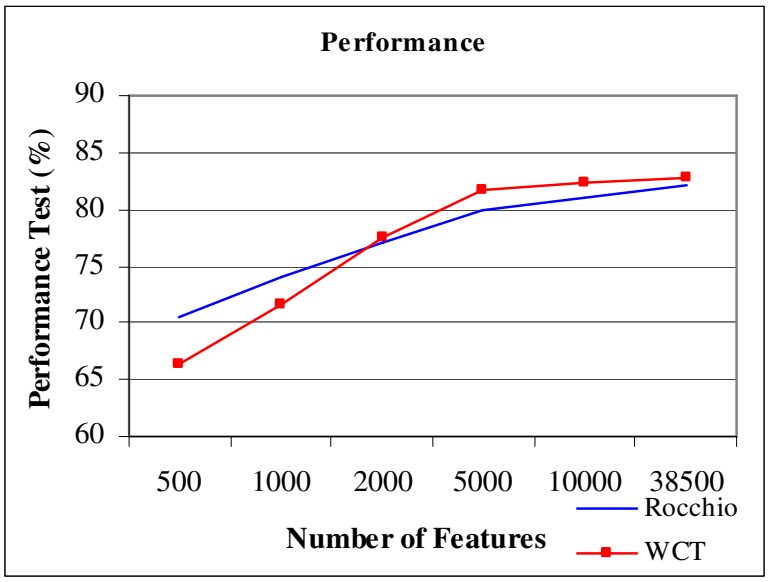

Figure 28 Comparaison avec l'algorithme Rocchio.

Dans la Figure 28 nous présentons les résultats obtenus à partir de la variation du nombre des caractéristiques. Comme nous nous y attendions, la performance du système est plutôt faible avec un petit nombre de caractéristiques. Dans ce cas, puisque les caractéristiques sont choisies de façon individuelle pour chaque classe, la plupart des caractéristiques risquent de ne pas avoir une bonne qualité de discrimination. En revanche, au fur et à mesure que le nombre de caractéristiques augmente, plus de caractéristiques discriminantes risquent d'apparaître.

Nous n'avions pas beaucoup d'espoir d'avoir de bons taux de classification car notre algorithme d'apprentissage n'utilise qu'une vision partielle de la base de documents pendant la création des modèles. Malgré cela, la performance de notre approche est comparable à celle de l'algorithme Rocchio. Puisque la performance dépend directement de la mesure de discrimination, nous pouvons conclure que la méthode gini index (section 4.3.5) joue très bien son rôle.

4.3.8. Considérations sur le filtrage collaboratif

Dans le filtrage collaboratif la performance n'est pas un aspect fondamental. Le but consiste à diminuer le nombre de messages (mails, documents, etc.) transmis dans un réseau et ne livrer que des messages intéressants aux destinataires. Ainsi, quand une erreur est commise, deux situations peuvent arriver : (i) l'utilisateur reçoit un document qui ne l'intéresse pas ou (ii) un utilisateur ne reçoit pas un document qui l'intéresse. Le deuxième cas est beaucoup plus grave

que le premier. Par conséquent, nous devons essayer de diminuer celui-ci. Une deuxième question à considérer concerne la nature multi-classe des documents, puisqu'un document peut intéresser plusieurs utilisateurs.

L'algorithme présenté dans la Figure 27 peut être étendu de façon à diminuer les erreurs de type (ii) et également permettre l'association d'un document à plusieurs utilisateurs. À partir d'une analyse des résultats, nous avons pu constater que les classes placées en tête de la queue de priorité contiennent une certaine relation sémantique. Par exemple, les classes "*talk.religion.misc*", "*alt.atheism*" et "*soc.religion.Christian*" sont normalement groupées parce qu'elles sont formées par un vocabulaire commun. Par conséquent, un seuil de similarité peut être mis en œuvre pour les distinguer des autres.

La Figure 29 présente une situation typique où l'algorithme de la Figure 27 se trompe. La classe du document est "*talk.religion.misc*" mais la classe en tête de la queue est "*soc.religion.christian.*" Ces classes sont normalement très proches et donc il est très difficile de les distinguer. Dans ce cas, nous pouvons empiriquement établir un seuil prenant en compte la valeur attribuée à la première classe, par exemple multipliant cette valeur par 0.8. Si l'on reprend l'exemple de la Figure 29, le seuil *T* est égal à 1.4656543 * 0.8 = 1.17252344. Ensuite nous choisissons seulement les classes dont la valeur est supérieure au seuil établi ("*soc.religion.christian*", "*talk.religion.misc*", "*alt.atheism*"). Avec cette technique simple un document peut donc être envoyé à plusieurs utilisateurs. Par conséquent nous augmentons la probabilité d'avoir des erreurs de type 1 mais nous aurons moins des erreurs de type 2.

```
(:CLASS "talk.religion.misc")
(:CLASSIFICATION
      ("soc.religion.christian" 1.4656543))
Priority Queue:
(("soc.religion.christian" 1.4656543)
 ("talk.religion.misc " 1.3538731)
 ("alt.atheism" 1.2848469)
 ("talk.politics.misc" 0.54359883)
 ("talk.politics.mideast" 0.51632446)
 ("comp.graphics" 0) …)
```

Figure 29 Queue de priorité.

4.3.9. Discussion et travaux concernés

Des applications de filtrage collaboratif (Delgado et al., 1998) (Sarwar et al., 2001) peuvent stocker les informations des utilisateurs sur des bases de données individuelles ou génériques. Ensuite, normalement des algorithmes d'apprentissage du type « voisin le plus proche » servent à classer des nouveaux exemples (documents, messages, etc.). Une base de données générique contient des informations sur tous les utilisateurs. Puisque chaque utilisateur peut avoir beaucoup d'exemples, ces bases de données peuvent représenter beaucoup trop d'information pour les algorithmes d'apprentissage qui devront consommer énormes quantités de ressources. La performance de ces algorithmes dépend du nombre d'utilisateurs et du nombre d'exemples pour chaque utilisateur. Elle tend à se dégrader au fur et à mesure que de nouvelles données sont ajoutées au système. Ce problème est connu comme le problème de *passage à l'échelle*.

Une alternative au problème consiste à utiliser des bases de donnés individuelles pour chaque utilisateur. Dans ce cas, les exemples sont classés comme intéressants ou non. Cela suppose que le système doit capturer le feedback de l'utilisateur pour assurer la qualité des données d'apprentissage. Dans (Delgado et al., 1998) par exemple, l'utilisateur raconte au système quels documents il trouve intéressants lors de l'analyse des résultats des recherches faites sur le web. Ces documents sont ensuite ajoutés à la base de données et classés comme « positifs ». Ensuite le système observe la gestion de sites favoris comme la suppression d'un site, la mauvaise classification d'un site ou encore les sites refusés et les classe comme exemples « négatifs ».

Dans certains cas, la nécessité d'un feedback peut provoquer une surcharge de travail pour l'utilisateur et donc une technique plus générale doit être mise en place. La technique

présentée dans ce travail n'exige pas d'exemples négatifs, ce qui peut soulager le travail de l'utilisateur. Par ailleurs elle présente une haute tolérance au problème de passage à l'échelle. Puisque dans la phase d'apprentissage seulement des bases de données individuelles sont considérées et chaque assistant apprend les modèles des utilisateurs en parallèle, le système possède une faible dépendance au nombre d'exemples. De plus, comme chaque utilisateur est représenté comme un seul vecteur de termes dans l'agent décision, nous assurons une croissance linéaire de complexité lors de la mise au jour du modèle générale (calcul des poids).

Des agents sont souvent mis en œuvre dans la récupération distribuée d'information (Clark et Lazarou, 1997). Dans ces systèmes le problème principal consiste à combiner les résultats des différentes sources d'information. Kretser et collaborateurs (1998) et Callan (2000) travaillent sur ces questions. Kretser conclut dans son article que des approches distribuées de récupération d'information sont plus rapides et plus efficaces, mais moins précises. Cependant, les résultats obtenus dans les expériences faites sur la base de données des 20 groupes de discussion de l'UCI montrent que la performance de notre approche peut être comparable à celle de l'algorithme Rocchio (Joachims, 1997).

4.4. Les assistants personnels et la capitalisation des connaissances

Notre vie quotidienne est de plus en plus sous influence des ordinateurs. À mesure que l'utilisation de l'ordinateur se banalise dans un nombre énorme d'activités quotidiennes nous sommes de plus en plus concernés par au moins trois gros problèmes : (i) l'environnement logiciel devient de plus en plus complexe ; (ii) le travail est souvent développé en coopération avec d'autres personnes et (iii) nous avons besoin de gérer notre connaissance sous forme d'expériences.

Puisque les environnements informatiques deviennent complexes, nous sommes souvent contraints d'utiliser plusieurs outils informatiques comme les gestionnaires de messages, des navigateurs web, les editeurs de documents, etc. La conséquence immédiate de cette complexité est une forte surcharge cognitive qui entraîne indirectement une désorganisation. Cette désorganisation s'avère dramatique principalement lorsque l'utilisateur participe à un processus collaboratif où l'information doit être partagée.

Vis-à-vis de ce problème, des environnements « intelligents » capables de communiquer de façon intuitive avec l'utilisateur et d'automatiser certaines tâches sont tout à fait envisageables. Pour cela, plusieurs techniques ont été proposées, parmi elles les collecticiels, les outils collaboratifs ou encore les collecticiels et outils collaboratifs gérés par des agents intelligents, notamment les agents assistants personnels.

Les collecticiels et les outils collaboratifs (Tacla, 2001) mettent en œuvre un environnement multi-application. Par conséquent, les utilisateurs peuvent travailler sur des espaces collaboratifs, organiser les documents, communiquer et automatiser certaines tâches simples. Cependant, nous pouvons rencontrer les problèmes suivants dans ces environnements :

- Certaines tâches du domaine ne peuvent être représentées facilement, par exemple, des tâches comme « fabriquer un moteur électrique » ne peuvent être capitalisées avec un niveau de détail élevé sans une forte personnalisation ;

- Certains outils (e-mail, moteur de recherche, agenda) ne sont généralement pas très intégrés. Étant donné que les utilisateurs travaillent toujours dans un contexte particulier, le contexte pourrait être utilisé pour prévoir des actions ultérieures ou raffiner les résultats passés ;

- Les préférences des utilisateurs ne sont généralement utilisées que pour la personnalisation des interfaces. En revanche, des profils extraits à partir des contenus pourraient fournir des informations intéressantes pour la résolution de certains problèmes ;

- Les expériences des utilisateurs sont en général perdues. Celles-ci devraient être partagées. Des informations supplémentaires doivent être capturées puisque les documents ne représentent pas l'expérience complète de l'utilisateur. En particulier, comment et pourquoi un document a été écrit peut être plus important que le document lui-même.

Une autre approche consiste à utiliser des agents assistants et les systèmes multi-agents (voir (de Azevedo, 1997), (Shen, 2001) et (Barthès et Tacla, 2002)) pour améliorer les fonctionnalités des collecticiels. A priori celle-ci n'a pas les mêmes limitations les outils collecticiels, car un agent assistant personnel peut être développé pour s'adapter à son maître, tout en fournissant d'une façon adéquate la sémantique nécessaire pour l'exécution des

services externes. Une communication intuitive peut être envisagée avec des interfaces en Langage Naturel développées et modifiées dynamiquement et des informations peuvent être acquises au fil de l'eau pour capitaliser les connaissances. Ces connaissances peuvent alors être utiles de deux manières : permettre aux agents d'agir de façon autonome et réduire la charge de travail de l'utilisateur ou encore permettre aux agents d'avoir une représentation précise des croyances, comportements et compétences du maître qui peuvent être utiles lorsque celui-ci ne fait plus partie du groupe de travail ou de l'organisation.

Malgré tout, la complexité de telles techniques limite aujourd'hui l'utilisation des systèmes d'agents pour la capitalisation des connaissances. Dans les applications de capitalisation des connaissances le rôle de l'assistant consiste à (i) identifier les besoins des utilisateurs (à travers une communication explicite ou implicite) ; (ii) contracter des agents service pour la réalisation des tâches ; (iii) afficher les réponses de façon adéquate et (iv) acquérir des informations au fil de l'eau pour l'organisation des connaissances.

Avec ces idées en tête, nous allons présenter au long de cette section un prototype d'agent assistant personnel développé pour le projet AACC[10] pour améliorer la collaboration entre groupes d'étudiants français et américains de l'UTC et l'ISU (Iowa State University) pour des études d'ingénierie mécanique. L'assistant a été développé pour permettre l'accès à des documents, à des outils de communication (synchrone et asynchrone), à des tâches du domaine (génie mécanique) ou encore à des informations des collègues. En effet, l'agent assistant personnel enregistre les actions des utilisateurs dans l'espace de travail de l'utilisateur et ensuite les utilise pour faire des propositions concernant la tâche courante. Par conséquent, le travail de l'utilisateur peut être contextualisé et attaché à une tâche du domaine. Les activités indexées sont ensuite organisées sous forme de préférences qui sont mises en place pour proposer automatiquement des documents, des traces de communications ou solutions anciennes qui peuvent aider à résoudre le problème courant. L'agents assistants personnels peut également communiquer avec d'autres assistants pour partager l'information. Par conséquent, le SMA formé par les agents assistants personnels sert à partager les connaissances et les expériences entre les participants du projet. Cette caractéristique est typiquement présente dans des systèmes de capitalisation des connaissances, comme par exemple dans (de Azevedo, 1997) ou (Schlichter et collaborateurs, 1997).

[10] Le projet AACC (Agents d'Aide à la Conception Coopérative) est un projet de collaboration entre le laboratoire CNRS HEUDIASYC de l'UTCet le laboratoire LaRIA de l'UPJV.

4.4.1. La structure générale de l'agent assistant personnel

Nous avons adopté la définition d'agent assistant personnel introduite par Negroponte (1995) et discutée dans la section 2.4. Dans cette application l'agent assistant personnel est spécialisé dans l'interaction avec l'utilisateur. Comme nous avons vu dans la section 2.4, Barthès et Ramos ont proposé une architecture générale d'un agent assistant personnel composée des éléments suivants : *actions*, *tâches et projets*, *ontologies*, *monde*, *soi-même*, *utilisateur* et encore une interface réseau et un mécanisme de contrôle.

Notre AP réalise trois des éléments proposés : actions, ontologies et modèle de l'utilisateur. Nous discutons chacun de ces composants concernant l'agent assistant personnel pour la capitalisation des connaissances :

- Actions : les actions demandées par l'utilisateur sont représentées à l'intérieur de l'agent comme des tâches. Elles sont étudiées dans la section 4.4.5. La seule tâche réalisée de façon autonome par l'agent assistant personnel est la recommandation de documents intéressants selon les préférences de l'utilisateur et l'activité courante du maître ;

- Tâches et projets : nos agents assistants personnels s n'ont pas de comportement délibératif ni représentations de haut niveau ; ils possèdent seulement des actions de bas niveau ;

- Le monde : Dans cette application le monde n'est pas représenté explicitement car il n'est pas composé de nombreuses entités, mais seulement d'agents impliqués dans une communication directe et très simple ;

- Soi-même : Dans ce travail tous les agents assistants personnels sont identiques. Ils n'ont pas de services particuliers. Par conséquent, nous n'avons pas besoin de représenter les agents explicitement. Par ailleurs, nous n'utilisons pas les communications entre les agents pour reconnaître les agents du système ou pour un autre motif quelconque car nos agents ne sont pas nombreux et travaillent normalement en paires ;

- Ontologies : Les ontologies sont utilisées dans ce travail pour modéliser les objets et les relations entre les concepts du domaine. Par ailleurs, ces concepts sont mis en place dans l'indexation des documents et l'aide personnalisée ;

- Le modèle du maître : Un point important dans les APs consiste à représenter les préférences des utilisateurs. Dans la section 4.3.3 nous avons discuté comment un agent assistant personnel peut fabriquer un modèle de préférences du maître à partir de documents de référence. Le modèle de préférences constitue normalement le *modèle à long terme* de l'utilisateur, c'est-à-dire, toutes les activités passées sont utilisées dans la construction du modèle. Dans ce travail un autre modèle est aussi mis en place : le modèle *à court terme*. Ce modèle est fabriqué à partir de l'activité courante de l'utilisateur. Ensuite l'utilisateur peut choisir quelle stratégie sera utilisée par l'agent assistant personnel dans la recommandation des nouveaux documents et/ou tâches.

4.4.2. L'architecture du projet AACC et le rôle de l'agent assistant personnel

La qualité d'un projet de collaboration est souvent proportionnelle au niveau des échanges réalisés entre les membres du projet. La communication entre les membres donc, joue un rôle essentiel. Malheureusement la communication entre les participants peut être imparfaite et insuffisante, notamment quand des personnes de domaines différents, avec des expériences et des niveaux de connaissances différents communiquent. Pour cela nous avons besoin de formaliser les connaissances de chacun et de les rendre disponibles. Ainsi, les agents assistants personnels du projet AACC jouent deux rôles différents : la formalisation des expériences individuelles et le partage avec le reste du groupe. Les sections suivantes donnent plus de détails sur comment le système d'agents du projet AACC a été conçu.

Architecture du projet AACC

Dans le projet AACC deux groupes d'étudiants ont besoin de communiquer pour développer ensemble un artefact eletro-mécanique. La Figure 30 montre l'architecture générale du système. Les groupes communiquent à travers des agents assistants personnels. Chaque agent assistant personnel, à son tour, utilise un ensemble d'agents services capables d'envoyer des messages d'e-mail ou rechercher l'information dans une base de documents. Un agent spécial nommé Agent Profile est responsable de la gestion des profils des utilisateurs et doit fournir des informations personnalisées à l'agent assistant personnel. Par ailleurs, un outil collecticiel standard fourni une base de documents du projet. Les agents sont normalement placés dans une coterie (Tacla, 2001) et communiquent à distance à travers l'Internet. Les communications internes à la coterie concernent normalement l'interaction entre le maître et l'agent assistant personnel et entre l'agent assistant personnel et les agents services.

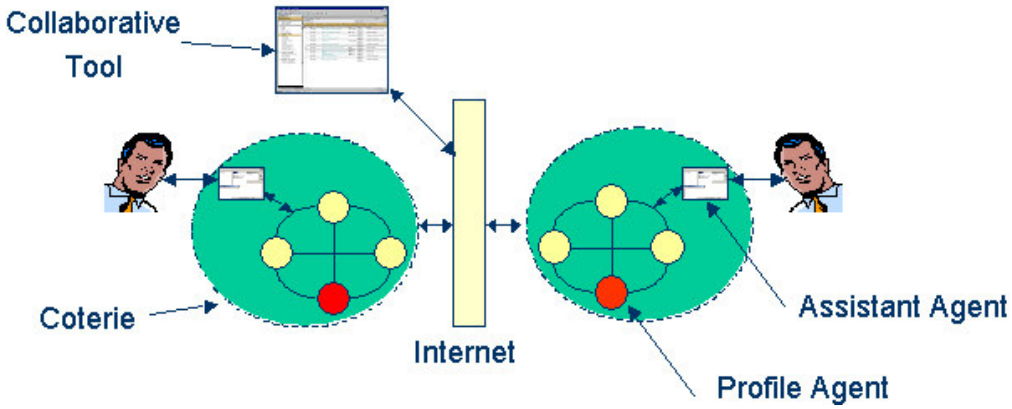

Figure 30 Architecture du système.

Communication entre l'utilisateur et l'Agent Assistant

La communication entre le membre du projet et l'agent assistant personnel permet à l'utilisateur de demander l'exécution d'une certaine tâche, de demander des informations sur les autres membres du projet et de demander de l'aide par rapport un problème. Également, l'assistant acquiert automatiquement des informations sur le comportement de l'utilisateur. Pour cela, notre approche combine la communication explicite et implicite. Dans une communication explicite l'utilisateur doit fournir des détails suffisants pour l'accomplissement de la tâche. En revanche dans une communication implicite l'agent analyse le travail de l'utilisateur et utilise sa mémoire pour proposer une liste de problèmes et des solutions qui peuvent aider l'utilisateur à avancer. Pour mettre en œuvre un tel mécanisme nous avons combiné deux modes d'interaction : des IHMs (Interfaces Homme-Machine) classiques et le langage naturel.

Des applications IHM classiques utilisent des fenêtres, boutons ou boîtes de texte où l'utilisateur peut donner des informations. Ces applications sont suffisantes pour de simples interactions où toute l'information nécessaire pour la solution du problème peut être ajoutée dans des formulaires. En revanche, le langage naturel est mis en œuvre dans des interactions plus complexes où l'utilisateur peut rentrer l'information plus librement.

Communication entre l'assistant et les agents de service

Normalement, les agents assistants personnels communiquent avec les agents services quand une nouvelle tâche est déclenchée, par exemple, l'agent assistant personnel doit garder un document dans la base de documents. Dans ce cas, l'agent assistant personnel récupère les informations nécessaires pour l'accomplissement de la tâche auprès de l'utilisateur (avec

langage naturel ou des formulaires) et envoie des requêtes aux agents service. Le protocole mis en place pour la distribution des tâches entre les agents est le Contract-Net (Smith, 1979). Certains auteurs, par exemple Shen et co-auteurs (2001) soutiennent que le Contract-Net n'est pas capable de résoudre des conflits entres les agents. En tout cas, dans notre application nos agents ne sont pas nombreux et normalement ne partagent pas des informations sensées contradictoires.

Communication entre l'assistant et l'agent profil

La communication entre l'agent assistant personnel et l'agent profil est très simple. Lors du démarrage d'un agent assistant personnel par l'utilisateur, l'agent assistant personnel envoie une requête à l'agent profil en demandant le profil de l'utilisateur. Les messages échangés sont du type Request/Answer selon le Contract-Net. Ce service est nommé « give-profile » et les arguments du message sont l'identification de l'utilisateur, le mot de passe et l'identification de l'agent assistant personnel. Les modifications faites par l'utilisateur dans le profil sont mises à jour localement. Le profil est renvoyé vers l'agent profil de temps en temps pour la mise à jour et aussi lorsque l'utilisateur sort du système. Pour transmettre le profil, l'agent assistant personnel envoie un « Request » concernant le service « save-profile » en fournissant l'identification de l'utilisateur et le profil. L'agent profil utilise donc le profil de l'utilisateur pour fabriquer un modèle de préférences ce qui sera discuté plus en détails dans la section 4.4.8.

4.4.3. Le rôle de l'assistant

Le rôle de l'agent assistant personnel consiste à identifier les besoins de l'utilisateur, exécuter des tâches, partager des connaissances entre les utilisateurs, fournir une aide de façon autonome et acquérir des connaissances au fil de l'eau :

- L'identification des besoins de l'utilisateur : les besoins de l'utilisateur sont fournis explicitement ou découverts automatiquement à partir des actions de l'utilisateur. Dans le dernier cas nous avons besoin de créer des modèles de tâches pour ensuite analyser et enregistrer le comportement de l'utilisateur. Nous discutons cet aspect dans la section 4.4.5. Par ailleurs, à travers de l'analyse des tâches des utilisateurs nous découvrons le modèle des préférences ;

- Le partage des connaissances: Pour améliorer la collaboration entre les membres d'un projet, ceux-ci doivent partager des informations. La question est de savoir quel genre d'information est important pour la collaboration. Par exemple, connaître l'activité courante de son collègue peut aider dans l'activité en cours. Schlichter et collaborateurs soutiennent que « Awareness is part of the 'glue' that allows groups to be more effective than individuals » (Schlichter et al., 1997). Nous soutenons que l'agent assistant personnel peut aider directement ce processus en fournissant des informations sur les utilisateurs en ligne. Par conséquent, les utilisateurs sont poussés à communiquer (de façon synchrone ou asynchrone) et à jeter un coup d'œil sur le travail des collègues. Dans cette forme de collaboration deux aspects principaux semblent évidents : (i) les personnes communiquent plus et par conséquent, échangent plus d'informations personnelles et (ii) les tâches observées dans le travail des autres collègues peuvent être récupérées et réutilisées, par conséquent, les utilisateurs peuvent « apprendre » à partir de l'expérience des autres ;

- L'aide proactive : L'autonomie est un comportement souhaitable dans un agent assistant personnel. Nous avons développé une forme simple d'autonomie qui concerne l'aide automatique. Par conséquent, les utilisateurs n'ont pas besoin de demander d'aide, mais elle est toujours présentée de façon adéquate. Une demande d'aide est toujours déclenchée automatiquement lors que l'utilisateur passe un certain temps sur une tâche ou la modifie. Le système présente donc des propositions intéressantes pour la tâche en cours à partir de l'analyse de profil des utilisateurs ;

- L'aide personnalisée : Le système peut identifier des problèmes et répondre à des questions à partir de descriptions libres en langage naturel. Pour cela le système doit être capable de reconnaître le vocabulaire des termes utilisés par l'utilisateur. Dans notre système, ces termes font partie d'une ontologie représentée sous forme d'une base de connaissances. Ces concepts peuvent appartenir à plusieurs classes différentes (*People*, *Task*, *Design*, etc.) et compter plusieurs synonymes et variations linguistiques. L'agent assistant personnel met en œuvre ces concepts sous forme d'un système de dialogue capable de répondre à des questions et exécuter certaines tâches ;

- Modèle de préférences : L'agent assistant personnel intercepte les actions de l'utilisateur et les stocke sous forme d'un profil. À partir du profil de chaque utilisateur un modèle de préférences est mis en œuvre pour la suite. Les actions interceptées peuvent concerner la

création de tâches simples comme « *send an e-mail* » ou encore de tâches complexes du domaine comme *« design a vacuum cleaner »*. Les tâches des profils des utilisateurs peuvent ensuite être récupérées et réutilisées par tous les utilisateurs.

4.4.4. L'interface de l'utilisateur

L'interface du prototype (Figure 31) a été conçue pour tester les idées discutées dans les sections précédentes. Sur le côté gauche de la fenêtre principale est affichée la hiérarchie des tâches créées par l'utilisateur. Sur le côté droit de la fenêtre est affichée la tâche courante avec les tâches et sous-tâches. Dans l'exemple de la Figure 31 l'utilisateur *Cacic* utilise la bibliothèque de tâches pour créer un aspirateur électrique « vacuum cleaner device. » Pour accomplir cette tâche, le concepteur réalise plusieurs tâches (envoi d'e-mails, écrit des rapports, fait la veille technologique et discute avec les collègues). Ensuite l'utilisateur travaille sur les sous tâches de façon identique.

Quand une nouvelle tâche est créée le concepteur doit fournir des informations complémentaires comme la définition de la tâche, des commentaires, la durée, etc. Par ailleurs, d'autres tâches (composées ou atomiques) peuvent être ajoutées à la tâche courante. Le système dispose d'une bibliothèque de tâches du domaine qui décrit la structure et contient toutes les informations importantes concernant les tâches. L'utilisateur peut, ensuite, créer des instances de ces tâches et les ajouter à son travail. Evidemment, les tâches peuvent être modifiées en fonction des besoins de l'utilisateur.

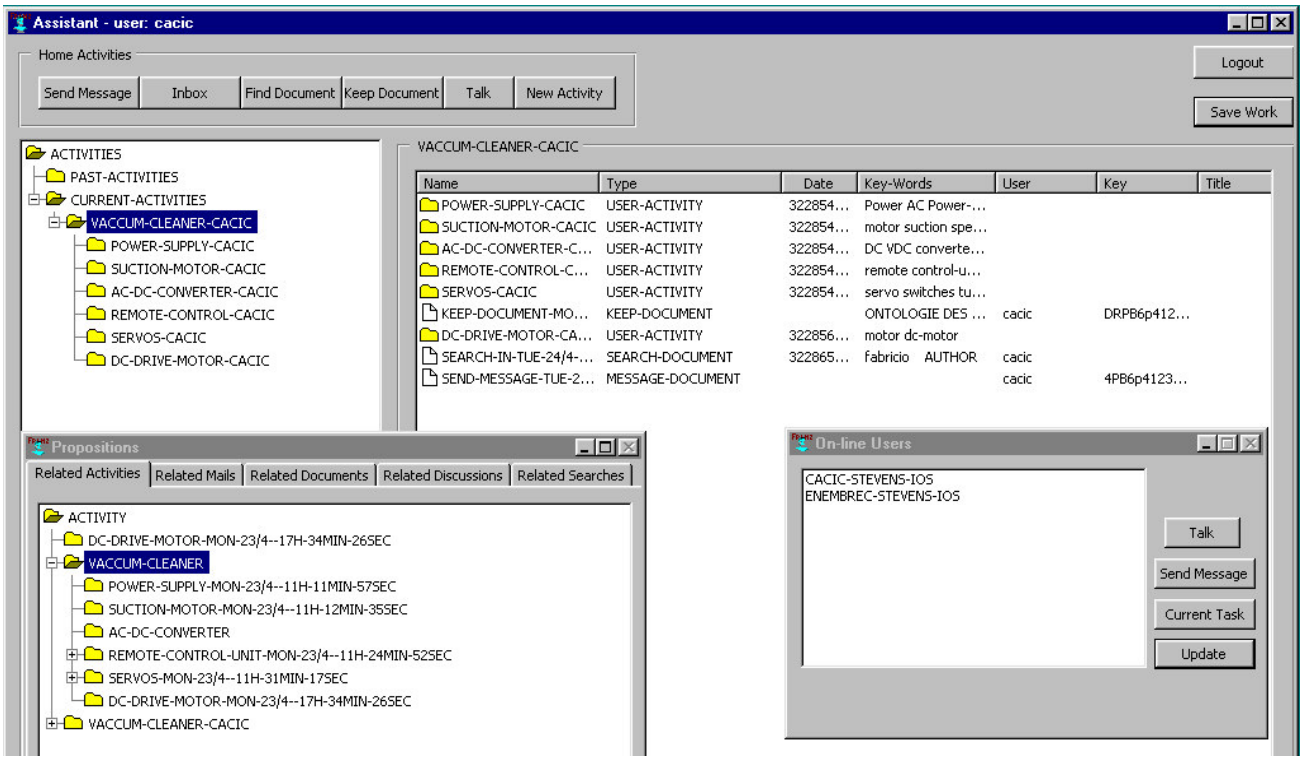

Figure 31 L'interface de l'assistant.

4.4.5. Le modèle de tâches

Les actions des utilisateurs sont composées des tâches enregistrées dans le profil des utilisateurs sous forme de documents. Les tâches enregistrées dans un profil sont accessibles à n'importe quel assistant. Pour chaque tâche le système fabrique une représentation XML. Toutes les tâches ont une structure générique composée par un *nom*, une liste récursive de *sous-tâches*, la représentation du *contenu*, et le *type* (Atomique ou Composée). Un exemple simple est donné dans la Figure 32. Le contenu de la tâche est spécifique selon le type de tâche et contient des informations complémentaires comme *mots-clés*, *auteur* ou *durée*. Tâches composées possèdent deux champs additionnels : *description* et *solution*.

```
- <TASK>
    <name>VACCUM-CLEANER</name>
  + <Sub-activities>
  - <contents>
      <Contents-Type>USER-TASK</Contents-Type>
      <DATE>3228541654</DATE>
      <DESCRIPTION>Here we are going to detail the vaccum.</DESCRIPTION>
      <SOLUTION />
      <DURATION>999</DURATION>
      <KEYWORDS>Vacuum Cleaner</KEYWORDS>
    </contents>
    <type>COMPOSED</type>
  </TASK>
```

Figure 32 Structure d'une tâche.

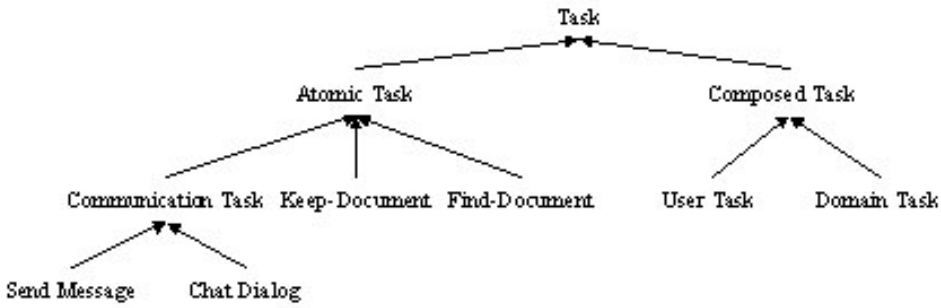

Figure 33 Types de tâches.

Sur la Figure 33 nous avons organisé de façon hiérarchique les types de tâches modélisées dans le système. Les tâches quotidiennes peuvent être structurées ou non-structurées. Les tâches structurées sont normalement enregistrées dans la bibliothèque de tâches comme de tâches du domaine. Ces tâches sont utilisées sous forme de modèles XML (*Domain Task* dans la Figure 33). Pour nous, « écrire un rapport » est un exemple d'une tâche non-structurée. Celles-ci sont créées par l'utilisateur (*User Task* dans la Figure 33) et n'ont pas un modèle XML prédéfini. La définition de chaque tâche modélisée dans l'agent assistant est indiqué dans le Tableau 7.

Tâche	Définition
Send-Message	Est une communication d'e-mail. L'agent assistant personnel utilise un client d'e-mail d'un outil collaboratif
Chat-Dialog	L'utilisateur peut demander une discussion avec un autre utilisateur connecté à un autre AP
Keep-Document	L'utilisateur peut ajouter des documents à la tâche courante. Comme dans les messages d'e-mail, l'agent assistant personnel utilise les services d'un

	dépôt de documents d'un outil collaboratif
Find-Document	L'utilisateur peut rechercher des documents dans la base avec l'agent assistant personnel. L'agent assistant personnel ensuite peut enregistrer des recherches réussies dans le profil de l'utilisateur
User Task	L'agent assistant personnel permet à l'utilisateur de créer ses propres tâches. Certaines tâches quotidiennes comme "write a paper," ou "write a PhD. Thesis" sont naturellement non-structurées. Par conséquent, l'utilisateur doit être libre pour travailler de la manière qui lui convient le mieux
Domain Task	L'agent assistant personnel possède une bibliothèque de tâches du domaine. Une tâche du domaine possède une structure hiérarchique (tâche/sous-tâche) qui donne des renseignements sur comment la tâche doit être accomplie

Tableau 7 Types de tâches disponibles.

4.4.6. Group Awareness

L'agent assistant personnel essaye de promouvoir la communication entre les membres affichant des informations sur des utilisateurs qui travaillent avec assistants (la petite fenêtre du côté droit de la Figure 31). L'utilisateur peut donc établir directement une communication avec les collègues ou encore regarder les tâches qui sont réalisées à distance par un collègue. Dans ce deuxième cas les tâches du collègue sont affichées exactement comme ses tâches privées dans une autre fenêtre. L'utilisateur peut ensuite ajouter ces tâches à son travail.

4.4.7. Comportement pro-actif

L'agent assistant personnel est capable de faire des propositions. Il peut proposer des tâches trouvées à partir des profils des utilisateurs qui prendront en compte la pertinence par rapport à la tâche courante (petite fenêtre du côté gauche de la Figure 31). Pour faire une proposition l'agent assistant personnel envoie une requête à l'agent profil passant la représentation XML de la tâche courante. L'agent profil recherche des tâches similaires à cette représentation regardant le profil de tous les utilisateurs et rend une réponse à l'agent assistant personnel.

Les sections suivantes donnent plus de détails sur le mécanisme de récupération de tâches mis en place par l'agent profil.

Les propositions sont présentées dans un ordre décroissant selon la similarité. Les activités des autres utilisateurs ont une priorité sur les activités du même utilisateur. Ainsi comme dans la section précédente, ces tâches peuvent être récupérées et ajoutées à la tâche courante.

4.4.8. Organisation et récupération de tâches

Étant donné que des tâches sont organisées sous forme de documents, l'agent met en œuvre des techniques de récupération d'information et de classification de documents pour gérer les tâches. L'idée consiste à apprendre un modèle de représentation pour la collection de documents et ensuite l'utiliser pour organiser, récupérer et proposer d'autres documents selon une certaine similarité de contenu. Pour cela nous avons utilisé les même approches de représentation de documents que dans la section 4.3.3.

À la différence de la technique de la section 4.3.3 nous n'avons pas besoin de rechercher parmi tous les termes du contenu de la tâche ceux qui vont composer le vecteur de caractéristiques. En revanche, ce choix prend en compte seulement les termes existants dans l'ontologie de concepts comme nous le montre la Figure 34. Nous avons également limité la taille des vecteurs à 1000. Des variations linguistiques comme singulier et pluriel par exemple sont prises en compte dans la conception de l'ontologie.

La Figure 34 et la Figure 35 montrent comment une tâche est représentée à l'intérieur du système. La Figure 34 montre la représentation d'une tâche complexe. La Figure 35 montre les termes de l'ontologie présents à l'intérieur du document.

4.4.9. Représentation des préférences de l'utilisateur

Dans ce travail nous n'utilisons pas un algorithme d'apprentissage car nous n'avons pas besoin de créer un modèle générique de l'utilisateur à partir des données d'apprentissage. Dans notre cas, les données d'apprentissage sont mises en forme comme des documents qui représentent les tâches. Les préférences sont les documents eux-mêmes.

Ainsi, les documents sont utilisés pour l'identification de ceux qui semblent plus concernés par le document qui représente la tâche courrante de l'utilisateur. Cette tâche de classification est mise en œuvre avec une mesure de similarité entre les vecteurs. Dans ce travail nous

utilisons empiriquement la mesure du cosinus. Le calcul du cosinus se fait selon la formule (5).

$$\text{sim}(\mathbf{d}_i,\mathbf{d}_j)=\cos(\mathbf{d}_i,\mathbf{d}_j)=\frac{\mathbf{d}_i\cdot\mathbf{d}_j}{\|\mathbf{d}_i\|_2\times\|\mathbf{d}_j\|_2} \quad (5)$$

où "·" est le produit scalaire entre les deux vecteurs et $\|\mathbf{d}\|_2$ est la taille du vecteur **d**.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
- <Task>
    <name>AC-DC-Conv-01</name>
    <Sub-task />
  - <contents>
      <Contents-Type>DOMAIN-TASK</Contents-Type>
      <USER>enembrec</USER>
    <SUBJECT>Construction of the AC-DC Converter for vacuum cleaner
    </SUBJECT>
    <DATE>3237625095</DATE>
    <KEYWORDS>AC DC CONVERTER POWER POWER-SUPPLY</KEYWORDS>
    <DESCRIPTION>The second parallel line will go to a power supply
        which will convert the 120 VAC to 4.8 or about 5 VDC which is
        needed to power the drive motor and the receiver box. This 4.8
        or 5V is needed because that is what the receiver box needs; no
        other voltage will work.
    </DESCRIPTION>
    <SOLUTION>I've asked many stores about their power supplies and
        they are either too big, too costly or not the correct voltage
        altogether. I think the best alternative is to buy a wall converter
        and just wire that into the system. These are the small boxes you
        just plug into an outlet to charge your phones or run small tape
        players, etc. They usually have the cylindrical plug on the end,
        which we could just cut off and hard-wire it in. If anyone can find a
        website about these small converters let me know or throw in your
        ideas.
    </SOLUTION>
    <DURATION>15</DURATION>
  </contents>
  <type>ATOMIC</type>
  </Task>
```

Figure 34 Une tâche complexe.

Term:	AC/VAC	DC/VDC	Converter	Power	Power-Supply	Receiver-Box	Voltage	Parallel-Line	Drive-Motor	Outlet	Cylindrical-Plug
Frequency:	2	2	4	2	3	2	2	1	1	1	1

Figure 35 Vecteur de termes qui représentent la tâche de la Figure 34.

4.4.10. L'aide en langage naturel

Dans cette section nous présentons le module d'aide personnalisée fourni par l'assistant. Le module d'aide permet aux étudiants de demander des informations sur les concepts du

domaine. Nous avons essayé d'établir une communication naturelle permettant aux étudiants de poser des questions le plus librement possible. Pour cela nous avons utilisé la plupart des fonctionnalités du système de dialogue de la section 3.1, comme des grammaires, des actes de langage, et l'analyse sémantique des énoncés.

La représentation des concepts du domaine s'appuie sur le système MOSS (Barthès, 1994). Les concepts ont été organisés sous forme d'un réseau sémantique. MOSS permet l'indexation des termes par des synonymes et le partage de points d'entrée (indexés) entre différents concepts. Les concepts de cette ontologie ont été identifiés à partir des documents du domaine conçus pendant les projets précédents (voir Annexe IIIA).

La création d'une ontologie est un processus très difficile qui demande beaucoup d'effort dans l'analyse empirique des documents et aussi dans l'usage d'outils lexiques dans l'identification et la définition des concepts importants pour le domaine (Thouvenin et al., 2003). Chaque concept dans l'ontologie possède une définition formelle et un ensemble de points d'entrée (des termes qui font référence au concept).

Chaque tâche représente un concept. Les concepts concernant des tâches possèdent des propriétés additionnelles comme la *fonctionnalité* et l'*explication*. La fonctionnalité contient des informations concernant les objectifs de la tâche et l'explication raconte comment la tâche doit être accomplie (voir Annexe IIIB pour le code source).

La décomposition des tâches est mise en forme avec des liens sémantiques "has-subtasks". La Figure 36 montre une partie de l'ontologie. Dans la Figure 36 le « vacuum cleanner » est l'artefact fabriqué dans la tâche VC-Prod (Vacuum Cleanner Production). *VC-Prod* possède quatre sous-tâches : AC-Mot-Prod (AC Motor Production), DC-Mot-Prod (DC Motor Production), AC-DC-Conv-Prod (AC-DC Current Converter) et RC-Production (Remote Control Production).

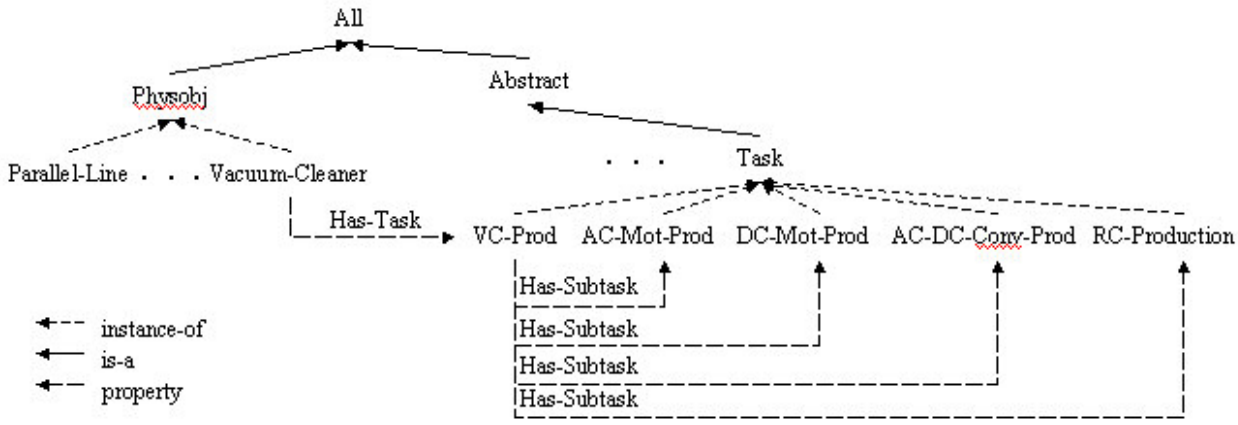

Figure 36 Hiérarchie de Concepts.

L'utilisateur peut demander des informations sur les tâches en utilisant une terminologie spécifique au domaine d'ingénierie mécanique. La Figure 37 montre quelques exemples de questions posées à l'assistant. Lorsqu'une explication sur *VC-Prod* est demandée, une petite description de chaque sous-tâche est affichée. Le lecteur intéréssé par des dialogues plus complexes peut consulter la section 3.3 de ce mémoire.

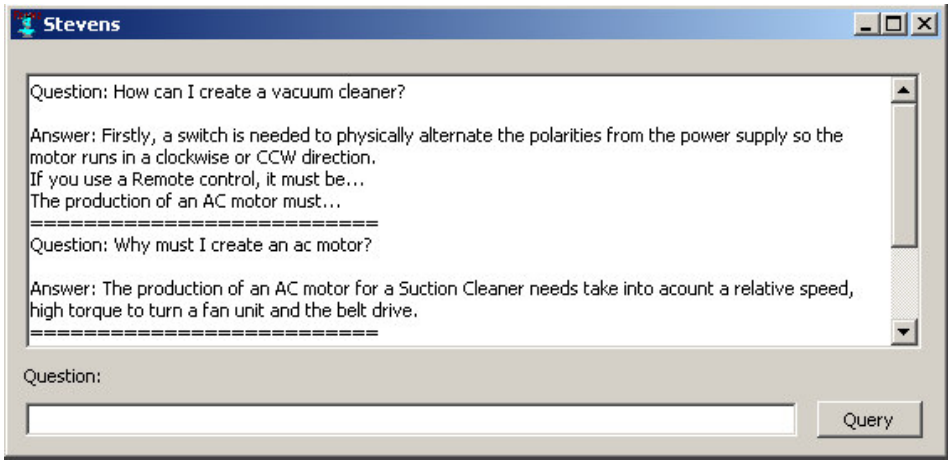

Figure 37 Fenêtre d'aide.

4.4.11. Discussion sur les agents et la capitalisation des connaissances

Dans cette section nous discutons quelques résultats préliminaires du projet AACC et les avantages à utiliser des SMAs pour le travail collaboratif.

L'Assistant personnel et le projet AACC

Depuis le deuxième semestre de l'année 2001 nous avons observé le comportement des étudiants de génie mécanique pendant le cours de design. Dans le deuxième semestre de

l'année 2002 les étudiants ont utilisé un outil de gestion documentaire développé au sein du labo et en même temps nous avons développé et testé les mécanismes présentés dans les sections précédentes. Le prototype développé n'a pas été mis en place pour une utilisation massive des étudiants car l'interface développée est fonctionnelle mais pas très adaptée, car elle n'a été conçue que pour tester les mécanismes. Cependant nous discutons dans la suite quelques questions qui nous ont frappé pendant l'évolution du projet.

L'une des questions les plus importantes dans ce projet était de concilier les *contraintes pédagogiques* et le *partage et récupération d'information*. En effet, l'environnement pédagogique est différent d'un environnement professionnel car nous avons besoin de contrôler l'information pendant le processus d'apprentissage. Par exemple, dans un environnement professionnel les personnes normalement ont une grande expérience et des connaissances suffisantes pour analyser des informations venant d'autres projets. En revanche, des étudiants souvent ne font pas très attention à la fiabilité des sources d'information disponibles. Un autre problème consiste à aider les étudiants sans brider leur créativité. Ainsi le système doit fournir plusieurs niveaux de confidentialité de l'information car certains étudiants n'hésiteront pas à réutiliser les idées innovantes des collègues.

Les SMA dans le travail collaboratif

Les SMAs rendrent possible la distribution et la réutilisation des sources de connaissances. Ainsi, des informations particulières peuvent être mises ensemble dans la solution d'un problème plus général. Pour nous, ce principe est fondamental. Nous considérons que chaque utilisateur est une source d'information. Le rôle principal de l'agent assistant personnel est de formaliser ces sources et les mettre à disposition d'autres APs. Lorsque les agents possèdent une représentation du maître ils peuvent coordonner ses activités, et, également, lorsque les agents connaissent les compétences des utilisateurs ils peuvent décider qui doit faire quoi et ainsi aider dans la répartition des tâches. Malgré le fait que nous n'abordons pas directement le problème de coordination de tâches, nous pouvons dire que les connaissances partagées par les APs sur les collègues peuvent aider les utilisateurs dans les choix des tâches et coordination du travail.

La communication entre les participants normalement est fondamentale dans un processus de travail collaboratif. Par conséquent, les utilisateurs doivent faire usage d'outils de communication connus par tout le monde et simples de prise en main. Dans ce travail les

outils de communication (synchrone et asynchrone) sont mis en œuvre par l'agent assistant personnel, qui dispose d'une messagerie, un outil de discussion en-ligne, une base de documents et un module de partage de tâches. Avec ces outils nous espérons couvrir tous les aspects d'un processus de communication réussi dans un environnement collaboratif.

4.5. Conclusions

Dans ce chapitre nous avons examiné comment des assistants personnels peuvent faire évoluer les connaissances concernant leur maître pour adapter et donc personnaliser leurs services.

La personnalisation des services est aujourd'hui un sujet très actif de recherche, qui peut être mis en œuvre dans n'importe quel système interactif. Le besoin de personnalisation semble évident si nous envisageons dans le futur des machines capables de remplacer les êtres humains dans certaines activités. Pour cela, le système doit avoir une représentation suffisamment étoflée et complexe du maître capable de reproduire artificiellement les désirs, les intentions et l'avis du maître face à une certaine situation. Dans certaines applications, l'analyse documentaire peut aider les agents d'information sur ces aspects. Pour démontrer cela nous avons développé certaines techniques qui permettent à un SMA de capturer les préférences des utilisateurs et ensuite distribuer des documents de façon personnalisée. Ces techniques ont été réutilisées pour la distribution et la recommandation personnalisée de connaissances dans un environnement collaboratif.

Puisque, dans certains domaines, la modélisation de l'utilisateur doit se faire de façon automatique car nous ne disposons pas de connaissances *a priori*, nous avons utilisé dans ce chapitre quelques techniques d'apprentissage fondées sur le raisonnement par mémoire. Dans la première application une mesure de similarité spécifique a été présentée pour retrouver de documents. La deuxième application utilise une mesure classique de similarité pour identifier de représentations des tâches similaires.

5 La méthode ELA (Entropy-based Learning Approach)

Nous avons présenté dans les chapitres précédents plusieurs méthodes d'adaptation pour des agents autonomes. Les mécanismes d'apprentissage mis en place ne sont toutefois pas suffisants, car ils ignorent souvent les contraintes de ressources (temps d'apprentissage et mise en place des modèles) et la complexité de mise en œuvre. Nous présentons dans ce chapitre une nouvelle méthode d'apprentissage destinée aux agents assistants personnels adaptatifs.

Cette méthode a été développée pour répondre au besoin d'adaptation d'un agent assistant personnel. Elle est fondée sur le principe d'apprentissage incrémental et a été conçue pour minimiser les problèmes de complexité et améliorer les performances des algorithmes traditionnels d'apprentissage dans des environnements dynamiques.

Après une présentation détaillée de la méthode, nous comparons ses performances avec celles d'autres travaux pour évaluer son efficacité. En fin de chapitre, nous discutons d'autres approches qui partagent certains concepts avec la méthode proposée.

5.1. Caractéristiques

Dans la méthode proposée, les exemples d'apprentissage qui arrivent sont pris en compte immédiatement. La méthode n'a pas d'étape d'apprentissage. Celui-ci se fait en permanence comme dans les algorithmes MBR.

Dans des algorithmes MBR une mesure de similarité sert normalement à récupérer des exemples dans une base de données, exemples conformes à une description initiale du problème. Dans le problème du contrôle aérien, par exemple, la position courante d'un avion et des avions voisins peut décrire le problème. Le système utilise alors une mesure de similarité pour récupérer le cas le plus voisin et décider de l'action à faire, par exemple, augmenter de vitesse ou changer de direction. Nous avons donné dans le chapitre précédent deux exemples de mesures de similarité : une mesure spécifique pour le problème de classification de documents (voir section 4.3.6) et une mesure standard pour le problème de recommandation d'information (voir section 4.4.8).

Les algorithmes MBR toutefois, ont la nécessité de posséder une mesure de similarité adéquate, ce qui conduit à deux problèmes principaux :

- La mise au point d'une mesure de similarité satisfaisante : une mesure de similarité dépend, en général, du domaine. Normalement, des connaissances du domaine sont nécessaires pour la définition d'une heuristique de comparaison entre les différents exemples ;

- Le temps de calcul nécessaire : généralement l'indice de similarité doit être calculé pour tous les exemples présents en mémoire pour n'en selectionner que quelques-uns. Cela peut rendre la classification extrêmement lente.

Selon Aamodt et Plaza (1994) deux techniques peuvent être mises en œuvre pour la définition d'une mesure de similarité : l'*approche statistique* et l'*approche sémantique*. Dans l'approche statistique un ensemble de caractéristiques facilement identifiables est choisi. Par exemple, dans le cas du contrôle aérien, la vitesse, la vitesse du vent, la position, la distance des avions sont des caractéristiques simples. Dans l'approche sémantique, les caractéristiques possèdent une définition plus complexe qui fait appel à des connaissances du domaine, par exemple, le niveau de turbulence ou encore l'état général de l'appareil.

Une mesure de similarité adéquate doit donc donner des niveaux d'importance différents aux caractéristiques qui définissent un cas. Cela peut être très difficile et demander des connaissances approfondies du domaine, car certains attributs peuvent avoir des types de valeurs très différents.

Dans certaines applications de type MBR, le nombre de cas présents dans la base de données peut augmenter énormément, par exemple, dans des systèmes d'agents autonomes qui ont besoin d'interpréter périodiquement l'environnement, disons, toutes les secondes. Par conséquent, ces applications ne sont pas capables de stocker toutes les situations passées et une stratégie de réduction de données doit être mise en place. Des chercheurs comme Brighton et Mellish (2002) travaillent sur des méthodes génériques capables de supprimer dans la base de données les exemples qui sont superflus ou nocifs pour la prise de décision. Nous avons fait quelques essais sur la réduction des données et nous les présentons dans la section 5.6.

5.2. Le mécanisme d'apprentissage

Notre mécanisme d'apprentissage est divisé en deux étapes: *l'acquisition/représentation* des données et la *classification*.

Pour *l'acquisition/représentation* des données nous utilisons une structure de graphe. Le graphe est construit au fur et à mesure que les exemples arrivent. Chaque nœud du graphe correspond à un ensemble d'exemples (une partition) qui partagent la même valeur pour un attribut déterminé.

L'algorithme de *classification* sélectionne les partitions concernées par les valeurs présentes dans l'exemple, en choisit une et analyse les classes existant dans cette partition, en utilisant une fonction de qualité qui prend en compte le comportement de la classe dans toutes les partitions sélectionnées au début. Avant de présenter le mécanisme de classification nous introduisons la représentation des données sur forme de graphe.

5.2.1. La représentation des données

Des algorithmes classiques d'apprentissage représentent en général les données sous forme de tableaux. Les colonnes du tableau sont les attributs et les lignes sont les exemples. Chaque exemple, ainsi, est composé par un ensemble $e=<e_1, \ldots, e_m>$ de m valeurs où e_i est une valeur quelconque pour l'attribut i et m est le nombre d'attributs. L'ensemble des exemples (la base de données) représente alors l'espace complet de recherche. Ainsi, les algorithmes d'apprentissage doivent explorer tout l'espace de recherche pour trouver des partitions qui représentent les modèles des classes des données. Dans notre approche en revanche, nous

partons du principe que l'espace de recherche peut être réduit avec une représentation sous forme de graphe, ce qui permet de diminuer le travail de l'algorithme de classification.

Nous avons donc développé une structure nommée Graphe de Concepts (GC), dont le but principal est de fournir à l'algorithme d'apprentissage un certain nombre de partitions et par conséquent, de diminuer l'espace de recherche de l'algorithme. Dans le graphe, chaque nœud représente un concept qui correspond à la valeur d'un attribut (dénoté par un rectangle) et chaque lien représente une transition d'un attribut vers un autre. Nous n'attribuons pas de poids aux attributs ni ne faisons aucun traitement ultérieur. L'ordre des attributs (niveau de profondeur de chaque attribut dans le graphe) est donnée par la représentation originale des données. Les concepts et les liens sont créés au fil de l'eau. Un lien entre deux concepts A et B contient les *ids* des exemples qui présentent les valeurs A et B (dénoté par une liste d'*ids* entre crochets).

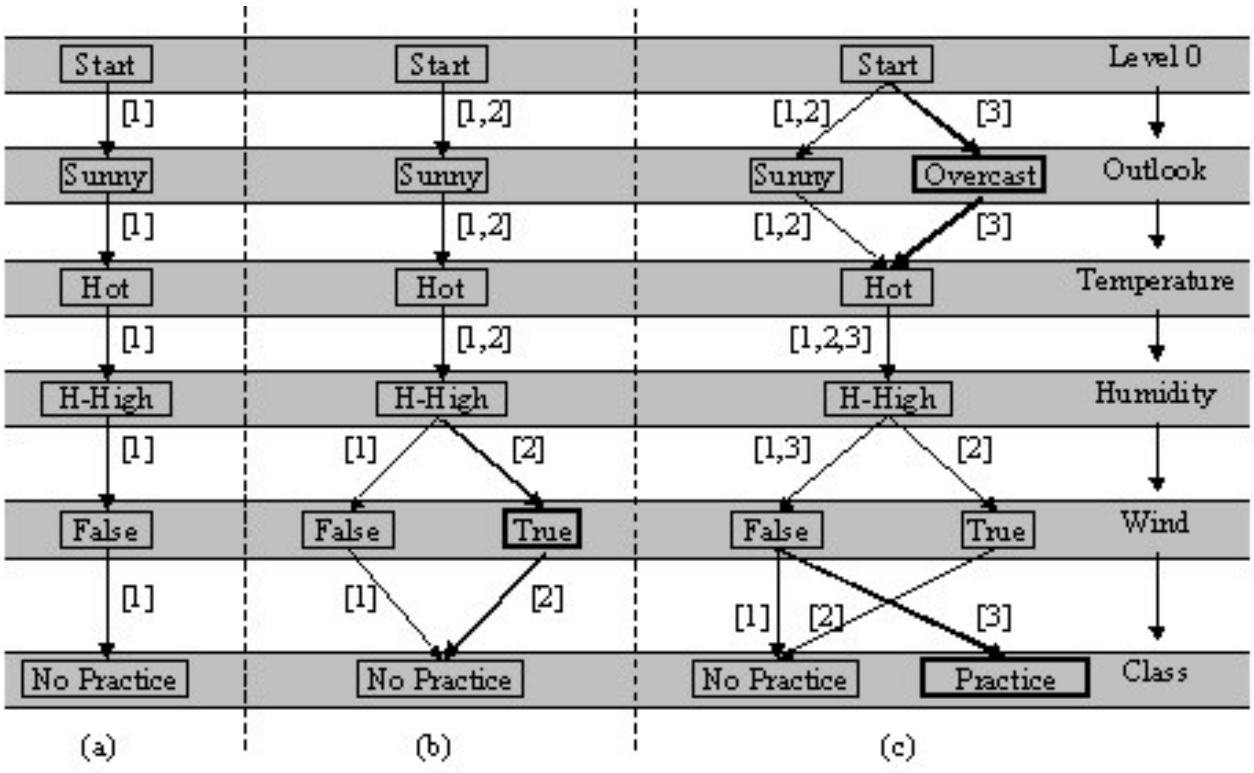

Figure 38 La construction du graphe.

La Figure 38 illustre la construction progressive du graphe avec trois exemples (1, 2, 3) extraits de la base de données de Quinlan (1986). Les concepts et les liens sont créés au moment où ils apparaissent. Ils sont représentés par des lignes grasses. La Figure 38(a) représente l'exemple 1 : ((“Outlook” “Sunny”) (“Temperature” “Hot”) (“Humidity” “H-High”) (“Wind” “False”) (“Class” “No Practice”)). Dans la Figure 38(b) nous ajoutons

l'exemple 2 : (("Outlook" "Sunny") ("Temperature" "Hot") ("Humidity" "H-High") ("Wind" "True") ("Class" "No Practice")). Dans la Figure 38(c) nous ajoutons l'exemple 3 : (("Outlook" "Overcast") ("Temperature" "Hot") ("Humidity" "H-High") ("Wind" "False") ("Class" "Practice")).

La Figure 39 montre le graphe qui représente l'ensemble complet des données de la base de Quinlan (1986). Nous constatons qu'un GC représente seulement les exemples connus de la base de données. Il constitue une description rigide des données qui ne contient pas de généralisations.

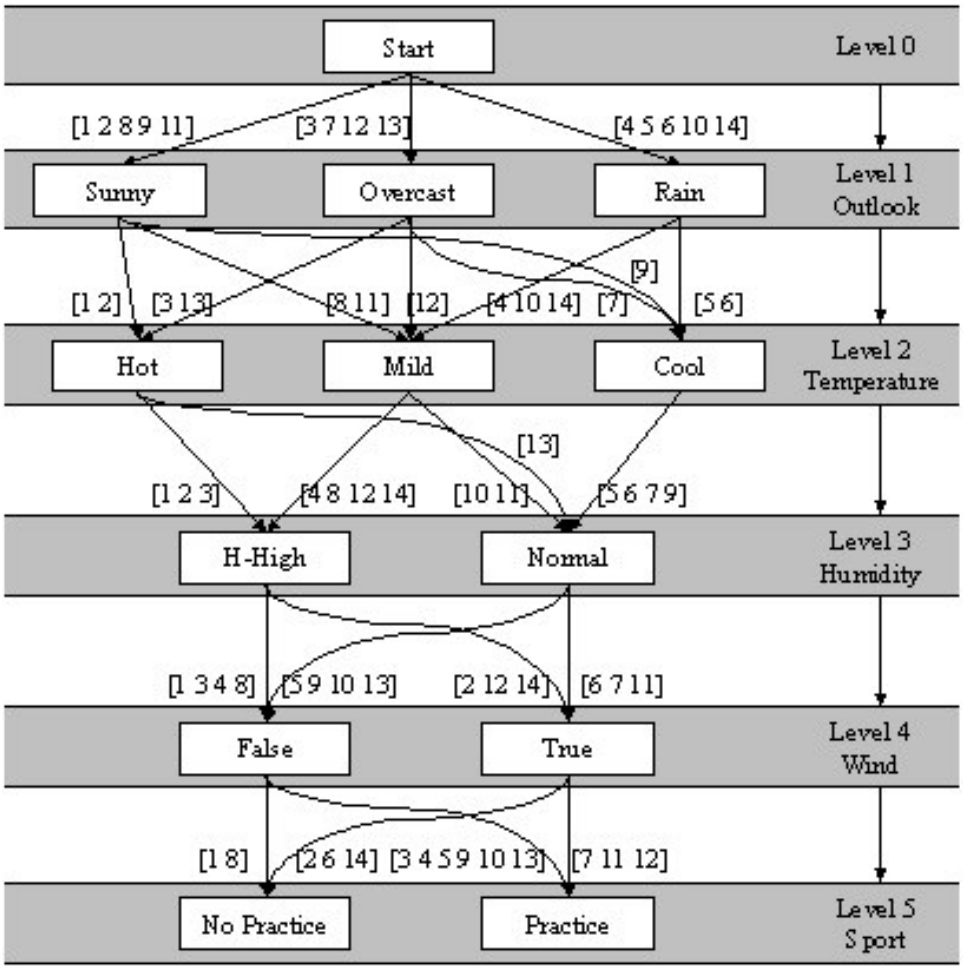

Figure 39 Graphe créé à partir des concepts extraits du travail de Quinlan (1986).

Le nombre de nœuds d'un GC dépend du nombre d'attributs et de la cardinalité de chacun. Ainsi, à partir d'une base de données avec *n* attributs, le nombre de nœuds du graphe est :

$$\sum_{a \in Attributs} cardinal(a)$$

où *a* est un attribut. La profondeur du graphe est *n* + 1.

Contrairement à la majorité des approches qui représentent des données sous forme d'arbres, de graphes ou de règles, le graphe que nous présentons ici n'est pas orienté vers un but. Les techniques traditionnelles représentent les données comme un ensemble d'attributs de *prédiction* et un seul attribut *but*. Dans ce cas, le problème consiste à trouver des relations entre des attributs de prédiction qui permettent de prévoir la valeur de l'attribut but. Le graphe présenté ici est, au contraire, une représentation générale des données où l'attribut but peut changer. Nous discutons cette caractéristique dans la section 5.7.1.

Définition 1. *L'opérateur* α. L'opérateur α peut être appliqué sur chaque nœud *nd* du graphe. Un tel opérateur calcule tous les exemples présents dans les liens qui atteignent le nœud *nd* (fermeture transitive inverse). Par exemple, dans la Figure 39, *α("True")* = [2 6 7 11 12 14].

Définition 2. *Le principe de non-taxonomie.* Il est important de constater que des nœuds descendants ne sont pas des spécialisations des nœuds pères. Par exemple, si *B* est le fils de *A*, $\alpha(B) \subset \alpha(A)$ est généralement vrai, ce qui indique que les attributs ne sont pas ordonnés hiérarchiquement.

5.2.2. Classification

La classification d'un exemple nouveau est faite de la manière suivante : D'abord l'algorithme identifie dans le graphe les concepts retrouvés dans l'exemple à classer. Par exemple, dans la Figure 42 les concepts "Rain", "Cool", "H-High" et "True" sont choisis. Deuxièmement, l'algorithme applique l'opérateur α sur chaque concept et obtient une collection de sous-ensembles. Dans la Figure 42 les sous-ensembles sont [4 5 6 10 14], [5 6 7 9], [1 2 3 4 8 12 14] et [2 6 7 11 14]. Le but est trouver un ensemble d'exemples d'une même classe qui satisfont le nombre maximal de conditions de l'exemple à classifier. Les conditions sont "Outlook=Rain", "Temperature=Cool," "Humidity=H-High" et "Wind=True." Ceci dit, l'algorithme doit trouver pour chaque classe combien d'intersections ne sont pas vides entre les exemples de la classe et les sous-ensembles choisis dans le graphe. La meilleure classe doit avoir des exemples éparpillés tout au long des sous-ensembles d'une façon homogène pour couvrir le nombre maximal de sous-ensembles. Ce critère est exactement le contraire de la mesure d'entropie et d'autres méthodes de mesure de pureté d'ensembles. Ainsi, l'inverse de tels critères peut être employé dans l'identification de la classe correcte.

Définition 3. *Entropie*. Pour calculer la pureté ou le désordre d'un ensemble d'exemples nous utilisons la mesure d'entropie. L'entropie correspond à la quantité d'information en bits nécessaire pour classer un exemple quelconque *e* qui appartient à l'ensemble *E*. L'entropie d'un ensemble est alors donnée par :

$$Entropie(E) = -\sum_{i=1}^{k} \frac{|E_i|}{|E|} \times \log_2 \frac{|E_i|}{|E|}$$

où E est l'ensemble d'exemples et E_i est l'ensemble d'exemples qui appartiennent à la classe i. La Figure 40 illustre le comportement de l'entropie pour deux classes (C_1 et C_2) mutuellement exclusives. Quand une classe est presque certaine, c'est-à-dire, sa probabilité est proche à 1.0, l'entropie est petite. En revanche, quand les classes ont des probabilités équivalentes (0,5) l'entropie est maximale car nous avons besoin de plus d'information pour découvrir quelle est la classe correcte.

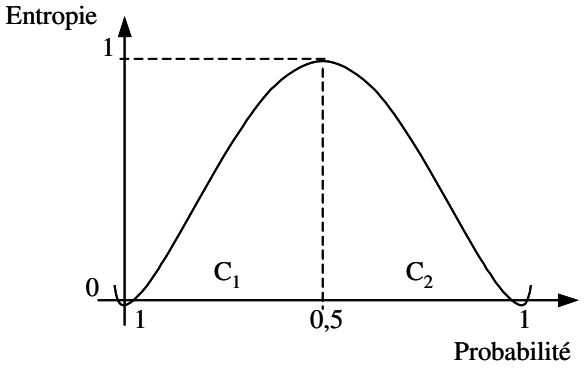

Figure 40 Schéma de l'entropie.

Une question importante consiste à établir une répartition initiale des classes, c'est-à-dire, pour chaque classe, choisir les exemples à considérer. Nous n'utilisons pas tout l'ensemble de données pour trouver une distribution initiale des classes. Plus précisément, nous cherchons parmi les sous-ensembles, celui où la distribution de la classe est la meilleure (où la classification est la plus facile) en utilisant le critère d'entropie. En choisissant un sous-ensemble nous réduisons l'espace de recherche pour la classification.

```
function classify (Partitions)
{
        P = best-partition(Partitions); // minimal entropy
        C = split-class(P);
        foreach C' ∈ C
        {
            E = split-partition(c, Partitions);
            q = insert-priority-queue (class(C'), entropy(E));
         }
         return rear(q); // maximal entropy
}
```

Figure 41 L'algorithme de classification.

La Figure 41 présente l'algorithme de classification. Dans la Figure 41 la fonction « best-partition » sert à choisir la partition initiale (celle qui présente l'entropie la plus faible). « split-class » répartit la partition choisie selon les classes produisant un ensemble de sous-partitions homogènes (qui contiennent des exemples d'une seule classe). La fonction « split-partition » calcule les intersections entre chacune des sous-partitions trouvées dans le pas précédent et les partitions initiales. Ensuite, la classe respective et l'entropie calculée à partir des intersections sont ordonnées. Finalement, la classe qui est en fin de queue de priorité (présente l'entropie la plus importante) est retournée comme réponse.

La Figure 42 illustre un exemple du processus complet de classification. D'abord, l'algorithme trouve la meilleure division des données selon l'entropie ([5 6 7 9]). Cette partition est alors repartie selon les classes (l'ensemble [5 7 9] pour la classe *P* et l'ensemble [6] pour la classe NP). Pour chacune de ces partitions l'ensemble d'intersections est calculé en prenant en compte les partitions initiales ([4 5 6 10 14], [5 6 7 9], [1 2 3 4 8 12 14] et [2 6 7 11 14]). Comme résultat de ce processus nous nous retrouvons avec l'ensemble { [5] [5 7 9] [7] } pour la classe *P* et { [6] [6] [6] } pour la classe *NP*. Ensuite, nous calculons l'entropie de chaque ensemble (1.378 pour la classe *P*) et (1.585 pour la classe *NP*). Finalement nous choisissons la classe qui présente l'entropie la plus importante.

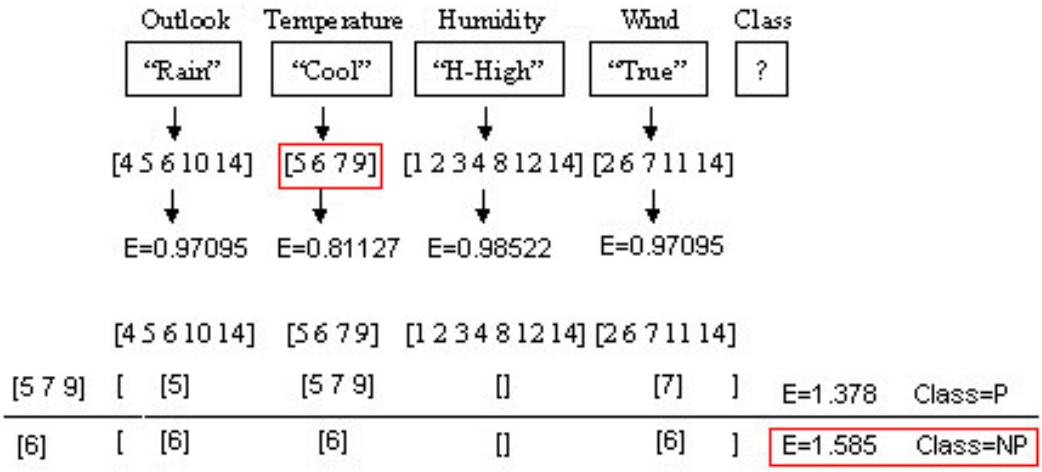

Figure 42 Classification d'un exemple.

D'après les concepts de Quinlan (1986), **si** ("outlook" = "overcast") **et** ("wind" = "true") **alors** classe = NP. Ainsi notre algorithme classe l'exemple correctement. Cela met en évidence la capacité de généralisation de la technique par rapport à des exemples inconnus.

5.3. Pourquoi cela marche-t-il?

L'algorithme de classification reçoit comme entrée un certain nombre de partitions. L'ensemble complet de partitions ne fournit pas les informations suffisantes pour discriminer les classes d'exemples. Notre stratégie consiste alors à réduire le problème en choisissant parmi l'ensemble de partitions, celle qui rendra la classification plus facile. Le choix de la classe prend alors en compte, les classes présentes dans la partition choisie.

La mesure de qualité des classes est calculée en fonction du comportement de la classe dans toutes les partitions initiales. En effet, une classe est optimale si elle est représentée entièrement dans l'ensemble complet de partitions. Évidemment, quand c'est le cas, l'intersection entre la classe et les partitions doit produire des sous-ensembles de taille identique. Cela arrive dans la Figure 42 pour la classe NP, où l'ensemble suivant est obtenu : { [6] [6] [6] } ; Dans ce cas, nous avons trois sous-ensembles de taille 1. Une classe imparfaite doit produire des intersections de tailles variables, par exemple { [5] [5 7 9] [7] } sur la Figure 42. Par conséquent, nous pouvons utiliser ce comportement pour identifier la meilleure classe. Nous avons vu dans la section précédente comment nous utilisons le critère d'entropie maximale pour identifier la classe correcte.

Dans les sections suivantes nous présentons la performance de la méthode en la comparant à d'autres techniques d'apprentissage. Les résultats ont été obtenus à partir de bases de données du domaine public.

5.4. Complexité de calcul

Comme discuté dans le travail de Tsaparas (1999), la complexité d'un algorithme standard fondé sur le principe du voisin le plus proche est *O(nm)* (où *n* est le nombre d'exemples de la base de données et *m* est la dimension de l'espace d'attributs) dans n'importe quel cas, puisque toute la base de données doit être comparée pendant la recherche de l'exemple le plus proche. Quelques techniques ont été proposées pour réduire le temps de classification de ces algorithmes en introduisant le principe des *voisins approximatifs* (Indyk et Motwani, 1998). Dans cette technique, un paramètre ε est utilisé pour définir un seuil de distance que les voisins approximatifs ne peuvent pas dépasser considérant la circonférence autour du point qui représente l'exemple à classer. Néanmoins, pour réduire le temps de classification ces approches ont besoin d'espace supplémentaire pour stocker des représentations complexes de données.

Dans notre approche aucun espace additionnel n'est nécessaire et nous discutons dans cette section sa complexité algorithmique. Dans le Tableau 8 nous avons réécrit l'algorithme de façon à faciliter le calcul de la complexité. La variable *c* correspond au nombre de classes.

Dans la première partie de l'algorithme (lignes 2-11) nous recherchons la meilleure partition. Pour chaque partition de l'ensemble *P* nous vérifions si elle est homogène (contient des exemples d'une seule classe). Si oui, la procédure est finalisée, sinon nous calculons l'entropie de la partition. Nous avons attribué une complexité 1 pour le calcul d'entropie (ligne 6) car cette valeur peut être récupérée directement du graphe si un mécanisme de mémoire cache est mis en place.

	Algorithme	Nombre de calculs
1	function classify (P)	
2	ent = MAX	1
3	FOR i = 1 to m	1 * m
4	IF one-class(P_i) THEN	1 * m
5	RETURN class(P_i)	1
6	ent' = entropy (P_i)	2 * m
7	IF ent' < ent THEN	1 * m
8	ent = ent'	1 * m
9	index = i	1 * m
10	ENDIF	
11	ENDFOR	
12	ent = 0	1
13	FOR i = 1 to (classes P_{index})	1 * c

14	length = 0	1 * c
15	FOR j = 1 to m	1 * m * c
16	$inter_j$ = intersection (P_j, P_{index}^{i})	1 * m * c + (n – 1) * m * c
17	length = length + length ($inter_j$)	3 * m * c
18	ENDFOR	
19	ent' = 0	1 * c
20	FOR j = 1 to m	1 * m * c
21	p = length($inter_j$) / length	3 * m * c
22	ent' = ent'+ (-1 * p * $\log_2$ p)	5 * m * c
23	ENDFOR	
24	IF ent'> ent THEN	1 * c
25	ent = ent'	1 * c
26	class = $class_i$	1 * c
27	ENDIF	
28	ENDFOR	
29	RETURN class	1
		= (n - 1)mc + 14mc + 6c + 7m + 4 = nmc – mc + 14mc + 6c + 7m + 4 = nmc + 13mc + 6c + 7m + 4 =O(nm)

Tableau 8Complexité de l'algorithme de classification.

Dans la deuxième partie de l'algorithme (lignes 13-28) nous analysons chaque classe appartenant à la meilleure partition (P_{index}). Dans la ligne 16, P_j est une des partitions initiales et P_{index}^{i} est l'ensemble d'exemples de la partition P_{index} qui appartiennent à la classe *i*. Le pire des cas se passe lorsque $| P_{index}^{i} | = (n - 1)$. Dans ce cas, il y a précisément seulement deux classes dans P_{index}^{i} et une des classes contient seulement 1 exemple. Sachant que P_{index}^{i} et P_j sont deux ensembles ordonnés, le nombre de comparaisons de l'intersection peut être calculé avec (*n* - 1) comparaisons. Les lignes 20-23 servent à calculer l'entropie de la classe *i*.

Le nombre maximal de calculs de l'algorithme est donné par le polynôme « *nmc + 13mc + 6c + 7m + 4* » qui revient à *O(nm)* si l'on ignore le nombre de classes *c* qui est normalement limité. Nous avons alors, dans le pire des cas, une complexité équivalente à celle du plus proche voisin. Toutefois, nous aurons une complexité beaucoup moins importante lorsque la meilleure partition est petite (dans ce cas, la taille du domaine de l'attribut correspondant est plus importante que 2), la meilleure partition contient plus de deux classes et les exemples sont distribués de façon plus équilibrée, ce qui correspond normalement à la plupart des situations.

5.5. Résultats expérimentaux

Pour évaluer la performance du système nous avons utilisé quelques bases de données du domaine public disponibles dans l'UCI Repository (Blake et Merz, 1998). Les bases utilisées sont les suivantes : Soybean, Zoo, Audiology, Tic-Tac-Toe et Mushroom.

5.5.1. Méthode de comparaison

Nous avons comparé les résultats à ceux présentés par Eklund (Eklund et Hoang) et Kasif et collaborateurs (1998). Pour comparer la performance avec celle de Kassif, la validation croisée de 10 interactions est employée (voir Tableau 9). La comparaison avec les résultats de Eklund a été accomplie avec une validation croisée de 5 interactions (voir Tableau 10). Le processus de validation croisée sert à mesurer la performance approximative d'un algorithme d'apprentissage. La performance est mesurée sur des exemples de test. Dans chaque itération *i* de *N*, la base de données (*BD*) est divisée en un ensemble pour l'apprentissage et un ensemble de test. Les exemples sont choisis aléatoirement. La taille de l'ensemble de test est (1/N)*|BD| et le reste des exemples est utilisé pour l'apprentissage. L'algorithme fabrique le classifieur avec les données d'apprentissage et sa performance est mesurée sur l'ensemble de test. À la fin des itérations, la performance moyenne et l'écart type sont calculés.

5.5.2. Les résultats

La comparaison avec les résultats de Kasif montrent que la performance d'ELA est similaire à celle de l'algorithme NN (Nearest-Neighbor). Ce comportement était attendu car l'algorithme de classification de la section 5.2.2 prend en compte le nombre d'attributs satisfaits dans le choix de la classe. Ainsi nous profitons de la précision du NN qui, selon Michie et al. (1994), surpasse la plupart des algorithmes de classification dans des situations normales. La section suivante donne une comparaison plus détaillée entre NN et ELA.

Algorithms	Tic-Tac-Toe	Soybean	Zoology	Mushroom
ELA	76,63 +-5,70	90,33 +-4,06	98,00 +-4,00	98,35 +-0,31
Bayes *	69,40 +- --	81,40 +- --	89,10 +- --	99,70 +- --
NN *	81,70 +- --	91,20 +- --	96,00 +- --	100,00 +- --
PEBLS *	94,40 +- --	93,20 +- --	95,00 +- --	100,00 +- --
-- Non fourni.				
* Résultats de (Kasif et al., 1998).				

Tableau 9Comparaison des Résultats.

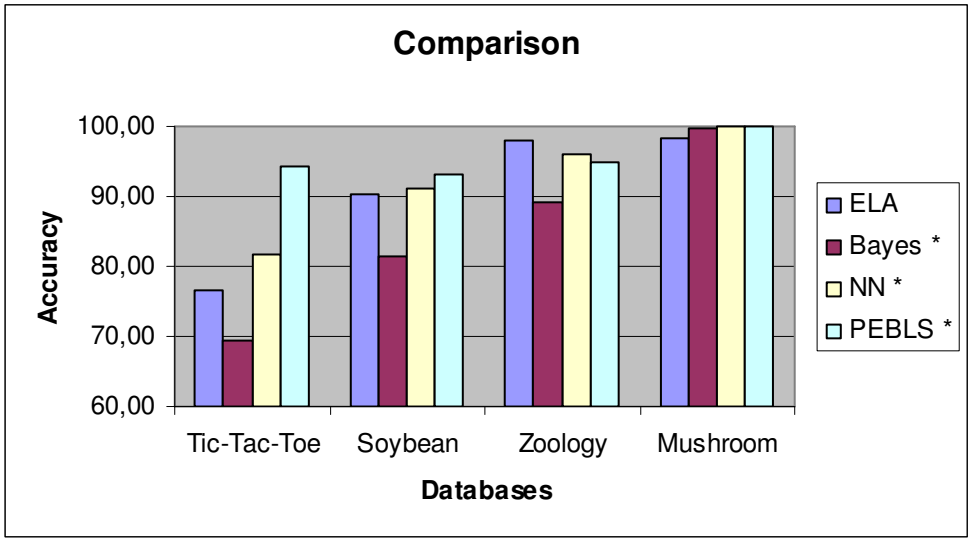

Figure 43 Représentation Graphique des résultats du Tableau 9 (* Résultats de (Kasif et al. 1998)).

Algorithms	Tic-Tac-Toe	Soybean	Zoology	Mushroom	Audiology
ELA	74,34 +-1,19	92,13 +-2,41	95,00 +-0,00	98,49 +- 0,24	78,00 +-4,84
C45 *	70,03 +-7,69	91,60 +-5,09	86,85 +-0,62	71,02 +-2,10	77,05 +-4,77
C45 Rules *	75,95 +-7,24	91,58 +-5,09	--	71,55 +-3,92	74,84 +-6,23
CN2 *	75,21 +-7,20	91,92 +-5,95	70,64 +-0,34	72,19 +-2,36	69,53 +-5,03
LMDT *	78,94 +-8,69	95,45 +-4,71	--	73,51 +-4,30	81,71+-13,28
-- Non fourni.					
* Résultats de (Eklund et Hoang).					

Tableau 10 Comparaison des Résultats.

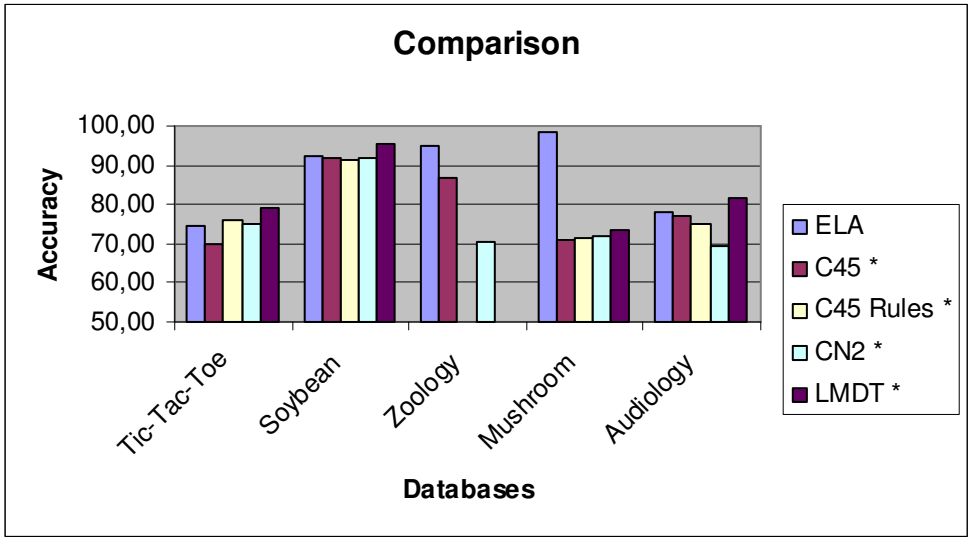

Figure 44 Représentation Graphique des résultats du Tableau 10 (* Résultats de (Eklund et Hoang)).

L'algorithme Naïve Bayes présente une performance très faible par rapport aux autres algorithmes. Les résultats sur la base TIC-TAE-TOE suggèrent que ELA est moins précis sur des bases de données où les attributs ont un pouvoir discriminant similaire. Étant donné que la division initiale dans le processus de classification a un effet direct sur le résultat de la classification. Ainsi une mauvaise répartition va diminuer la précision du système.

Les résultats du Tableau 10 sont très encourageants. Malgré une précision similaire, l'algorithme présente une très faible variation (écart-type). En ce qui concerne la base Mushroom, ELA obtient une précision bien meilleure avec une déviation beaucoup plus petite. Sur la base Audiology la performance du ELA est aussi très bonne. Cette base de données possède une distribution de classes très hétérogène car parmi les 200 exemples de la base, une seule classe possède 48 exemples et six classes possèdent seulement 1 exemple (le nombre total de classes est 24). Ces conditions rendent l'apprentissage très difficile. Un autre problème avec la base Audiology est la quantité d'attributs non pertinents. Parmi les 72 attributs présents dans la base, moins de 10 sont normalement utilisés par des arbres de décision. Cela suggère que le ELA est relativement robuste par rapport à ces problèmes.

5.5.3. ELA et NN

Comme nous avons vu précédemment, le ELA utilise indirectement le nombre de conditions satisfaites pour classer un exemple, ainsi comme NN. Dans cette section nous discutons les avantages de ELA sur NN.

Souvent, les algorithmes basés sur NN ont besoin d'un nombre K différent de voisins pour améliorer leur performance. Ainsi, dans les algorithmes comme K-NN l'estimation de K n'est pas une tâche simple. La stratégie la plus simple consiste à utiliser différentes valeurs de K sur des bases de données de validation et à choisir celle qui maximise la précision. Dans ELA ce problème n'existe pas.

Un autre avantage de ELA sur K-NN et d'autres algorithmes probabilistes est le temps de classification, étant donné que seulement une partie des données est utilisée dans la classification d'un nouvel exemple. Par ailleurs, le calcul des intersections entre les sous-ensembles est très rapide. K-NN et les algorithmes probabilistes sont généralement très lents car ils utilisent toute la base de données pour calculer les mesures de similarité et les probabilités, ce qui peut être problématique dans de grandes bases de données.

5.6. Réduction des données

L'un des principaux problèmes des techniques de MBR concerne la grande quantité de données nécessaires à conserver en mémoire. Par conséquent, ces algorithmes ont besoin de beaucoup d'espace pour stocker les données et, pire encore, demandent beaucoup de puissance pour le calcul des distances. Dans cette section nous allons étudier comment le graphe présenté dans la section 5.2.1 peut être utilisé pour réduire la taille de la base de données de façon significative sans trop dégrader la précision du système.

5.6.1. Quelques techniques connues

La manière la plus simple de réduire la taille de la base des données d'un algorithme MBR est de considérer que l'espace de recherche du domaine correspond à des périodes de temps limitées. Il suffit alors de garder en mémoire des données correspondantes à la période désirée. Par exemple, considérer toujours les données du mois dernier ou de la semaine dernière. Cette simplicité, évidemment a un coût. En effet, quand des données sont supprimées de la base, la précision du système peut diminuer ou augmenter. Si la précision augmente, cela veut dire que nous avons exclu des exemples *superflus* et du *bruit*. Des exemples superflus sont ceux qui normalement ne participent pas au processus de choix des classes. Le bruit est composé par des exemples qui conduisent le processus de choix de la classe à commettre des erreurs. Ils sont soit des erreurs, soit des aberrations. Si jamais la précision diminue, cela veut dire que des exemples pertinents ont été supprimés. Cela peut arriver facilement si nous prenons en compte seulement une période de temps pour réduire la

taille des données. Par conséquent, des chercheurs travaillent sur des méthodes capables d'identifier et d'éliminer les mauvais cas présents dans la base.

Certains algorithmes comme la famille IBL (Instance-Based Learning) de Aha (voir Aha et al., 1998) utilisent des heuristiques pour le stockage des données pertinentes. A chaque fois qu'un cas apparaît ses *K* (*K* doit être petit et impair) voisins les plus proches dans la base sont trouvés, si la classe plus fréquente parmi les voisins n'est pas la même qui celle de l'exemple nouveau, alors le nouveau cas est stocké, sinon (ils ont la même classe) il est tout simplement ignoré. En effet, ces algorithmes essayent de stocker seulement les exemples qui sont à la frontière des classes et ignorent les cas qui sont à l'intérieur des classes. Wilson et Martinez (2000) ont fait une étude complète sur le domaine et présentent dans leur article une comparaison entre 16 algorithmes différents de réduction de cas. Plus récemment Brighton & Mellish (2002) ont présenté un nouvel algorithme de sélection de cas nommé ICF (Iterative Case Filtering). Selon les auteurs, un cas *c* est *adaptable* au cas *c'* (dénoté *Adaptable(c,c'))* si *c* appartient à l'ensemble de cas qui sont à l'intérieur de la plus grande circonférence centrée sur *c'* et qui contient seulement des cas avec la même classe que *c'*. La *couverture* d'un cas *c* (dénotée *Coverage(c)*) est l'ensemble des cas dans la base (BC) auxquels *c* est adaptable ($\forall_{c' \in BC} Adaptable(c,c')$). L'accessibilité d'un cas c (dénotée *reachable(c)*) est l'ensemble des cas dans la base qui sont adaptables à c ($\forall_{c' \in BC} Adaptable(c',c)$). Un cas c doit être éliminé si $|reachable(c)| > |coverage(c)|$. Autrement dit, un cas doit être éliminé si le nombre de cas qui peuvent s'adapter à lui est plus important que le nombre des cas auxquels il peut s'adapter. Les auteurs appliquent cette règle itérativement sur la base de cas jusqu'à qu'il n'y ait plus de cas à éliminer. La Figure 45 illustre quatre itérations de l'algorithme sur une base de données quelconque.

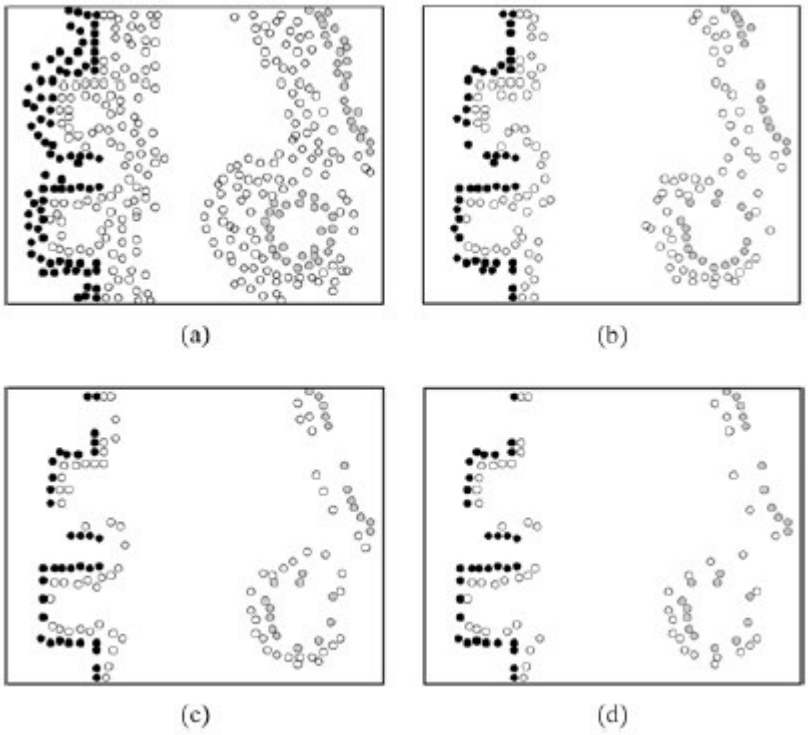

Figure 45 Processus d'élimination des cas[11].

5.6.2. Réduction des données avec ELA

Bien que la réduction des cas ne soit pas le sujet principal de cette thèse, nous avons quand même décidé d'évaluer la capacité du système à réduire la taille des données sans avoir trop de perte de précision. La technique employée est fondamentalement la même que celle employée dans les algorithmes IBL de Aha (Aha et al., 1991) : Quand un nouveau cas arrive, l'algorithme essaye de le classer. S'il y arrive, l'exemple est ignoré sinon (l'algorithme trouve une autre classe) il est stocké dans la base.

5.6.3. Méthode de comparaison

Nous avons récupéré les résultats présentés par (Wilson et Martinez, 2000). Les auteurs ont procédé à l'évaluation de 16 algorithmes MBR sur 30 bases disponibles dans la base des données de l'Université de la Californie Irvine (Blake et Merz, 1998). Parmi ces bases de données nous en trouvons deux que nous avons utilisées dans les tests de la section 5.5 : Audiology et Soybean. Nous avons aussi récupéré des données des essais accomplis par (Brighton & Mellish, 2002). Tous ces essais ont été faits avec une validation croisée de 10

[11] Extrait de (Brighton & Mellish 0)

interactions. Les résultats sont présentés dans le Tableau 11. L'écart type est dénoté par le signe « +- » .

Algorithms	Soybean	Espace	Zoology	Espace	Mushroom	Espace	Audiology	Espace
ELA	90,33 +-4,06	100%	98,00 +-4,00	100%	98,35 +-0,31	100%	76,63 +-5,70	100%
ELAr	80,33 +-6,70	23,63%	95,00 +-6,70	20,32%	96,50 +-3,71	0,43%	70,00 +-8,06	45,58%
Moyenne16*	84,81 +- --	38,31%	91,05 +- --	34,69%	--	--	--	--
RT3**	--	--	87,08 +- --	26,13%	98,89 +- --	5,50%	--	--
* Résultats de (Wilson et Martinez, 2000)			-- Non fourni					
** Résultats de (Brighton et Mellish, 2002)			ELAr : ELA avec réduction					

Tableau 11Comparaison des résultats avec réduction des données.

Dans le Tableau 11 nous comparons les algorithmes suivants : notre méthode (ELA) sans réduction des données (les mêmes résultats que dans Tableau 9) ajoutée de la base Audiology ; le graphe avec la méthode de réduction de données (ELAr) ; la performance moyenne des 16 algorithmes comparés par (Wilson et Martinez, 2000) ; le résultat de l'algorithme RT3 de (Wilson et Martinez, 2000) présenté par (Brighton et Mellish, 2002).

5.6.4. ELA, ELAr et RT3

Nous pouvons constater que malgré une réduction de 80% il n'a pas eu une dégradation de performance dramatique sur les bases Soybean et Zoology (10% pour Soybean et 3% pour Zoology). En ce qui concerne la base Mushroom le résultat est surprenant. D'après les environs 7313 cas utilisés dans l'apprentissage, l'algorithme a stocké seulement 33 cas (~ 0,43%). Malgré cela, la performance est pratiquement inchangée. Cette performance est largement supérieure à la performance du RT3 qui a besoin de stocker environ 402 cas. Pour la base Audiology les résultats sont aussi encourageants, car environ 50% des données ont été stockées pour l'apprentissage sans une grande perte de précision.

5.6.5. ELAr et Moyenne16

Selon le Tableau 11 nous constatons que la capacité de réduction de données de ELAr est largement supérieure à la moyenne des 16 algorithmes présentés par (Wilson et Martinez, 2000). Malgré une grosse différence dans le nombre de cas stockés la précision de classification est comparables.

5.6.6. Commentaires généraux

À partir des résultats obtenus avec la réduction de données, nous pouvons conclure que l'algorithme a un potentiel de réduction important. Nous n'avons pas fait d'essais très rigoureux, principalement parce qu'il est difficile de trouver des bases de données symboliques.

Nous croyons que les bons résultats son principalement dus à la propriété de la mesure d'entropie. C'est-à-dire, une fois que la distribution des classes ne change pas, la valeur de l'entropie ne change pas non plus. Par exemple, si nous avons les distributions de classes suivantes : (450 45) et (10 1) la valeur de l'entropie est la même. Ceci dit, nous pouvons représenter un problème avec une instance très réduite qui garde la même distribution des données.

5.7. Des travaux similaires

Dans cette section nous discutons quelques travaux qui présentent des approches similaires à celle que nous avons développée dans ce chapitre.

5.7.1. Construction de graphes

Kohavi (1994) propose une représentation nommée *oblivious read-once decision graphs* pour représenter des données. Le terme **read-once** correspond au fait que chaque attribut est présent une seule fois dans n'importe quel chemin du graphe. L'auteur appelle un **leveled graph** celui où les nœuds de même niveau contient des sous-ensembles disjoints d'exemples. Un **oblivious decision graph** est un **labeled graph** où chaque niveau du graphe correspond au même attribut. D'après ces observations nous pouvons dire que notre graphe (voir section Figure 38) est un *oblivious read-once decision graph*.

Quinlan (1986) (1993) propose de représenter des concepts extraits à partir des données dans des arbres de décision. Des arbres de décision ne sont pas des *oblivious graphs*, car à un même niveau il peut avoir des attributs différents. Les travaux de Quinlan et Kohavi ne concernent pas des algorithmes incrémentaux. D'autres tavaux plus récents comme (Utgoff, 1994) ont étendu la création d'arbres pour réaliser un apprentissage incrémental, mais ils sont complexes et ont besoin de beaucoup de temps pour construire les arbres. L'apprentissage incrémental à partir de règles (Shoenauer et Sebag, 1990) présente les mêmes problèmes.

La principale différence entre ces travaux et les nôtres concerne la façon dont le graphe est construit. Ces programmes branchés utilisent des critères différents de sélection d'attributs et des opérations de division et conjonction de nœuds pour produire des graphes réduits. Ensuite l'algorithme de classification est trivial. En revanche, dans notre travail, la phase la plus complexe n'est pas la création du graphe mais la phase de classification. Notre algorithme choisit un attribut une seule fois, pour trouver une partition initiale des données. Ensuite, à travers l'intersection de sous-ensembles, il est capable de trouver la classe correcte. Ainsi nous ne choisissons pas des attributs plusieurs fois comme dans les arbres de décision, ce qui rend l'algorithme simple et rapide.

Les techniques discutées dans les paragraphes précédents sont orientées vers un but, c'est-à-dire, les structures utilisées pour représenter les données sont connectées à un seul attribut but. Néanmoins, dans certaines applications, le système doit pouvoir prévoir la valeur de plusieurs variables. Par exemple, des assistants personnels peuvent avoir besoin de prévoir le **récepteur** d'un message, la **valeur** d'un service ou encore, **quand** envoyer un message. Dans ce cas, les techniques traditionnelles ont besoin de construire un modèle spécifique pour chaque variable. En revanche, le graphe présenté dans ce travail peut être utilisé pour la prédiction de n'importe quelle variable symbolique de la base de données.

5.7.2. L'usage du graphe

Friedman et collaborateurs (1996) ont développé une approche similaire à la nôtre en ce qui concerne la méthode de classification. Les données sont utilisées sans aucun pré-traitement et seul l'algorithme de classification est suffisant pour trouver la classe des exemples tests. Pour classer un exemple test l'algorithme fabrique un arbre de décision dynamique adaptée à l'exemple à partir des données brutes avec une procédure récursive en choisissant à chaque niveau de l'arbre un pair (attribut × valeur) et en répartissant les données. Néanmoins, beaucoup de calculs sont nécessaires pour classer un exemple étant donné que plusieurs paires (attribut × valeur) doivent être analysées à chaque niveau de l'arbre.

5.8. La problèmatique et la solution apportée

Nous avons introduit au début de ce mémoire la problématique concernant le manque de méthodes d'apprentissage adaptées à la conception des agents assistants personnels adaptatifs. Parmi les problèmes rencontrés nous avons souligné : la complexité, le temps d'apprentissage, et le mauvais usage des ressources.

Nous pensons que la méthode présentée dans ce chapitre peut résoudre ces problèmes. D'abord parce que la construction du graphe et l'algorithme de classification sont faciles à comprendre. Cela permet la mise en place de la méthode sans beaucoup d'effort et ne demande pas de connaissances du domaine d'application.

D'après nos essais, nous pouvons dire que notre algorithme n'a pas besoin de beaucoup de données pour apprendre, ce qui est très important pour un agent assistant personnel qui ne peut pas dépendre de nombreux feedbacks de l'utilisateur. Cette caractéristique réduit également les besoins de ressources nécessaires pour stocker des informations et pour réaliser des calculs.

5.9. Conclusions

Dans ce chapitre nous avons présenté une nouvelle technique incrémentale d'apprentissage. Nos agents ont besoin d'apprendre à partir de méthodes qui représentent des données de façon efficiente, qui classent des exemples rapidement et qui sont génériques (sans des paramètres comme les mesures de similarité, nombre de voisins, etc.). Pour accomplir cette tâche nous représentons les données sous forme d'un graphe qui découpe partiellement l'espace d'exemples. Le graphe est construit au fil de l'eau, à mesure que les exemples arrivent. Aucun traitement (découpage, sélection d'attributs, conjonction ou division de nœuds) n'est fait sur le graphe. Pour trouver la classe d'un exemple, l'algorithme choisit une partition initiale des données et essaye de trouver la classe capable de satisfaire le plus grand nombre de conditions.

Les résultats de notre approche, comparés à d'autres travaux, sont très encourageants. Nous avons procédé à plusieurs essais et nous avons pu constater que l'algorithme présente une bonne performance et, de plus, peut réduire énormément la quantité des données nécessaires pour l'apprentissage dans certaines bases de données.

6 Le système MAIS

Dans ce chapitre nous allons présenter le système MAIS (Multi-Agent-Based Internet Search). Le système MAIS a été conçu pour améliorer l'efficacité de la recherche de documents sur le web. Nous avons décomposé le processus de recherche en le distribuant sur plusieurs agents spécialisés dans des tâches spécifiques. Certains d'entre eux fabriquent dynamiquement le modèle des préférences de l'utilisateur en utilisant la méthode ELA présentée au chapitre précédent pour apprendre un tel modèle, puis pour raffiner les résultats de la recherche.

Le modèle de l'utilisateur est composé de centres d'intérêts divers et de pages web favorites. La combinaison de ces structures aboutit à un modèle générique de préférences qui sert à ordonner la liste de sites à présenter à l'utilisateur. Cette combinaison est faite grâce à la création d'un Graphe de Concepts. Ce mécanisme permet de personnaliser les résultats de la recherche. Ainsi, une même requête aboutira à des résultats différents selon les utilisateurs.

La recherche personnalisée d'information sur le web est une application classique d'agent assistant personnel. Dans notre approche, nous favorisons la distribution des fonctionnalités de l'assistant à plusieurs agents appartenant à un staff pour simplifier la structure de l'assistant, composée dans notre cas, seulement d'une interface et d'un ensemble de fonctions LISP. Nous favorisons également la personnalisation des services et non la personnalisation de l'interaction pour les raisons déjà discutées dans le Chapitre III.

Après avoir détaillé le système nous faisons quelques remarques concernant les expériences faites dans le laboratoire pour évaluer l'efficacité du système.

6.1. Problématique

La plupart des utilisateurs qui cherchent des documents sur le web utilisent plusieurs moteurs de recherche. Savoy (2002) soutient que 85% des utilisateurs utilisent des moteurs de recherche différents pour trouver des pages web. Cela met très nettement en évidence les limites des moteurs de recherche actuels. Les différents résultats des moteurs de recherche sont provoqués par deux problèmes principaux : (i) les moteurs de recherche possèdent différentes bases de pages indexées et (ii) les moteurs de recherche emploient différentes mesures de qualité des pages résultantes en fonction de la requête soumise. Dans ce travail nous nous intéressons à ce deuxième problème.

La qualité des sites peut être calculée de différentes manières selon les moteurs de recherche. Certains utilisent, par exemple, la fréquence des termes de la requête et d'autres utilisent des techniques heuristiques. Une technique bien connue est celle utilisée par Google. Elle consiste à mesurer la qualité d'une page en prenant en compte le nombre de liens existants vers cette page. Cette technique est connue comme *Page Rank* (Grabowicz, 2002). L'heuristique dans ce cas est fondée sur la forte relation sémantique existant entre les liens et la page en question.

Ces techniques peuvent marcher pour certains cas, mais elles ne sont pas suffisantes dans la majorité des cas. Cela est dû principalement au fait que les préférences et les connaissances des utilisateurs ne sont pas prises en compte, et donc, que les résultats ne sont pas personnalisés.

La personnalisation pour le web n'est pas un sujet nouveau, mais elle continue à stimuler la création de techniques nouvelles. Les premières idées de personnalisation pour le web ont été conçues au début des années 90 par Lieberman (1997), Armstrong et collaborateurs (1995), Chen et Sycara (1998) et Lashkari et collaborateurs (1994). Les trois premiers ont fabriqué des agents d'aide à la navigation sur le web et le dernier a conçu un système d'agents capable de gérer des courriers.

Dans ce chapitre nous avons décidé d'attaquer le problème de la recherche personnalisée d'information sur le web, car malgré de nombreux efforts, la solution à ce problème reste encore insuffisante. Deuxièmement, dans cette tâche, le comportement adaptatif du système semble fondamental. Cet aspect doit être pris en compte dans l'adaptation du modèle de l'utilisateur car le système doit s'adapter aux changements et aux évolutions concernant les préférences et centres d'intérêts de ce dernier (le maître). Nous développons cet aspect avec la

méthode d'apprentissage discutée dans le chapitre précédent. Les sections suivantes donnent plus de détails sur le système que nous avons réalisé.

6.2. Les agents adaptatifs et le web

Les agents sont souvent mis en œuvre dans l'élaboration de services Internet intelligents. Ces services aident les utilisateurs à personnaliser des journaux numériques, trouver des articles, des pages et des experts ou encore filtrer des informations. Depuis la deuxième moitié des années 90 une énorme vague de travaux utilisant des agents spécialistes a déferlé. Une caractéristique commune à tous ces agents est la présence d'un modèle suffisamment précis de l'utilisateur qui sert à guider le processus de solution ou à améliorer les résultats. Dans le rapport de Middleton (2001), environ soixante outils différents sont mentionnés. L'auteur a classé ces outils selon les différentes manières de concevoir le modèle de l'utilisateur :

- Superviser le comportement de l'utilisateur : ces agents construisent des bases de données non-classées qui représentent les actions de l'utilisateur, ensuite une méthode d'apprentissage (par exemple, clustering ou réseau de comportement) est utilisée pour identifier les séquences d'actions à prendre en compte dans une situation donnée ;

- Utiliser le feed-back explicite de l'utilisateur : les informations fournies par l'utilisateur sont précieuses car elles aident l'agent à corriger de possibles erreurs et à s'approcher plus vite de la solution du problème; Lieberman appelle cette méthode « Profiling by Example » (Lieberman, 2001) ;

- Utiliser un algorithme d'apprentissage sur un ensemble d'apprentissage : l'apprentissage peut être incrémental ou non (comme discuté dans le chapitre III de la thèse). Tandis qui l'apprentissage de comportement est normalement incrémental, d'autres formes d'apprentissage comme le planning ou les arbres de décision peuvent être utilisées dans la construction du modèle du maître. Dans ce cas, l'apprentissage est non-incrémental ;

- Laisser l'utilisateur programmer l'agent : l'utilisateur peut lui-même utiliser un langage de programmation pour améliorer l'efficacité de l'agent. Dans ce cas, des techniques comme la programmation par icônes ou la programmation graphique peuvent aider à réduire le travail de l'utilisateur ;

- Utiliser une base de connaissances : Les stéréotypes de l'utilisateur peuvent être définis au préalable sous forme d'une base de connaissances. Des méthodes classiques permettent à l'utilisateur de remplir des formulaires dont les réponses sont mises en œuvre sous forme de règles.

Dans le problème de recherche sur l'Internet nous avons choisi d'utiliser la première approche. Le modèle de l'utilisateur est mis à jour dynamiquement à partir de ses actions. Dans notre cas, les actions se limitent à ajouter un nouveau site dans les favoris et/ou changer les centres d'intérêts. Comme les sites ajoutés par l'utilisateur sont souvent ceux que le système lui a présentés comme résultats de la recherche, cela constitue aussi une façon d'utiliser le feed-back de l'utilisateur.

6.3. Les systèmes multi-agents et le web

Dans cette section nous mettons en évidence l'intérêt d'utiliser un SMA pour une application pour l'Internet. Parmi les avantages : l'extensibilité, la stabilité et l'équilibrage de charges.

- *Extensibilité*: le système peut être mis à jour et étendu sans effets collatéraux quand il est en l'opération. Étant donné que les agents sont indépendants, ils peuvent être créés et mis à jour tandis que d'autres agents continuent à travailler et à fournir des services ;

- *Stabilité*: quand un agent tombe en panne, d'autres agents peuvent coordonner la distribution des services et assurer la continuité du système ;

- *l'équilibrage de charges* : les agents distribuent naturellement la charge de travail. Certaines tâches comme l'apprentissage des profils des utilisateurs ou la recherche d'information sur le web peuvent demander beaucoup de calculs. Des protocoles de coordination d'agents distribuent uniformément la charge de travail entre les agents qui peuvent évoluer dans différents processus, threads, machines ou réseaux.

Malgré le potentiel évident des SMA, les applications des SMA l'Internet ne sont pas très courantes. Cela est dû principalement au manque de mécanismes standards qui permettraient l'échange d'information et l'interaction entre différentes plates-formes d'agents. Cependant, à ce jour, un effort important est réalisé par des groupes de recherche qui visent à établir des normes ou des standards pour le développement des SMA, notamment celui de FIPA (FIPA).

Depuis un certain temps, un consensus général semble s'affirmer. Ce consensus établit que des agents intelligents vont guider tout utilisateur connecté au web pour réaliser une tâche donnée. Pour cela, évidemment, des mécanismes standards qui permettent aux agents de dialoguer semblent indispensables. Ces idées sont reprises aujourd'hui par au moins deux applications développées sur le web: Agentcities Task Force (Agentcities) et Trade Agent Competition — TAC (TAC).

Agentcities Task Force (Agentcities) regroupe plus d'une cinquantaine de plates-formes d'agents FIPA distribuées dans le monde. À ce jour, ces plates-formes échangent seulement des services très simples qui visent à assurer la conformité aux normes FIPA mais dans un futur proche elles pourront fournir des services beaucoup plus complexes.

TAC est une compétition publique entre plusieurs agents commerciaux qui jouent le rôle d'un agent de voyage qui doit satisfaire les besoins de ses clients. Chaque compétition est faite entre huit agents. Chaque agent doit mettre en œuvre des stratégies commerciales pour maximiser les ressources de l'utilisateur. Les ressources doivent permettre au client d'organiser un voyage en prenant en compte l'achat des billets d'avion, la réservation de l'hôtel et, sur option, des divertissements. Ainsi, TAC montre bien le type de services que les agents pourront fournir aux utilisateurs connectés à l'Internet.

6.4. Pourquoi des systèmes multi-agents dans la recherche d'information

Nous avons décidé de concevoir un SMA pour la recherche sur web car ce problème peut naturellement être décomposé en des problèmes plus petits.

D'abord, la recherche web doit prendre en compte le plus grand espace de pages possible pour assurer que toutes les pages potentiellement intéressantes sont consultées. Nous avons donc attaqué cette première question avec des agents jouant le rôle de meta-chercheurs capables de ramasser des sites à partir de plusieurs moteurs de recherche connus comme Google, Altavista ou Lycos.

Deuxièmement, le système doit être capable de modéliser les préférences de l'utilisateur pour personnaliser la présentation et les résultats d'une recherche. Ainsi, des personnes qui ont différents centres d'intérêts pourront recevoir des résultats différents pour la même requête.

Troisièmement, nous avons besoin d'un mécanisme pour coordonner ces activités. Ces activités nécessitant un système qui, s'il est centralisé et capable d'accomplir toutes ces tâches, doit être très complexe. Par ailleurs, l'apprentissage des préférences de l'utilisateur et l'analyse des pages web demandent en général beaucoup de calculs et doivent donc être repartis dans des machines et/ou des agents différents. Par conséquent, un SMA peut être utilisé pour distribuer les calculs entre plusieurs agents spécialisés dans des tâches plus simples. Nous avons utilisé ces idées dans la conception de l'architecture du système MAIS discuté dans les sections suivantes.

6.5. Architecture MAIS

Notre système de recherche d'information sur la toile contient quatre catégories d'agents : l'Assistant Personnel (AP), l'Agent Bibliothèque (AB), les Agents de Filtrage (AF) et les Agents de Recherche (AR), comme schématisé sur la Figure 46.

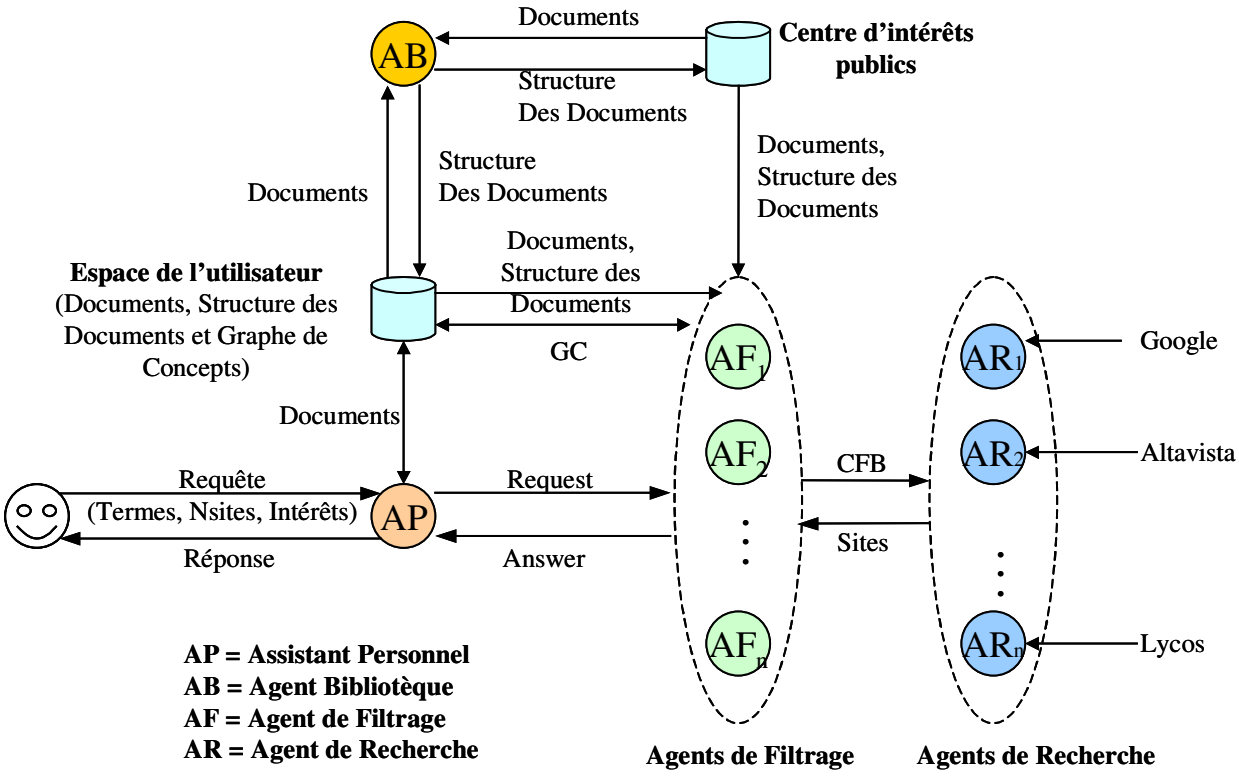

Figure 46 L'architecture du système MAIS.

Tout d'abord, l'utilisateur interagit toujours à travers son agent assistant personnel. L'agent assistant personnel fournit fondamentalement deux services : la gestion des sites favoris et, évidemment, la recherche de sites.

6.5.1. La gestion des sites favoris

Les sites favoris sont stockés par l'agent assistant personnel dans l'espace de travail de l'utilisateur dans des dossiers personnels sous forme de documents. Chaque document correspond au contenu textuel d'un site. Ces documents sont ensuite utilisés par l'agent bibliothèque (AB) dans la création d'une structure unique et générale. Cette structure est composée par la liste des termes les plus importants pour la collection de documents personnels. Aucun agent n'interagit directement avec l'agent bibliothèque. En effet, l'agent bibliothèque peut être programmé pour mettre à jour la structure des documents de l'utilisateur automatiquement (une fois par jour, par exemple) ou recevoir une requête de l'administrateur du système. L'agent bibliothèque est également utilisé par l'administrateur du système dans la construction de centres d'intérêts publics.

Les centres d'intérêts publics fournissent aux utilisateurs des modèles acquis à partir de documents de domaines bien précis. Chaque centre d'intérêts est composé par une collection de documents du domaine et la structure de ces documents acquise avec l'agent bibliothèque. Artificial-Intelligence-Adaptative-Agents, Groupware-Knowledge-Management et Artificial-Intelligence-MBR sont des exemples de centres d'intérêts existants dans le système. L'utilisateur peut donc choisir ceux qui lui conviennent. Les documents de base utilisés dans la construction de chaque centre d'intérêts sont des articles demandés aux experts du labo. Ces centres d'intérêts seront ensuite utiles pour filtrer et organiser le résultat des recherches.

6.5.2. Le processus de recherche

L'utilisateur déclenche une recherche en passant à l'agent assistant personnel une liste de termes recherchés, le nombre de sites voulus comme résultat et les centres d'intérêts qui lui conviennent. L'assistant renvoie ensuite la requête en broadcast. Parmi les agents de filtrage (AF) celui qui est disponible sera responsable de la réponse. Si jamais plus d'un agent est disponible, le contrat sera attribué à celui qui réagit le premier.

Les agents de filtrage jouent un rôle central dans le système. La première tâche d'un agent de filtrage engagé dans un processus de recherche est de ramasser des sites auprès des agents de recherche (AR) du système. Il envoie donc un call-for-bids en passant la liste même des termes reçus dans le contrat et attend un certain temps l'arrivée de toutes les réponses des agents de recherche. Le temps d'attente est paramétrable (à ce jour il est de 20 secondes). À ce point, l'agent de filtrage récupère une liste énorme de sites qu'il doit réduire et ordonner

pour finalement présenter les résultats à l'utilisateur. C'est à ce moment que le mécanisme de personnalisation entre en action.

Le processus de personnalisation consiste à ordonner la liste de sites reçus des ARs selon les préférences de l'utilisateur. En effet, chaque page est soumise à un processus de classification qui rend comme réponse la classe du site et le degré de qualité de la classification. Les classes possibles sont les noms des dossiers créés par l'utilisateur pour organiser les favoris et les centres d'intérêts qu'il a choisis. Finalement après avoir mis les sites sur une queue de priorité les meilleurs sites sont séléctionés. Les sections suivantes donnent plus de détails sur comment chaque agent a été conçu, notamment sur le mécanisme de classification de l'agent filtrage.

6.5.3. L'agent assistant personnel (AP)

Dans notre application l'agent assistant personnel est composé de deux éléments principaux : l'interface et une base d'actions. L'interface permet à l'utilisateur de déclencher des recherches, de visualiser les résultats et de gérer les favoris. Elle constitue le côté client de l'agent assistant personnel. La base d'actions est composée par une série de fonctions LISP codées dans un serveur HTTP qui sont déclenchées à distance par l'interface. Par conséquent, n'importe quel utilisateur connecté à l'Internet peut utiliser l'interface LISP pour faire des recherches. La Figure 47 montre l'interface de l'agent assistant personnel. L'interface permet aussi à l'utilisateur de choisir les centres d'intérêts qui lui conviennent (voir la Figure 48).

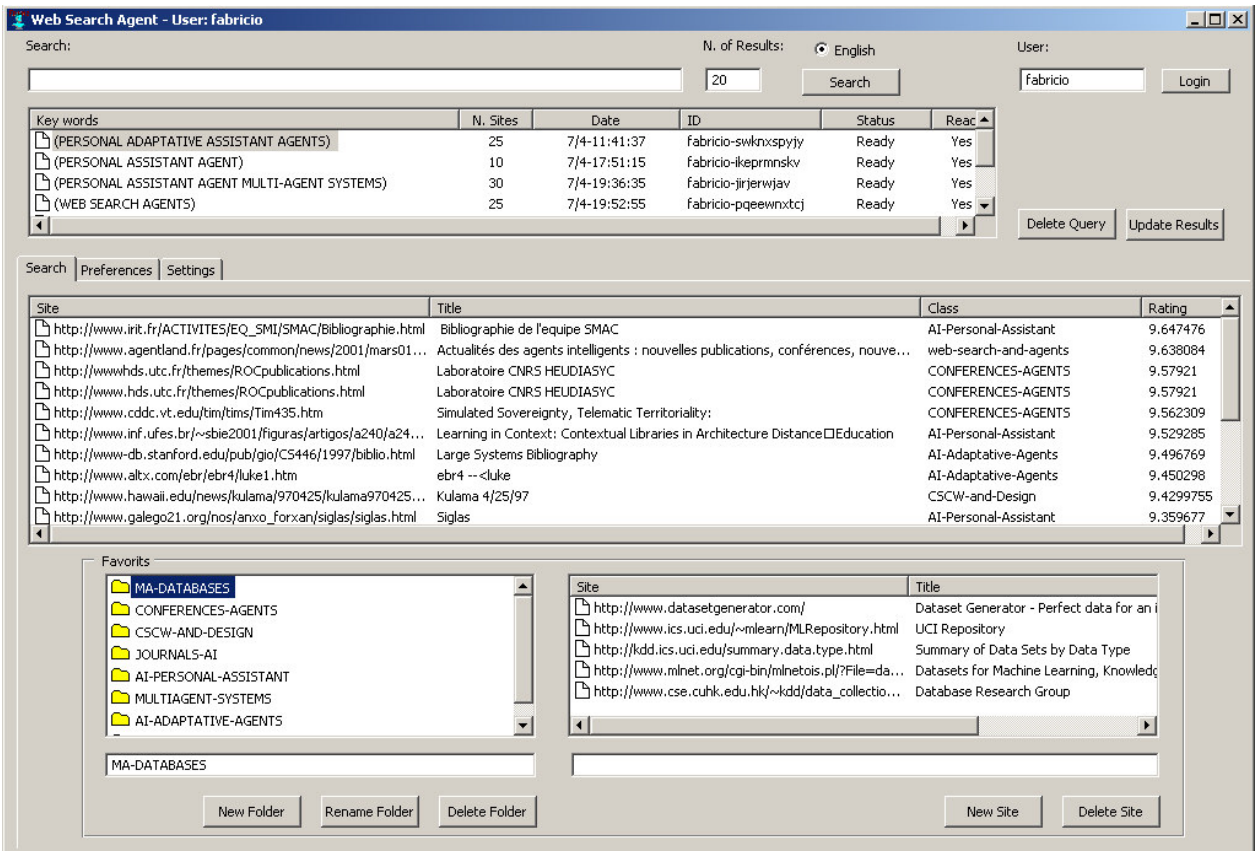

Figure 47 L'interface de l'agent assistant personnel.

Étant donné que le processus de recherche peut durer plusieurs minutes, les recherches sont déclenchées de façon asynchrone. L'utilisateur peut donc activer plusieurs recherches pendant la nuit, par exemple et analyser les résultats le lendemain.

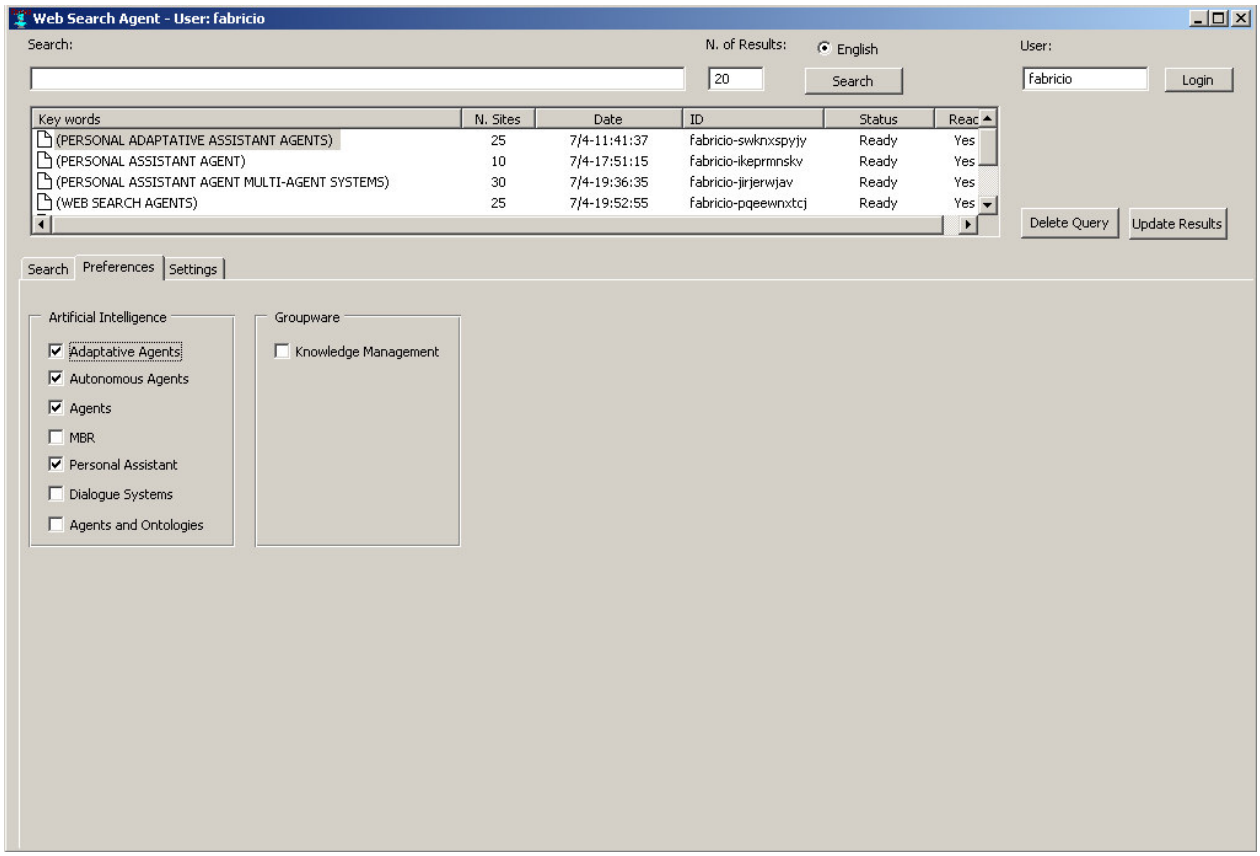

Figure 48 Les centres d'intérêts publics.

Pour faciliter les échanges entre l'interface et le serveur HTTP nous avons développé un petit langage de contenu, le langage MAIS-L. Il est également utile pour la communication entre les agents internes à notre plate-forme OMAS (Barthès, 2002) et agents externes développés dans d'autres plates-formes compatibles avec FIPA (FIPA), comme par exemple, JADE (JADE). Voici la représentation syntactique du langage :

---

command ::= login-command
| verify-command
| give-query-command
| remove-query-command
| update-proxy-command
| site-addition-command
| fold-addition-command
| rename-fold-command
| site-deletion-command
| fold-deletion-command
| query-command

login-command ::= "(" "USER-LOGIN" User-Name ")"

verify-command ::= "(" "VERIFY-IF-RESULT" User-Name Query-ID ")"

give-query-command- ::= "(" "GIVE-QUERY-RESULT" User-Name Query-ID ")"

remove-query-command ::= “(” “REMOVE-QUERY-FROM-PROFILE” User-Name Query-ID “)”

update-proxy-command ::= “(” “UPDATE-PROXY” Server Port User-Name “)”

site-addition-command ::= “(” “ADD-SITE-ON-PROFILE” Site-Item Fold-Name User-Name “)”

fold-addition-command ::= “(” “ADD-FOLD-ON-PROFILE” Fold-Name User-Name “)”

rename-fold-command ::= “(” “RENAME-FOLD-ON-PROFILE” New-Fold-Name Fold-Name User-Name “)”

site-deletion-command ::= “(” “REMOVE-SITE-FROM-PROFILE” Fold-Name Site User-Name “)”

fold-deletion-command ::= “(” “REMOVE-FOLD-FROM-PROFILE” Fold-name User-Name “)”

query-command ::= “(” “START-REQUEST” User-Name Words Number-Sites Domains Date Query-ID Read “)”

Query-ID ::= string

Words ::= list of terms

Domains ::= nil | list of terms

User-Name ::= string

Server ::= url

Port ::= number

Fold-Name = string

New-Fold-Name = Fold-Name

Site ::= url

Site-Item ::= “(” Site Site-Title“)”

Site-Title ::= string

User-Profile ::= “(” :DOMAINS Domains :HTTP-SERVER Server :HTTP-PORT Port :FOLDERS Fold-List :id Search-List “)”

Fold-List ::= list of Fold-Item

Fold-Item := “(” Fold-Name Site-List “)”

Site-List ::= list of Site-Item

Site-Item ::= “(” Site Site-Title “)”

Site-Title ::= string

Search-List ::= list of Search-Item

Search-item ::= “(” Search-ID Words Number-Sites Date Read “)”

Date ::= 10 digits number

Read ::= “Yes” | “No”

Query-Answer ::= list of Answer-Item

Answer-Item ::= “(” “(” Site Site-Title Fold-Name “)” Relevance “)”

Relevance ::= Number

L’Annexe IV donne une définition sémantique de chaque commande disponible dans le langage. La Figure 49 montre comment une recherche peut être déclenchée à partir de l’environnement JADE. À gauche nous avons les informations sur notre plate-forme et à droite le format de message à envoyer à l’agent « searcher@OMAS » qui va traiter la demande. « searcher@OMAS » joue le rôle d’assistant et va démarrer le processus de dialogue entre les agents comme discuté dans la section précédente.

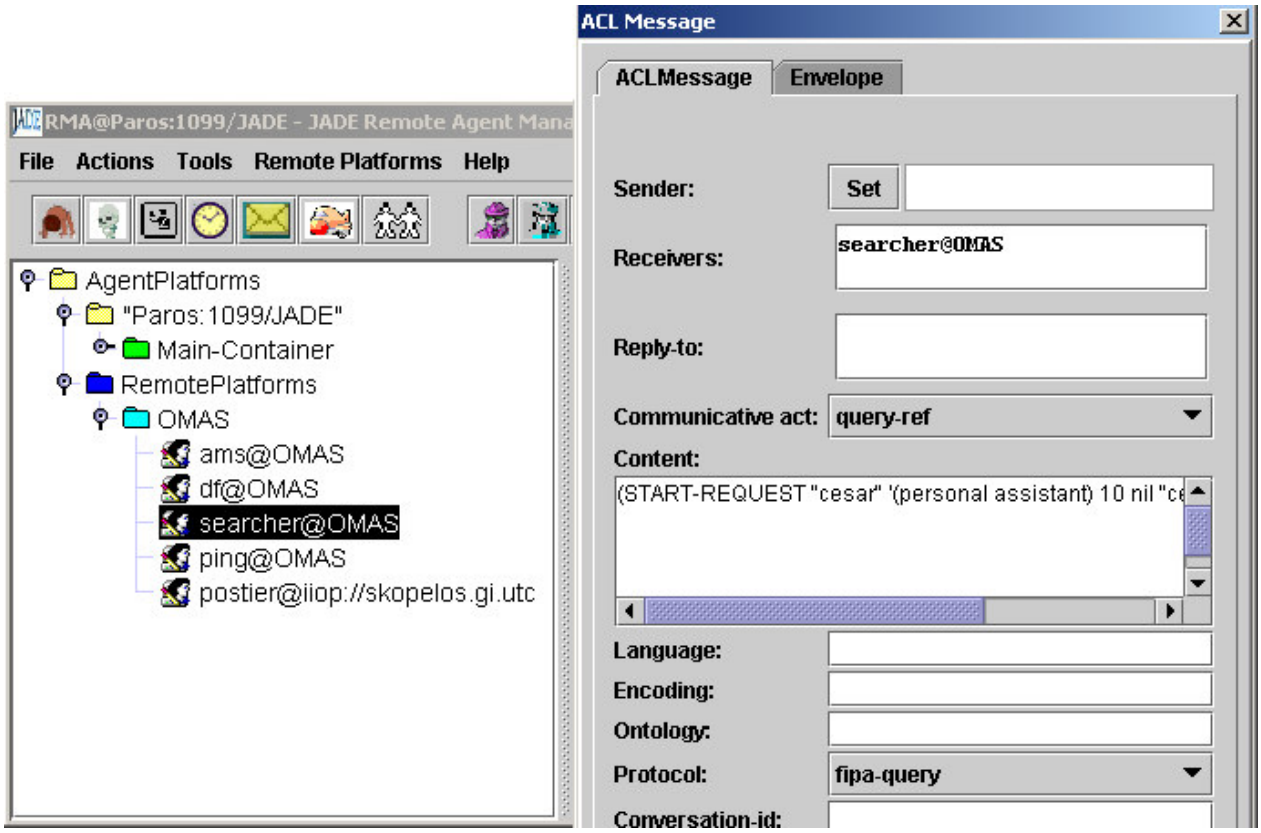

Figure 49 L'exécution de services avec des agents JADE.

6.5.4. L'agent bibliothèque (AB)

L'AB est spécialisé dans la manipulation de documents. Son rôle est de concevoir à partir d'une collection de documents, une structure unique capable de représenter tous les documents. Pour cela il met en œuvre certaines techniques dérivées de différents domaines comme *la classification automatique de documents* ou encore *la récupération d'information*.

Normalement, la classification de documents est divisée en deux phases : l'*apprentissage* et la *classification*. Dans la phase d'apprentissage, une collection de documents déjà classés est utilisée pour la création d'un ensemble d'apprentissage (training data). Pour transformer les documents en données, un certain nombre de caractéristiques sont extraites à partir des documents et chaque document est représenté sous forme d'un vecteur de caractéristiques. Ensuite, un algorithme d'apprentissage quelconque va fabriquer un modèle conceptuel à partir des données. Dans la phase de classification le classifieur fabriqué lors de l'apprentissage est mis en route pour classer les nouveaux documents.

```
(7
 (("AGENT" 265 6 6) ("ACTION" 207 7 7) ("SUBSTITUTION" 44 1 1)
   ("EVENT" 91 4 4) ("SYSTEM" 164 7 7) ("GOAL" 145 7 6) ("ROBOT" 79 4 4)
   ("NODE" 84 5 5) ("CONDITION" 97 6 6) ("CIRCUITRY" 30 1 1)
   ("GEORGEFF" 76 5 5) ("BLOCK" 29 1 1) ("INTENTION" 59 4 3)
   ("MODEL" 104 7 7) ("WORLD" 95 7 7) ("CONTROL" 87 7 7)
   ("ENVIRONMENT" 84 7 7) ("SUBSYMBOLIC" 22 1 1) ("FIGURE" 56 5 5)
   ("PLANNER" 37 3 3) ("ARCHITECTURE" 82 7 7) ("TREE" 81 7 5)
   ("AIRCRAFT" 28 2 2) ("ACHIEVE" 64 6 6) ("FIRST" 78 7 7)
   ("EXECUTION" 76 7 7) ("SYMBOLIC" 34 3 3) ("SEQUENCE" 60 6 6)
   ("EXAMPLE" 73 7 7) ("NILSSON" 49 5 4) ("EXECUTINGPLAN" 19 1 1)
   ("PREIMAGE" 19 1 1) ("TRUE" 59 6 5) ("OPTIONAL" 18 1 1)
   ("SYMBOL" 24 2 2) ("TELEOREACTIVE" 24 2 2) ("BENSON" 24 2 2)
   ("PROBLEM" 53 6 6) ("EXECUTINGBRANCH" 17 1 1) ("FUNCTION" 64 7 7)
   ("SPECIFICATION" 35 4 4) ("SIDE" 28 3 3) ("POSSIBLE" 62 7 7)
   ("COMMITMENT" 21 2 2) ("DIFFERENT" 58 7 7) ("GENETIC" 15 1 1)
   ("REGRESSION" 20 2 2) ("INTELLIGENCE" 56 7 7) ("ARTIFICIAL" 55 7 7)
   ("CONCEPT" 24 3 3) ("COMMUNICATION" 14 1 1) ("CURRENT" 52 7 7)
   ("FORMULA" 23 3 3) ("MOTOR" 13 1 1) ("THEOREM" 17 2 2)
   ("INTERNAL" 39 6 6) ("BELIEF" 26 4 4) ("MENTAL" 21 3 3)
   ("BRATMAN" 21 3 3) ("LIBRARY" 21 3 3) ("BOTWORLD" 21 3 2)...))
```

Figure 50 Exemple d'une liste de termes choisis à partir du domaine « Agents Autonomes » avec la mesure TF-IDF.

La tâche de l'agent bibliothèque consiste à trouver la bonne représentation pour l'ensemble de documents en choisissant les bonnes caractéristiques. Une technique bien connue dérivée des recherches en *récupération d'information* consiste à choisir certains termes existants dans les documents pour représenter les vecteurs de documents. Ces termes sont normalement les termes les plus importants pour la collection de documents. Pour choisir les termes nous avons donc utilisé une mesure de qualité très connue, TF-IDF (Term Frequency - Inverse Document Frequency) (Salton, 1989) présentée dans la section 4.3.3. La Figure 50 montre une liste de termes extraits par l'AB à partir des documents du domaine « Agents Autonomes ». Le premier élément de la liste est le nombre de documents et le deuxième est la liste de termes utilisés par l'algorithme d'apprentissage par la suite.

Ainsi la représentation de la collection de documents de chaque utilisateur et de chaque domaine est structurée selon les termes trouvés par l'agent bibliothèque. La section suivante montre comment ces structures sont mises en œuvre dans le processus d'apprentissage.

6.5.5. L'agent de filtrage (AF)

L'AF coordonne le dialogue entre les différents agents du système. L'AF reçoit comme paramètre la liste de termes à rechercher, le nombre de sites à rendre comme réponse et les noms des centres d'intérêts donnés par l'utilisateur. Il est responsable de la personnalisation

du système et son rôle principal est d'ordonner et de réduire la liste de sites qu'il reçoit des ARs en fonction du profil de l'utilisateur. Pour cela, il utilise la méthode d'apprentissage ELA présentée au chapitre précédent.

Dans notre approche la qualité d'un site est calculée selon le taux de classification donné par l'algorithme de classification. Pour créer le graphe de concepts et ensuite l'utiliser pour classer les sites, deux questions doivent être résolues. Premièrement, *les données d'un algorithme d'apprentissage doivent être structurées* (attributs) et, deuxièmement, *nous devons disposer d'exemples d'apprentissage*.

La structure générale des documents

Nous avons vu dans la section précédente comment l'agent bibliothèque est capable de construire une représentation des documents de chaque utilisateur et de chaque centre d'intérêts public. La représentation des documents de l'utilisateur et de ses centres d'intérêts doivent être combinés pour la construction d'une structure unique de document qui sera ensuite utilisée par l'algorithme d'apprentissage. La combinaison entre ces différentes structures se fait avec un processus de « fusion » entre les structures où le critère d'ordonnancement des termes est la valeur TF-IDF. Cette liste est limitée à 1000 éléments. Avec cette méthode nous assurons que des informations de toutes les classes seront utilisées par l'algorithme d'apprentissage. Également, nous prenons en compte un nombre raisonnable d'attributs (1000). La Figure 51 donne un exemple de la procédure adoptée.

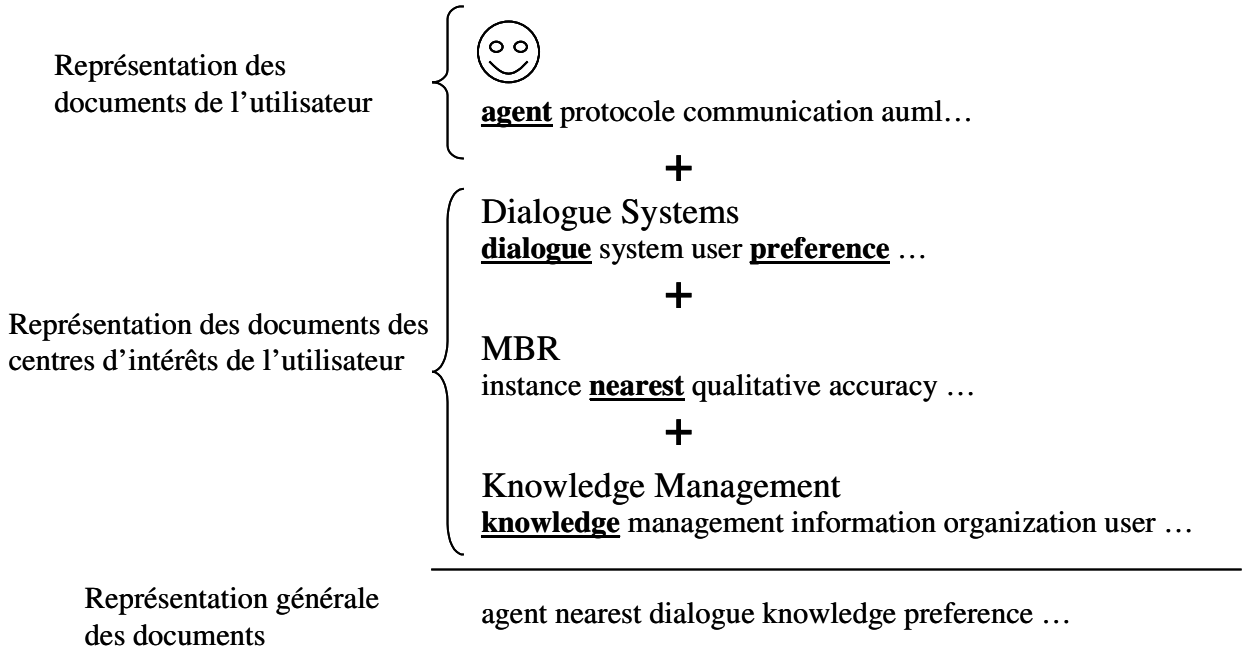

Figure 51 Combinaison des structures de documents dans une structure générale.

Les exemples d'apprentissage

Ayant résolu le problème de la structuration des documents, la deuxième question consiste à trouver suffisamment de données pour l'algorithme d'apprentissage. Pour fabriquer le graphe de concepts, l'agent filtrage utilise comme données les documents existants dans les favoris de l'utilisateur, plus les documents qui constituent chaque centre d'intérêts saisi par l'utilisateur. Ces documents sont représentés comme des vecteurs où les attributs sont les termes trouvés selon la procédure présentée dans la section précédente. Il faut souligner que la classe de chaque document est connue. Il s'agit du nom du dossier favori où l'utilisateur a enregistré le document (dans le cas des documents favoris) ou du nom du centre d'intérêts concerné (dans le cas des documents des centres d'intérêts).

Parce qu'à ce jour notre l'algorithme d'apprentissage ne travaille qu'avec des données symboliques, le remplissage de chaque vecteur prend en compte seulement l'existence ou non de chaque terme représenté par un symbole composé par le nom du terme plus les chaînes de caractères « -t » ou « -f » respectivement. Voici une représentation possible d'un document si nous reprenons l'exemple de la Figure 51 :

(“ agent-t” “nearest-f” “dialogue-t” “knowledge-t” “preference-f” …)

Ainsi, lorsque l'agent filtrage reçoit une requête il recherche le graphe de concepts dans l'espace de l'utilisateur. Si jamais le graphe n'est pas encore créé, il le crée. Ensuite il classe

tous les sites reçus des agents de recherche et enregistre la réponse. Sur la Figure 47 nous pouvons distinguer les classes trouvées par l'agent filtrage et le taux de classification. Le taux de classification est celui calculé par la méthode ELA lors de la classification du document (voir Figure 5 du chapitre IV).

L'aspect adaptatif de l'agent

Dans cette thèse nous nous sommes intéressé aux méthodes qui permettent aux agents de s'adapter au profil de l'utilisateur. Dans l'application de recherche documentaire sur web, l'adaptation est prise en main par les agents filtrage. Dans ce cas, le graphe de concepts constitue le modèle général des préférences de l'utilisateur.

L'adaptation du modèle de préférences de l'utilisateur se fait dans deux cas :

- L'utilisateur ajoute/supprime un document dans les favoris ;
- L'utilisateur demande une recherche avec des centres d'intérêts différents de ceux présents dans son profil.

Dans le premier cas l'aspect incrémental du graphe permet à l'agent d'ajouter et de supprimer un nouveau document quasiment sans aucun effort. Cela veut dire que l'utilisateur peut ajouter/supprimer successivement des documents et le modèle de son profil est mis à jour dynamiquement. Par conséquent, les recherches déclenchées prennent toujours en compte le bon modèle pour filtrer les résultats, car l'algorithme apprend instantanément. Dans le cas de la suppression d'un document, il suffit de parcourir les liens du graphe et de supprimer l'identifiant du document concerné. Si jamais un nœud devient vide, il est aussi supprimé.

Dans la deuxième situation, le graphe est reconstruit à partir de zéro. Chaque centre d'intérêt ajouté/supprimé correspond à plusieurs documents qu'il faut ajouter ou supprimer du graphe. La procédure la plus simple consiste à recréer le graphe. Le calcul nécessaire dépend évidemment du nombre de centres d'intérêts concernés et du nombre de documents stockés dans les favoris. Cependant, la simplicité de l'algorithme de création du graphe ne pose pas de problèmes quand il s'agit de représenter quelques centaines de documents.

6.5.6. L'agent de recherche (AR)

Les agents les plus simples du système sont certainement les agents de recherche. Chaque agent de recherche gère un moteur de recherche différent, comme Google, Altavista et Lycos. Comme les agents OMAS évoluent dans un environnement ouvert, les agents peuvent être créés au fil de l'eau ce qui permet l'addition de nouveaux moteurs de recherche à n'importe quel moment, sans que cela n'affecte le fonctionnement d'autres agents. Cette architecture permet également la recherche de sites en parallèle, ce qui entraîne aussi une réduction du temps total de recherche. Afin d'assurer un espace de recherche de documents raisonnable (pas très grand ni trop restreint), chaque agent de recherche a été programmé pour rendre comme réponse 2*N* sites, où *N* est le nombre de sites demandés par l'utilisateur. Ainsi, si le système compte *n* agents de recherche, l'agent de filtrage concerné va analyser jusqu'à 2n*N* sites. Généralement, le nombre de sites analysés sera inférieur à 2n*N*, car certains sites apparaissent souvent dans des moteurs de recherche différents.

6.5.7. La conception du système MAIS

Du point de vue informatique l'agent assistant est composé d'une interface Lisp capable d'envoyer des messages à l'agent de transfert (AT) qui tourne sur un serveur HTTP. Ensuite, l'agent de transfert envoie le message concerné vers l'intérieur de la plate-forme. Nous utilisons le serveur HTTP fourni par Allegro Common Lisp. L'agent de transfert est aussi codé en Lisp. Le choix du serveur Allegro nous permet d'exécuter des fonctions Lisp comme des programmes CGI, ce qui rend la tâche d'interprétation des messages beaucoup plus facile. La Figure 52 montre les ressources mises en œuvre par le système.

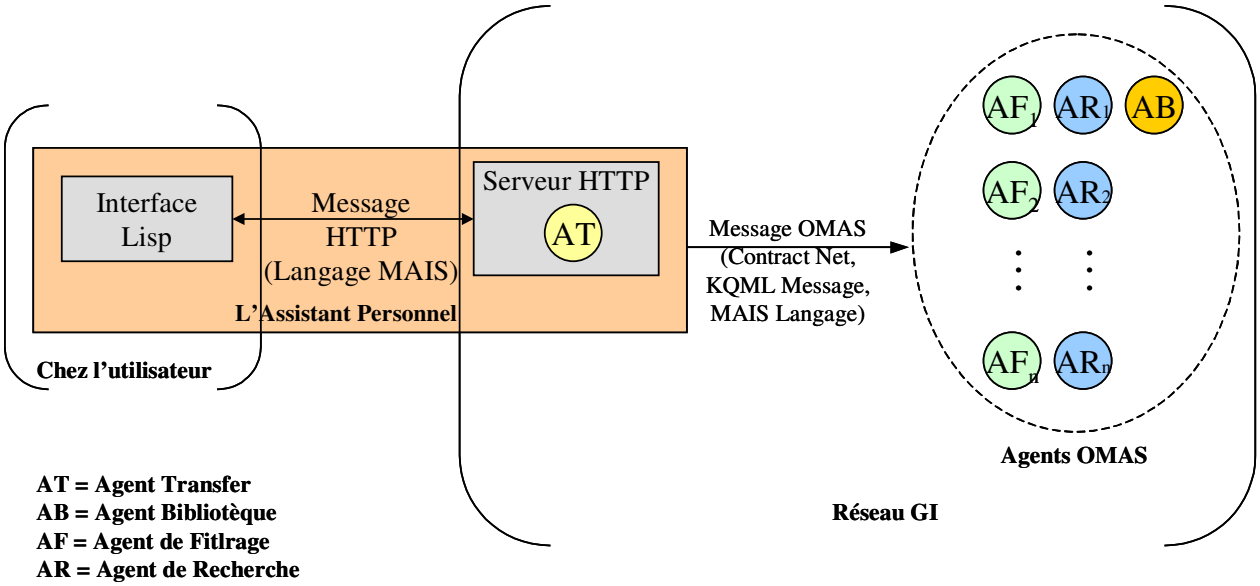

Figure 52 Ressources Informatiques utilisés par le système.

6.6. Le Protocole de Recherche

Pour permettre une communication standard, nous avons développé un protocole utilisant la représentation AUML qui est de plus en plus utilisé (Odell et al., 2000) (FIPAa). Pour cela nous avons dû proposer certains changements dans la représentation AUML à cause de certaines limitations du langage. Les ambiguïtés et les limitations de AUML ont été étudiées en détails par Paurobally et Cunningham (2002). Dans les sections suivantes nous introduisons les concepts de base de AUML, les extensions que nous avons apportées à AUML et finalement nous présentons le protocole de recherche.

6.6.1. AUML

La spécification de modèles et de sémantiques de conversation est un point fondamental dans les systèmes d'agents. Pour suivre des conversations cohérentes, les agents doivent partager la même sémantique et la même compréhension sur le type des messages échangés. Souvent, un protocole commun d'interaction entre agents spécifie le type de chaque message, la sémantique concernée et les changements que ces messages peuvent provoquer dans les agents, par exemple, dans les buts, dans les croyances ou encore dans les états mentaux.

Sous prétexte de créer une forme standard pour représenter des protocoles, Odell et collaborateurs (2000) ont récupéré certains concepts de UML[12] (Unified Modelling Language) et ont proposé le langage AUML (Agent Unified Modelling Language). Ils ont

[12] Le lecteur intéressé est invité à visiter le site de l'OMG: http://www.omg.org.

utilisé certains ressources de UML comme les diagrammes statiques et dynamiques, les représentations conditionnelles, les modèles d'implémentation, dans la conception de nouveaux composants capables de modéliser l'interaction entre agents.

Dans AUML, un diagramme de protocole est un modèle UML dynamique où la dimension horizontale contient les différents rôles des agents et la dimension verticale correspond à la ligne de vie des agents. La Figure 53 montre le protocole FIPA Contract-Net extrait de (FIPAa). Nous discutons par la suite les composants principaux des protocoles AUML.

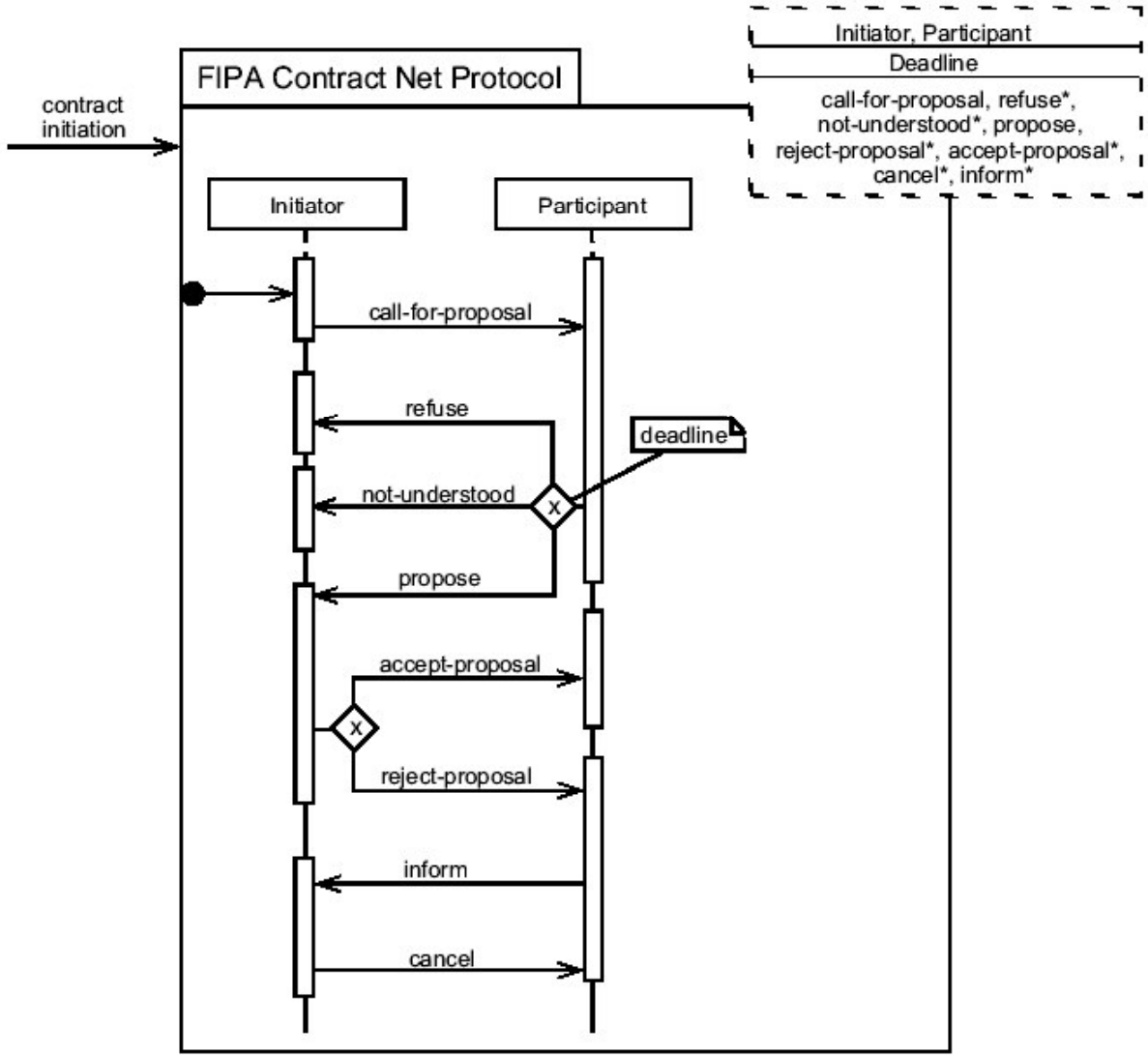

Figure 53 Le Protocole FIPA Contract-Net[13].

6.6.2. La représentation des Messages

Les messages AUML peuvent être synchrones ou asynchrones. Les messages asynchrones sont représentés par des flèches simples (→) tandis que les messages synchrones sont représentés avec des flèches à pointes pleines (—►).

6.6.3. Les lignes de vie parallèles

Les lignes de vie parallèles servent à dénoter les situations de condition. Dans la Figure 53 le losange doté d'un « X » représente un « ou exclusif ». Les losanges vides servent à

[13] Cette figure a été extrait de (FIPAa).

représenter la disjonction simple. Les lignes de vie parallèles qui ne contiennent pas des losanges représentent la conjonction.

6.6.4. Les protocoles imbriqués

Des protocoles complexes peuvent être organisés dans des protocoles imbriqués plus simples. Ces protocoles peuvent être représentés sous de nombreuses formes. Dans les sections suivantes, nous avons découpé le protocole complet en sous-protocoles qui peuvent être imbriqués par les symboles d'entrée (un cercle plein) et de sortie (un cercle entournant un cercle plein).

6.6.5. AUML étendue

L'une des questions importantes dans la modélisation des protocoles concerne la prise en compte du temps. Nous sommes souvent censés représenter des contraintes temporelles, par exemple pour prévoir des conditions d'erreurs comme les « timeouts ». Les représentations de « deadlines » de l'AUML ne sont pas suffisantes pour expliciter des comportements à adopter lorsque un certain temps est écoulé. Par exemple, sur la Figure 53 nous ne savons pas quelle action (refuse, not-understood ou propose) doit être prise quand le deadline est atteint. Par conséquent, nous avons introduit une nouvelle condition de parallélisme pour exprimer les timeouts. Nous avons créé la représentation « conjonction avec timeout » où nous pouvons explicitement représenter les actions à prendre dans le cas des timeouts.

La Figure 54 présente les nouveaux symboles proposés. Dans ces conjonctions, il y a une ligne de vie spéciale marquée d'un cercle plein. Cette ligne de vie est bloquée jusqu'à que le temps *T* soit écoulé, tandis que les autres lignes de vie de la conjonction travaillent normalement. Quand *T* est dépassé, les lignes de vie encore en exécution sont annulées et la ligne de vie marquée par le cercle plein sera déclenchée. Celle-ci peut être mise en œuvre, par exemple, pour annuler des contrats en cours ou fabriquer des messages d'erreur. En revanche, s'il n'y a pas de ligne de vie en cours d'exécution (toutes sont déjà finies) quand *T* est dépassé, rien n'est fait. L'extension de la sémantique introduite par ce mécanisme à la disjonction et à la disjonction exclusive est donc immédiate.

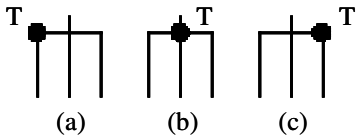

Figure 54 Extension de la conjonction AUML. Ici (a), (b) et (c) représentent “conjonction avec timeout”.

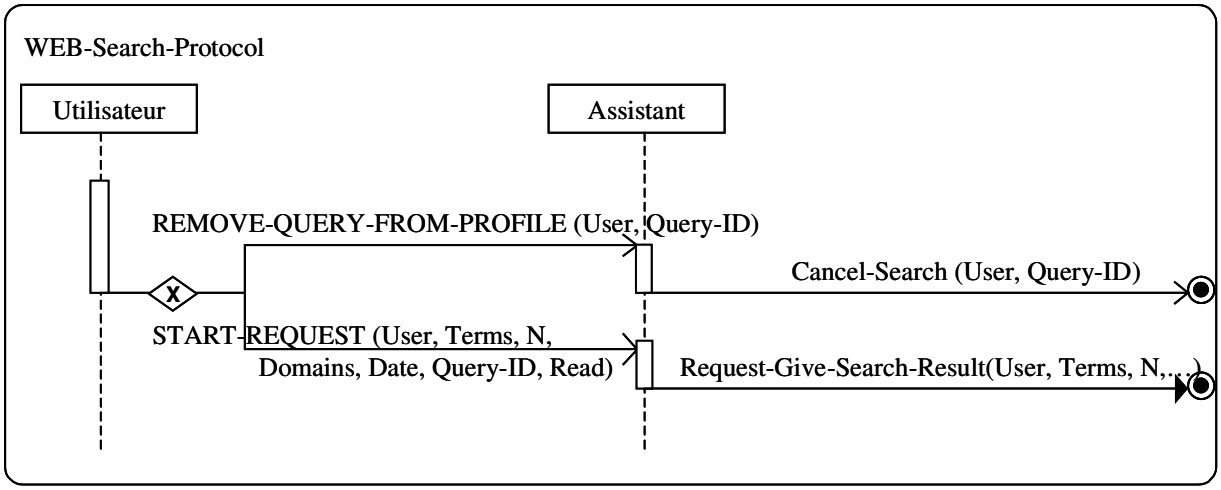

Figure 55 Protocole d’interaction - Partie I.

6.6.6. Le Protocole de Recherche

La Figure 55 et la Figure 56 montrent le protocole d’interaction partagé entre les agents du système MAIS. Dans le protocole, nous représentons l’utilisateur et trois autres types d’agents : Assistant, Filtrage et Recherche.

La Figure 55 montre que l’utilisateur peut démarrer le processus de recherche ou supprimer une recherche qui est en train de tourner ou qui a déjà fini. Pour cela, il dispose de l’interface présentée sur la Figure 48. Le démarrage d’une recherche n’est pas synchrone. C’est-à-dire, l’utilisateur déclenche une recherche, mais les autres fonctionnalités du système sont encore disponibles. L’utilisateur peut déclencher plusieurs recherches à la suite. Ces recherches sont traitées de façon séquentielle par les agents du système. À tout moment l’utilisateur peut annuler une recherche.

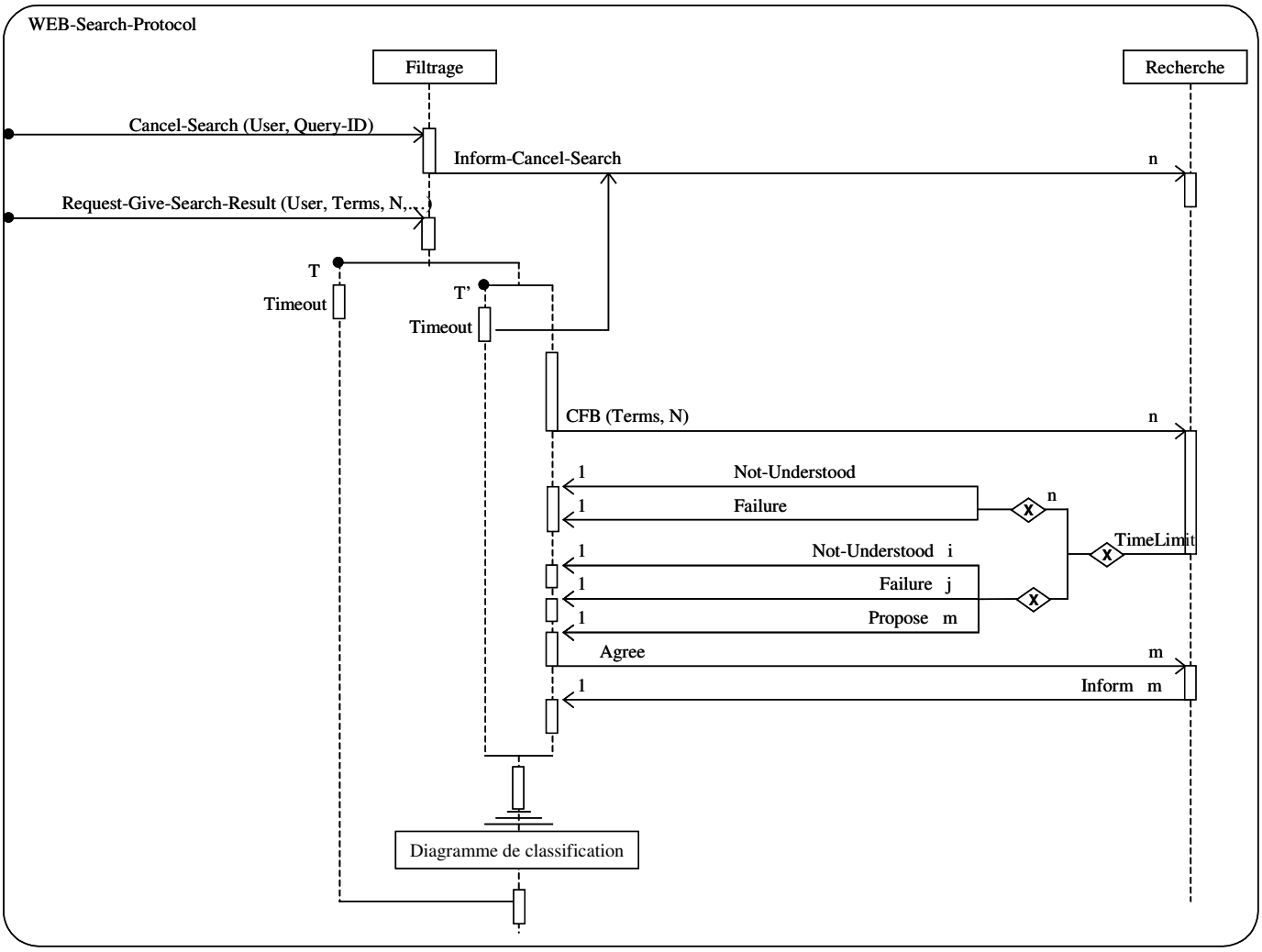

Figure 56 Protocole d'interaction - Partie II.

Sur la Figure 56 l'agent filtrage reçoit une requête et demande des sites aux agents de recherche. Il attend l'arrivée des réponses pendant un certain temps (à ce jour la valeur de T' est de 20 secondes). Les sites arrivés dans les 20 secondes sont alors traités selon le processus affiché dans les diagrammes de classification et traitement de la Figure 57 et de la Figure 58 respectivement. Nous avons aussi mis à disposition un timeout global (la valeur *T*) pour la recherche. Ainsi l'agent peut annuler une recherche qui dure par exemple plus d'une heure.

Sur la Figure 57 l'agent filtrage classe la liste de sites reçue des agents de recherche et choisit parmi ceux-ci une partie des sites, selon le nombre de sites demandés par l'utilisateur. Ensuite la réponse est enregistrée dans l'espace de travail de l'utilisateur. Le diagramme de traitement (Figure 58) décrit le processus de manipulation de chaque site dans le processus de classification. Pour chaque site de la liste la page correspondante est téléchargée, le contenu de la page est extrait, les termes existants dans le profil de l'utilisateur sont alors identifiés dans le contenu de la page et un vecteur de termes est créé. Ensuite le vecteur est classé selon

le graphe et le site mis dans la queue de priorité selon la valeur trouvée lors de la classification. Finalement la liste de sites ordonnée est retournée.

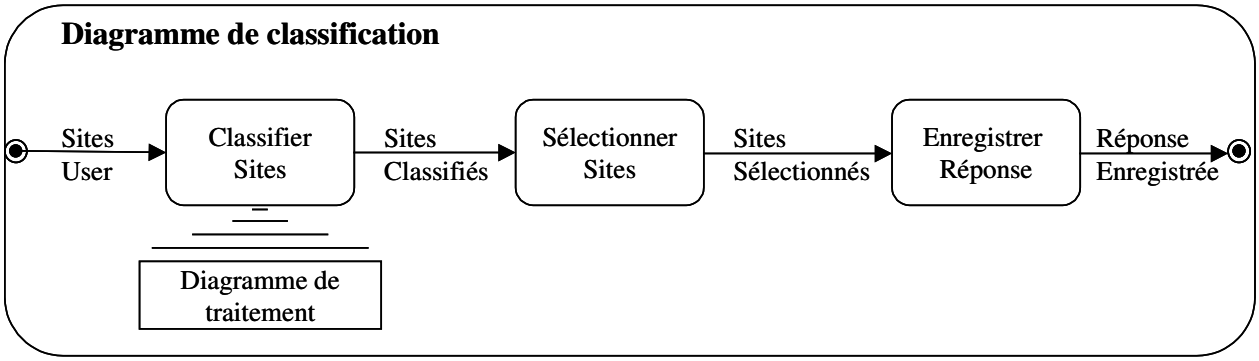

Figure 57 Le processus de classification des sites.

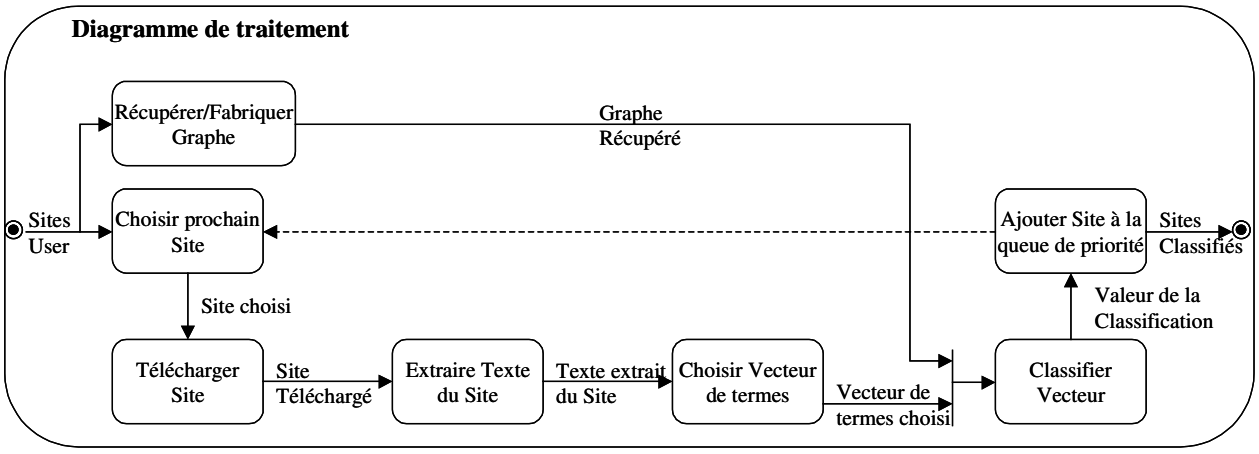

Figure 58 Processus de traitement de chaque site.

6.7. Qualité des Résultats

L'évaluation des mécanismes de recherche sur la toile est toujours une tâche compliquée. Les mesures couramment utilisées par les systèmes de récupération d'information comme *recall* et *precision* sont insuffisantes en raison de la dynamicité du corpus de pages. Malgré cela, la grande majorité de travaux concernant l'évaluation de mécanismes de recherche utilisent un corpus de pages pré-défini. Les pages sont ordonnées selon une mesure de similarité calculée automatiquement entre les résultats et les pages de référence. Par exemple, (Menczer et al., 2001) ont utilisé 100 catégories et pages de référence proposés par Yahoo pour comparer les résultats de trois mécanismes de recherche. Dans ce cas, les ensembles de pages utilisées comme références et comme tests sont statiques et de taille limitée. Pour éviter ces problèmes, Gasparetti et Micarelli (2003) ont utilisé les mesures *recall* et *precision* sur la base « .gov » proposée par TREC (Text REtrieval Conference - http://trec.nist.gov). Cette base compte 1.2

millions de pages « .gov » répertoriées jusqu'à 2002 et, selon les auteurs est suffisamment importante pour être représentative d'une collection de documents de la toile. Les auteurs ont évalué les résultats sur trois requêtes : « exchange office », « educational department » et « children foundation ». Le nombre de résultats moyens pour chaque requête est 373712, 445964 et 117036, respectivement. Nous pensons que dans une approche réaliste, le nombre de résultats ne doit pas être trop important, car l'utilisateur n'analysera que les premiers d'entre eux (Silvertein et al., 1999).

Dans notre approche d'évaluation, nous utilisons l'avis de trois utilisateurs sur les résultats obtenus directement sur la toile. Nous n'utilisons pas de bases de données simulées. Les utilisateurs analysent seulement les vingt ou trente premières pages en disant combien de pages sont intéressantes pour eux. Tous les utilisateurs cherchent des informations concernant le domaine des systèmes multi-agents. Chaque utilisateur a été invité à choisir un nombre de pages de référence. Le Tableau 12 donne plus de détails sur les utilisateurs.

Utilisateur	Expertise	Sites de référence	Centres d'intérêts
A	Systèmes MultiAgent	58	Adaptative-Agents Autonomous-Agents Agents Personal Assistant
B	Intelligence Artificielle	8	Adaptative-Agents Autonomous-Agents Agents Personal Assistant
C	Aucune	1	-
C-int	Aucune	1	Adaptative-Agents Autonomous-Agents Agents Personal Assistant

Tableau 12Informations sur les utilisateurs.

Tous les utilisateurs ont exécuté les requêtes suivantes : « agent », « autonomy agent », « personal assistant » et « personal assistant agent ». Ces requêtes sont intentionnellement ambiguës. Le but du mécanisme d'évaluation est de savoir jusqu'à quel point le système est capable de filtrer des informations intéressantes en utilisant l'expertise de chaque utilisateur comme référence. Nous mesurons également l'influence des centres d'intérêts publics proposés par le système pour l'utilisateur C, qui a été invité à réaliser les requêtes d'abord sans utiliser les centres d'intérêts publics et, dans un deuxième temps, en les utilisant.

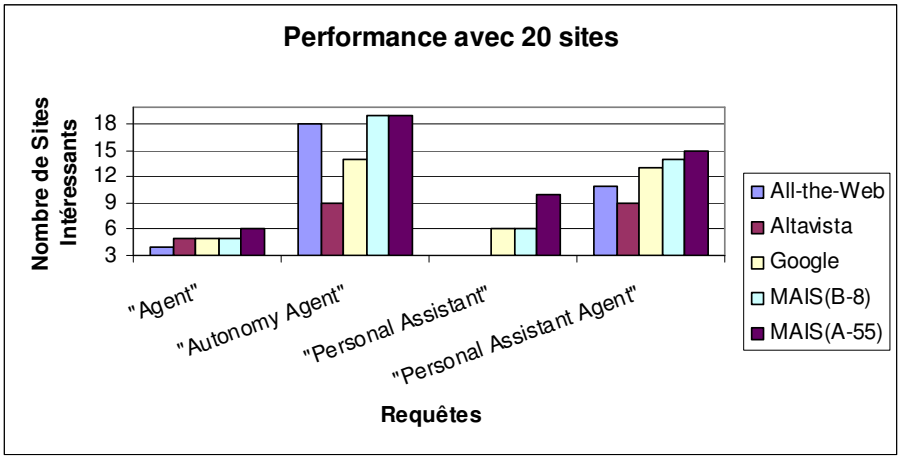

Figure 59 Performance mesurée sur les 20 premiers sites.

Le graphique présenté sur la Figure 59 illustre la performance de trois moteurs de recherche bien connus (All-the-Web, Altavista et Google) et les résultats des utilisateurs A (A-55) et B (B-8) qui ont utilisé MAIS. Comme nous nous y attendions, l'utilisateur A obtenu les meilleurs résultats pour toutes les expériences, ce qui démontre la capacité du système de tenir compte du contexte de l'utilisateur sous forme de ses pages favorites pour la personnalisation des résultats. L'utilisateur B a obtenu les mêmes améliorations.

Deux observations semblent évidentes à partir des résultats de la Figure 59:

- La première est la limitation de la performance du système MAIS concernant la qualité des moteurs de recherche ;
- La variabilité des moteurs de recherche.

Puisque MAIS réalise des méta-recherches, si les moteurs utilisés par MAIS sont incapables de trouver des pages intéressantes, MAIS sera également incapable de donner de bons résultats. Cela se confirme avec la requête « Agent » qui est extrêmement ambiguë.

Certains moteurs de recherche présentent de bonnes performances pour certaines requêtes, alors que pour d'autres les résultats sont très pauvres. Nous pouvons constater, par exemple, que All-the-Web, donne des bons résultats pour la requête « Autonomy Agent » et des résultats nuls pour « Personal Assistant ».

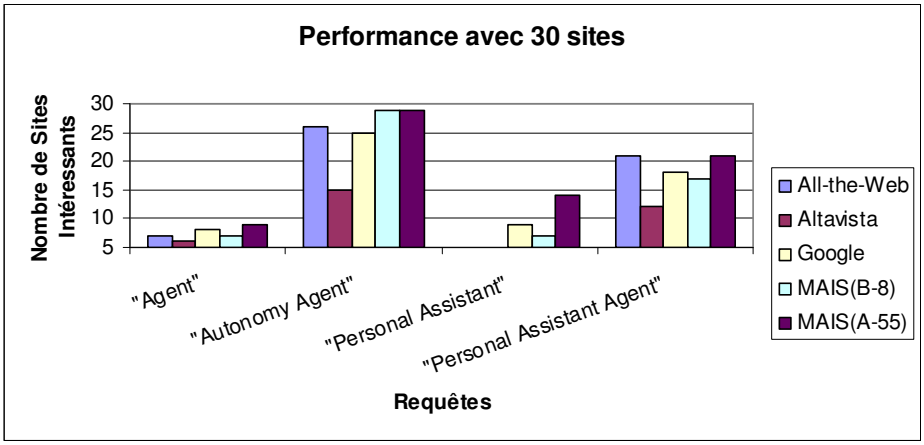

Figure 60 Performance mesurée sur les 30 premiers sites.

Lorsque les 30 premières pages ont été analysées, MAIS donne encore les meilleurs résultats pour l'utilisateur A, voir Figure 60. En revanche, la performance du système concernant l'utilisateur B est dépassée par d'autres moteurs de recherche sur 3 sur 4 requêtes. Cela démontre que lorsqu'un utilisateur possède un profil incompatible avec les informations recherchées, le système ne pourra pas l'aider. Ce problème peut être résolu simplement en informant le système que le profil de l'utilisateur ne devra pas être utilisé pour la recherche, mais qu'il faudra utiliser les centres d'intérêts publics.

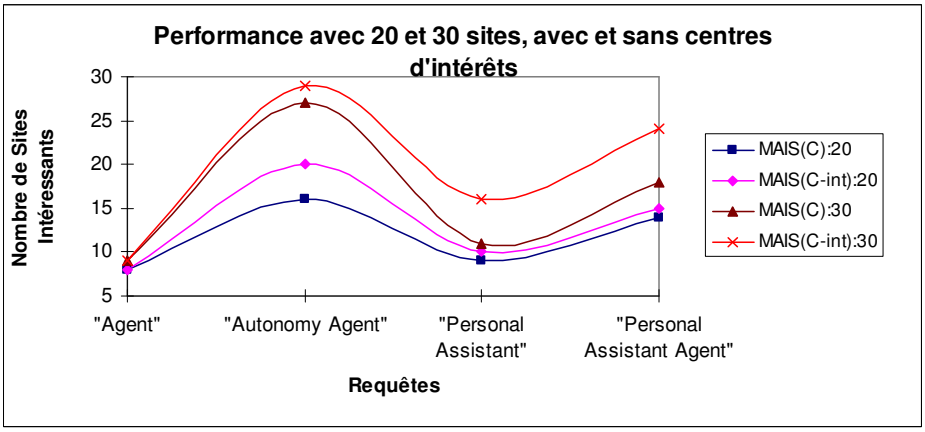

Figure 61 Résultats avec et sans les centres d'intérêts publics.

Afin de mesurer l'importance des centres d'intérêts publics pour la recherche d'information pour le système MAIS, nous avons demandé à l'utilisateur C qui possède un profil pratiquement vide, de réaliser des recherches sans référencer les centres d'intérêts « MAIS(C) » et en référençant les centres d'intérêts « MAIS(C-int) ». Les résultats sont indiqués sur la Figure 61. Pour la requête « Agent » l'utilisation des centres d'intérêts est indifférente, car la requête est trop imprécise. Pour la requête « Autonomy Agent » les centres

d'intérêts ont amélioré énormément la performance tant pour les 20 premières pages que pour les 30 premières. Pour les deux dernières requêtes, la performance du système a été sensiblement améliorée lorsque les 30 premières pages sont analysées. Lorsque seulement les 20 premières pages sont analysées les centres d'intérêts ont peu d'importance pour les requêtes « Personal Assistant » et « Personal Assistant Agent ». Ceci démontre que même pour des requêtes raisonnablement précises, comme « Personal Assistant Agent », les moteurs de recherche utilisés par MAIS ne sont pas capables de grouper les meilleures pages et de les présenter en priorité.

Avec 20 sites			Avec 30 sites	
MAIS(C-int)	13,25		MAIS(C-int)	19,5
MAIS(A-55)	12,5		MAIS(A-55)	18,25
MAIS(C)	11,75		MAIS(C)	16,25
MAIS(B-8)	11		MAIS(B-8)	15
Google	9,5		Google	15
All-the-Web	8,25		All-the-Web	13,5
Altavista	6,5		Altavista	9,25

Tableau 13Résultats moyens en nombre de sites intéressants sur les 4 requêtes.

La performance moyenne pour tous les moteurs de recherche sur les 4 requêtes est surprenante (Tableau 13). Analysant les 20 et 30 premières pages, MAIS a dépassé tous les moteurs de recherche. Ceci montre que MAIS est capable de profiter des points positifs de tous les moteurs de recherche particuliers en produisant un résultat global plus intéressant. La surprise vient surtout de l'utilisateur gagnant (C-int) qui contient un profil pratiquement vide, mais utilise les centres d'intérêts publics. Ces résultats montrent que les pages jugées intéressantes par les utilisateurs ne rajoutent pas forcément beaucoup d'informations importantes pour le système. En effet, plusieurs pages peuvent même diminuer la précision du système, car elles ne contiennent pas beaucoup de texte et introduisent beaucoup de bruit. Par ailleurs, le système est plus précis lorsque l'utilisateur choisit les centres d'intérêts publics, car ceux-ci sont composés de documents validés par des experts. Les documents sont représentatifs du domaine et contiennent beaucoup d'informations textuelles. Nous pouvons donc conclure que les utilisateurs devront analyser soigneusement les pages qui leur conviennent avant de les ajouter à leurs listes de favoris.

6.8. Discussion et travaux concernés

Dans cette section nous comparons notre approche de recherche sur le web à celles d'autres projets de la littérature.

6.8.1. Qualité du profil de l'utilisateur

Le travail de Lieberman (1997) a mis en place des techniques de récupération d'information dans la conception d'un (butineur) capable de proposer des sites à l'utilisateur tandis que celui-ci navigue sur l'Internet. Pour cela, un agent assistant, LETIZIA, analyse le contenu de la page courante, fait une recherche en largeur sur les liens trouvés dans cette page et affiche des sites jugés intéressants. Nous pouvons dire que ce système gère un profil de l'utilisateur à court terme modélisé pendant la session de navigation. D'autres approches semblables à celle-ci ont été mises en place dans les applications WebWatcher (Armstrong et al., 1995) et WebMate (Chen et Sycara, 1998) développées à CMU. Plus récemment, des applications commerciales et des systèmes de recommandation (Mobasher et al., 2000) mettent en œuvre des méthodes d'apprentissage automatique et de récupération d'information pour améliorer la qualité des services. La plupart de ces travaux ont en commun l'usage de l'Internet comme source d'apprentissage des modèles de l'utilisateur. Nous pensons toutefois qu'il est très difficile d'apprendre à partir de l'Internet pour plusieurs raisons : (i) les pages web sont très bruitées et n'ont pas été conçues pour servir de base à l'apprentissage automatique (par exemple, il est commun trouver des pages qui concernent plusieurs sujets) ; (ii) la plupart des techniques d'apprentissage du texte et de récupération d'information ont besoin de beaucoup d'information textuelle, mais des nouvelles technologies comme les flashes ou encore les applets java diminuent énormément la quantité de texte dans les sites. Ces problèmes demandent une phase de pré-traitement de l'information très coûteuse pour assurer la qualité des modèles. En revanche, dans notre approche une partie du modèle de l'utilisateur (les centres d'intérêts publics) est fabriquée à partir des documents représentatifs du domaine, comme des articles ou des rapports, qui sont des données fiables et moins bruitées.

6.8.2. Couverture de l'espace Internet

Au-delà de la personnalisation, une application de recherche d'information sur le web doit couvrir l'espace de recherche le plus large possible. Calvo (2000) soutenait en 1999 qu'aucun moteur de recherche n'indexait plus de 20% de l'univers de pages web. Cela n'est pas très différent aujourd'hui. Par ailleurs, des moteurs de recherche ont des bases de données incomplètes, périmées ou dépendant de la région où ils se trouvent. Des meta-moteurs sont souvent mis en place pour augmenter la couverture de l'espace Internet réalisant des recherches simultanément avec plusieurs autres moteurs de recherche. Les résultats sont souvent médiocres, car ils tendent à reproduire les heuristiques et les limitations des moteurs

employés dans la recherche. Dans (Lawrence et Giles, 1998) les auteurs ont présenté un meta-moteur pour le web qui fournissait de meilleurs résultats. Cependant, le système n'est pas capable de distribuer la charge de travail, ni de traiter des requêtes multiples. Dans ce travail, nous utilisons des agents de recherche encapsulant différents moteurs de recherche. Les agents évoluent dans un environnement ouvert où des agents recherche peuvent apparaître ou disparaître sans perturber le système. Ces agents fournissent des pages qui sont analysées par d'autres agents. Par conséquent, cette architecture distribue la charge de travail. De plus, pour réduire l'influence négative des moteurs de recherche sur le résultat, les agents de filtrage analysent chaque page à partir d'un espace de recherche plus importante que celui demandé par l'utilisateur.

6.8.3. Des informations sémantiques

Dans (Corchuelo et al., 2002) les auteurs présentent un système multi-agents pour la recherche sur l'Internet. Le système met en œuvre des agents « brokers » et différents canaux de communication pour rechercher des informations sémantiques. Les canaux de communication sont des systèmes experts spécialisés dans l'extraction d'information sémantique de certains sites. Chaque agent utilise des définitions sémantiques codées dans des ontologies et une base de règles pour analyser et récupérer l'information. Les agents « brokers » sont utilisés pour trouver des canaux d'information spécifiques selon les services demandés. Dans notre système MAIS, nous ne sommes pas intéressés dans la récupération d'information à travers des modèles sémantiques. Les représentations sémantiques peuvent être utiles pour améliorer la finesse des résultats, mais sont toujours restreintes à certains domaines. En revanche, nous sommes intéressés par des stratégies génériques qui peuvent être mises en œuvre dans n'importe quel domaine.

6.9. Conclusions

Dans le chapitre précédent nous avions présenté une nouvelle méthode d'apprentissage dynamique destinée aux agents assistants adaptatifs, la méthode ELA. Dans ce chapitre nous avons utilisé la méthode ELA dans le développement d'un système multi-agents capable de personnaliser la recherche d'information sur le web. Les caractéristiques de la méthode ELA comme l'apprentissage incrémental, la simplicité de représentation du modèle des données, la vitesse de classification et précision se sont avérés très importantes dans la conception de cette application.

La recherche d'information sur l'Internet est toujours un processus fastidieux qui demande l'exécution de plusieurs requêtes souvent avec des moteurs de recherche différents. Nous avons développé le système MAIS capable de filtrer et de classer les résultats de plusieurs moteurs de recherche en prenant en compte les meilleurs résultats de chacun d'entre eux. Par ailleurs, les résultats sont rendus selon le profil de l'utilisateur, conçu à partir de documents favoris et centres d'intérêts publics.

Nous avons distribué la recherche sur l'Internet sur plusieurs agents jouant différents rôles. Le système bénéficie des caractéristiques inhérentes aux systèmes multi-agents comme leur extensibilité, leur stabilité et la possibilité d'équilibrage des charges. Les agents communiquent à l'aide d'un protocole d'interaction représenté ici dans le formalisme AUML. Les résultats obtenus sont meilleurs qui ceux proposés par les moteurs de recherche traditionnels.

Nous pensons que des systèmes multi-agents peuvent donner de meilleurs résultats sur l'Internet que les agents assistants spécialisés. Cependant, le choix de l'architecture du système n'est pas une tâche simple et requiert l'étude des paramètres physiques comme la distribution de la complexité de développement et de calcul, les ressources réseaux et la performance et, plus encore, les paramètres humains comme les préférences de l'utilisateur et l'interaction avec le système.

Contrairement à la plupart des assistants destinés à la recherche sur l'Internet, notre système peut s'adapter au profil de l'utilisateur en utilisant la méthode d'apprentissage ELA. Cette méthode a été conçue pour faire évoluer des modèles internes d'un agent assistant. Bien que dans notre application la méthode ELA soit utilisée par l'agent *Filtrage*, nous pouvons dire que le système tout entier joue le rôle d'assistant.

7 Un SMA personnalisé

Tout au long de ce mémoire nous avons présenté plusieurs d'applications d'agents assistants et surtout, nous avons mis l'accent sur l'aspect adaptatif de ces applications. Parmi les approches développées : l'interaction homme-machine avec un système de dialogue, la gestion documentaire avec la classification et le filtrage automatique de documents, la capitalisation et le partage des connaissances avec des agents de recommandation d'information et la personnalisation d'informations pour la recherche sur la toile. Chaque application développée comptait un certain nombre d'agents assistants personnels spécifiques.

Nous sommes persuadés que l'utilisateur doit interagir avec un seul agent assistant selon le principe du majordome qui coordonne les différentes fonctionnalités du système. Pour garder une certaine cohérence, nous présentons dans ce chapitre un modèle d'agent assistant capable de grouper sous un seul toit les différentes applications présentées précédemment.

7.1. Les architectures des SMA et les assistants

N'importe quelle application destinée à l'assistance peut être décomposée en un système multi-agents doté d'un agent assistant et d'agents de service afin de rendre l'architecture du système plus souple et de limiter la complexité des agents en distribuant les services.

Dans cette section nous discutons les architectures possibles d'un système distribué qui fournit des services spécialisés. Certaines applications peuvent donner de meilleurs résultats lorsqu'elles sont développées sous forme d'un SMA (voir le travail de Huhns et Singh (1998) pour une discussion générale sur SMA), car les SMAs peuvent être facilement étendus, sont normalement résistants à des situations inattendues et peuvent distribuer naturellement la

charge de travail. Cependant, le développement des SMAs interactifs (qui contiennent des agents qui communiquent avec l'utilisateur) doit prendre en compte non seulement des *facteurs matériels*, comme les ressources du réseau et le temps de calcul des machines, mais aussi des *facteurs humains* comme la personnalisation, la charge de travail et l'interaction avec l'utilisateur.

Dans cette section nous discutons plusieurs architectures possibles des SMA qui prennent en compte l'existence de l'utilisateur. Par ailleurs, nous adaptons une architecture fondée sur la métaphore du majordome (voir section 7.1.2). Dans cette approche, l'utilisateur communique toujours avec un seul agent, qui communique à son tour avec d'autres agents pour satisfaire les besoins de l'utilisateur.

7.1.1. Des SMA interactifs non-personnalisés

Dans des SMA qui n'ont pas d'agents assistants personnels personnalisés, l'utilisateur peut travailler directement avec différentes applications ou utiliser un assistant public qu'il partagera avec d'autres utilisateurs. La Figure 62(a) et la Figure 62(b) illustrent respectivement ces deux situations.

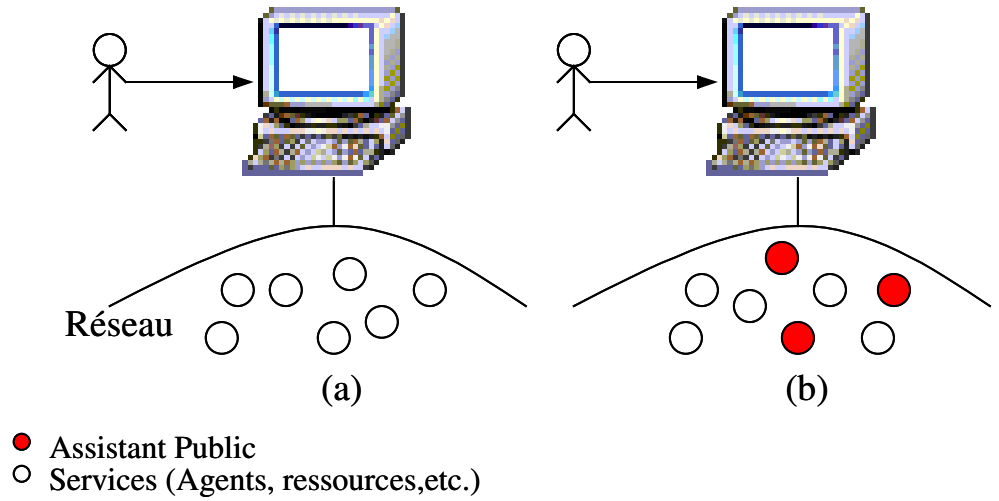

Figure 62 Les SMAs non-personnalisés.

Lorsque l'utilisateur interagit directement avec des ressources et différentes applications (Figure 62a), certains problèmes apparaissent immédiatement. Tout d'abord, ces applications ne possèdent pas d'interfaces normalisées du point de vue ergonomique, ce qui entraîne une certaine difficulté d'adaptation au système. Ensuite, ces applications souvent ne sont pas capables de personnaliser les différents services, ni l'interaction avec l'utilisateur.

Dans certains environnements (par exemple, sur l'Internet) un nombre d'agents assistants publics est disponible (Figure 62b). Ces agents sont, normalement, capables de fournir des résultats plus précis, car ils échangent des informations avec d'autres agents et peuvent avoir accès à des ressources inconnues de l'utilisateur. Cette approche est souvent mise en place dans des meta-systèmes de recherche d'information (le site http://www.botspot.com/ présente plusieurs outils commerciaux disponibles). Cependant, des applications différentes utilisent des agents différents, ce qui ne règle pas le problème de manipulation de plusieurs interfaces. Dans ce cas, l'utilisateur est obligé de s'adapter aux différents agents.

En outre, les agents assistants présentés dans la Figure 62b n'échangent pas d'informations personnelles sur l'utilisateur, ce qui provoque forcement des réplications et un stockage d'informations redondantes. Ce problème a été traité par Crabtree (1998), qui a proposé une approche de personnalisation où un agent *Profile* centralise les informations personnelles des utilisateurs. Ensuite ces informations son utilisées par plusieurs applications différentes (recherche d'information, présentation de nouvelles informations et recherche de personnes ayant des intérêts similaires).

7.1.2. Des SMA interactifs personnalisés : le concept du *majordome*

L'utilisation des assistants personnels rajoute une complexité importante dans l'architecture d'un SMA. Nous pensons qu'il est préferable que l'utilisateur travaille avec un agent assistant personnel unique. Cet assistant pourra avoir un corps technique d'agents, appelé *staff*, qui l'aideront à trouver des solutions aux problèmes posés par l'utilisateur. Cette approche est connue comme l'approche du *majordome* et a été proposée par Negroponte (1997).

D'après Negroponte un agent assistant personnel doit être spécialisé dans l'interaction avec son maître, de la même façon qu'un majordome doit connaître son maître pour anticiper et satisfaire ses besoins de la meilleure façon qui soit. L'assistant pourra faire appel à des services d'autres agents spécialisés mais son rôle principal est de connaître son maître.

À partir du concept d'agent assistant personnel du type majordome, nous pouvons identifier trois déploiements possibles d'un SMA personnalisé (Figure 63) :

- Agent assistant personnel avec Staff ;
- Agent assistant personnel simple et coterie ;

· Agent assistant personnel et staff dans une coterie.

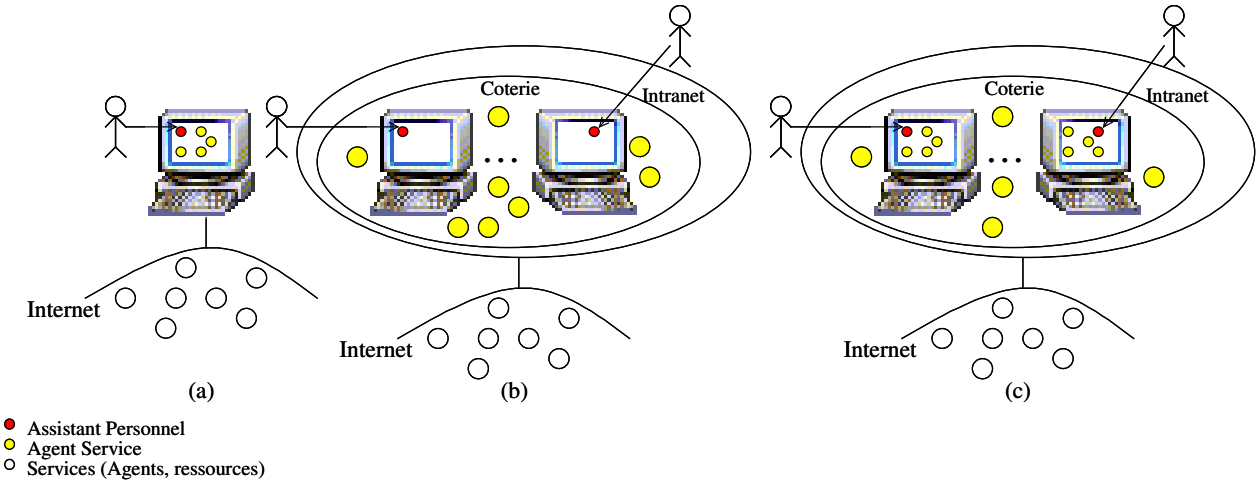

Figure 63 Les architectures des SMAs (a) PA avec staff (b) PA simple et coterie (c) PA et staff dans une coterie.

7.1.3. L'assistant et son staff

La Figure 63a illustre l'idée que l'assistant a un staff d'agents locaux spécialisés dans le traitement de services particuliers. Tous ces agents tournent sur la machine de l'utilisateur. Le rôle du PA est donc de coordonner son staff et de veiller à l'interaction avec l'utilisateur. L'assistant dialogue avec des agents locaux qui peuvent faire appel à des services extérieurs.

La personnalisation

En théorie, cette architecture doit donner de bons résultats, car le groupe d'agents peut faire usage du profil et des préférences de l'utilisateur pour adapter ses services. Par conséquent les services (même distribués) peuvent être personnalisés. Comme les services sont en général bien déterminés, l'assistant peut avoir une représentation relativement statique de chaque agent de son staff ce qui facilitera la distribution des tâches.

Les ressources

Cette architecture, ne demande pas beaucoup de ressources de la part du réseau car les messages sont rarement échangés entre des agents de machines différentes. Evidemment, la machine locale doit être suffisamment puissante pour accepter tous ces agents. La complexité de construction du système n'est pas très grande, car chaque agent peut être développé indépendamment.

La nature centralisée du système peut entraîner des soucis quand les agents ont besoin de garder une certaine cohérence avec d'autres systèmes distribués, notamment quand la machine de l'utilisateur n'est pas en ligne. Par exemple, considérons un SMA pour la gestion commerciale d'une entreprise où nous pouvons trouver des agents spécialisés dans la gestion des clients, des commandes, des entrepôts, etc. Chaque vendeur possède un staff d'agents sur son ordinateur portable. Puisque l'ordinateur du vendeur peut ne pas être toujours connecté au système de gestion de l'entreprise (par exemple, lorsqu'il fait une vente avec son portable dans un endroit où il n'y a pas de réseau), les agents ne peuvent pas récupérer en ligne des nouvelles consignes ou contraintes sur les ventes établies par l'entreprise.

7.1.4. L'assistant dans une coterie

Dans la Figure 63b les agents de service sont publics. Contrairement à la Figure 63a, seul l'agent assistant est personnalisé. Une telle stratégie simplifie beaucoup le développement, la mise en place et la mise à jour du système, car il y a moins d'agents à gérer. Dans ce cas, le réseau doit gérer les nombreux messages échangés.

La personnalisation

La personnalisation doit être prise en main par l'assistant qui doit gérer les informations de natures différentes produites par les agents service. Étant donné que les agents service sont extérieurs et n'ont pas accès aux informations personnelles de l'utilisateur, certains agents peuvent demander plusieurs fois le feedback de l'utilisateur, ce qui peut entraîner une surcharge de travail et une redondance d'informations. Cette question a été étudiée par Koch et Lacher (2000). Les auteurs ont proposé une architecture fondée sur des agences de profils des d'utilisateurs. La création du profil des utilisateurs est confiée à des agents jouant le rôle d'agence. Lorsqu'une application a besoin d'une information spécifique sur l'utilisateur elle fait appel aux agences.

Les ressources

Évidemment, le fonctionnement du SMA est complètement dépendent du réseau, à la différence de l'architecture présentée dans la section précédente. Si nous reprenons l'exemple de la section précédente, par exemple, le vendeur ne peut pas travailler sans avoir une connexion réseau.

Puisque les agents service peuvent être distribués dans le réseau et des nouveaux services peuvent être ajoutés ou supprimés au fil de l'eau, l'assistant doit être capable d'identifier les compétences des agents services et fabriquer une représentation de chaque agent du système. Dans le système RETSINA de Sycara et collaborateurs (2001), cette tâche est accomplie avec des agents intermédiaires appelés *Matchmakers*. Ces agents fabriquent dynamiquement une représentation des services et des agents qui les fournissent. Pour cela ils observent les messages de demandes et de propositions de services échangés dans le SMA. Quand un agent a besoin d'un service il peut demander aux agents matchmakers des informations concernant l'identité des agents capables de fournir les services, comme par exemple, son nom et son adresse. La question de distribution de services est à ce jour très étudiée par les chercheurs intéréssés aux échanges de type Peer-2-Peer (Koubarakis, 2003) (Dimakopoulos et Pitoura, 2003).

Lorsque le système est relativement statique et que les services ne changent que rarement, un modèle de pages jaunes peut être envisagé. FIPA propose pour cela la définition de *Directory Facilitator* (voir FIPAb), qui fournit une série de services d'enregistrement et de localisation de services et agents. Cependant, ce modèle ne peut être utilisé dans un système ouvert, où les agents peuvent aller et venir n'importe quand, car aucun mécanisme n'assure que les services sont disponibles à un moment donné.

7.1.5. L'assistant et le staff dans une coterie

Dans la Figure 63c le système présente des agents service publics et personnels. Cette technique hybride peut être prise en compte dans la majorité des applications de SMA pour l'Internet. Le système met en œuvre la personnalisation et réduit les problèmes de construction, de mise en route, de mise à jour, et la gestion des ressources du réseau et de l'ordinateur de l'usager, car le calcul est réparti entre la machine locale et le réseau. De plus, cette architecture évite les problèmes de réplications d'information présentés dans la section 7.1.1.

La personnalisation

Dans cette architecture l'assistant ne dialogue pas avec les agents services extérieurs. En revanche les agents service locaux peuvent faire appel aux services des agents de la coterie et du réseau extérieur (Internet). Les agents locaux peuvent alors profiter des informations personnelles fournies par l'assistant pour personnaliser des services publics.

Les ressources

Cependant, dans ce modèle, un canal de communication doit faire le pont entre les messages des agents locaux (qui tournent dans la machine de l'utilisateur) et les agents extérieurs (qui tournent en dehors la coterie, par exemple, sur l'Internet). Puisque dans une collection d'agents d'une plate-forme les messages échangés sont normalement dans un format privé, nous avons besoin d'un mécanisme de normalisation de la communication. La manière la plus simple de le faire est de fabriquer un *Agent de Transfer* capable de comprendre des messages privés et les messages échangés avec des agents Internet, très probablement fabriqués à partir d'une autre plate-forme. Ce problème apparaît également dans l'architecture de la section 7.1.3. Nous avons utilisé cette idée dans pour concevoir l'Agent Transfer discuté dans la section 2.5.3.

7.2. Idées générales

Nous avons vu dans les sections précédentes comment la conception d'un système multi-agent interactif peut être améliorée lorsque le système dispose d'un agent assistant spécialisé dans l'interaction avec l'utilisateur entraînant une amélioration de la communication entre l'utilisateur et le système et la possibilité de personnalisation de l'interaction et des services.

Dans un tel système, l'exécution de services et l'entrée d'informations par le maître peuvent être mises en œuvre avec une seule interface utilisant le langage naturel. Pour cela, nous avons développé dans la section 3.3 un module de dialogue. Par ailleurs, des recherches dans cette direction sont actuellement menées par Emerson Paraiso au sein du laboratoire (Paraiso et Barthès, 2003).

Parmi les applications développées dans cette thèse, celle qui présente le modèle plus complexe est le système d'interaction par dialogue. Il est clair que le module de dialogue doit être le cœur d'un SMA destiné à l'aide personnalisée. Nous pouvons ensuite placer autour du système de dialogue les différentes applications développées. La Figure 64 illustre notre raisonnement.

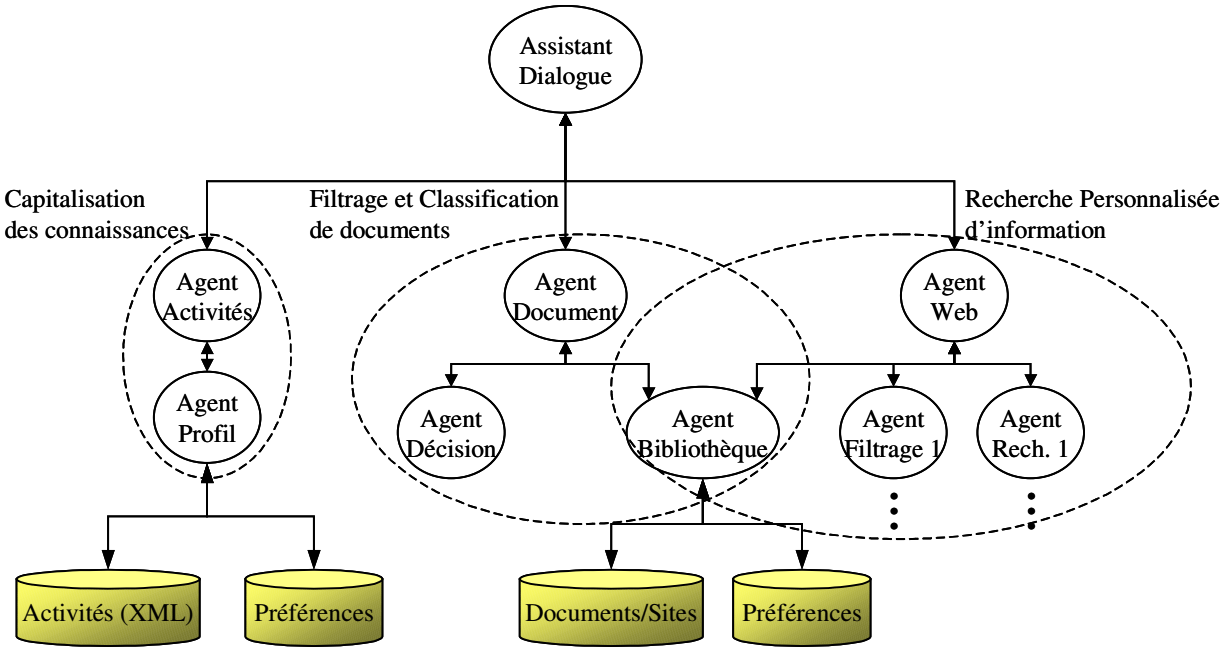

Figure 64 Architecture d'un SMA destiné à la personnalisation de services.

Dans la Figure 64 nous représentons les applications développées au cours de la thèse : la capitalisation des connaissances, le filtrage et la classification de documents, et la recherche personnalisée d'information. Nous avons gardé pour chaque application les mêmes agents mais certains ont été renommés simplement pour donner une meilleure cohérence au système, par exemple, l'agent assistant responsable de la capitalisation des connaissances a été renommé « Agent Activités », car désormais l'assistant dialogue est l'intermédiaire entre l'utilisateur et les applications. Les interfaces présentées dans ces applications peuvent être à la fois pilotées par le système de dialogue et contrôlées normalement par l'utilisateur, car dans certains cas, notamment pour les interactions qui ne demandent pas de connaissances (par exemple, la sélection d'un fichier sur le disque dur) le langage naturel peut compliquer ce qui peut être accompli facilement à l'aide d'interfaces classiques.

7.2.1. La capitalisation des connaissances

Dans la section 4.4 nous avons présenté un système destiné à l'acquisition et au partage des expériences entre les membres d'un groupe de recherche. Dans cette application l'agent activités met à disposition de l'utilisateur un certain nombre de services de communication et d'activités courantes. Puis, cet agent fabrique une représentation XML des activités de l'utilisateur. Ces représentations sont ensuite gérées par l'agent profil qui utilise des techniques similaires à celle de l'agent bibliothèque pour indexer ces activités, fabriquer un modèle de préférences et ensuite proposer des informations lorsque l'utilisateur a un

problème. L'utilisateur peut alors ajouter ces propositions à son travail ou simplement les consulter. Les propositions faites à un utilisateur spécifique peuvent être récupérées à partir des bases d'activités d'autres utilisateurs. Par conséquent, le système permet le partage et la réutilisation des expériences.

7.2.2. Le filtrage et la classification de documents

Dans la section 4.3 nous avons présenté un SMA destiné au filtrage de documents dans un environnement réseau. Dans ce système, l'agent assistant (Agent Document) fournit une interface pour le stockage et la consultation des documents favoris de l'utilisateur. Ensuite l'agent bibliothèque fabrique un modèle de préférences à partir de ces documents. L'agent décision récupère auprès de ces agents les modèles individuels des utilisateurs et fabrique un modèle général pour l'ensemble des utilisateurs. Lorsqu'un document nouveau arrive l'agent décision identifie selon le modèle général, les modèles individuels et le modèle du document, quels sont les utilisateurs potentiellement intéressés par ce document. Par voie de conséquence, le système est capable de diminuer le trafic de données sur le réseau et d'éviter aux utilisateurs la lecture de documents inutiles.

7.2.3. La recherche personnalisée d'information

Le problème de recherche d'information a été présenté dans le chapitre précédent. Nous avons conçu un assistant (Agent Web) qui permet le déclenchement et l'analyse des résultats de recherches, et la gestion des sites favoris. L'utilisateur peut également choisir des centres d'intérêts publics concernant certains domaines pour améliorer la recherche. L'agent web communique ensuite avec l'un des agents de filtrage qui devra effectuer la recherche. Cet agent récupère un nombre de sites auprès des agents de recherche, les ordonne selon le modèle de préférences et finalement enregistre la réponse dans l'espace de travail de l'utilisateur.

7.3. Le déploiement

Nous avons discuté dans la section 7.1 de quelques architectures possibles lors de la conception d'un SMA personnalisé. Nous pensons que l'approche hybride comprenant un assistant personnel et des agents de service qui évoluent dans la machine de l'utilisateur et dans la coterie d'agents est la plus adéquate (section 7.1.5). Ainsi, si nous utilisons cette

architecture pour modéliser un SMA capable d'accommoder les applications de la section précédente nous aurons un système comme celui présenté sur la Figure 65.

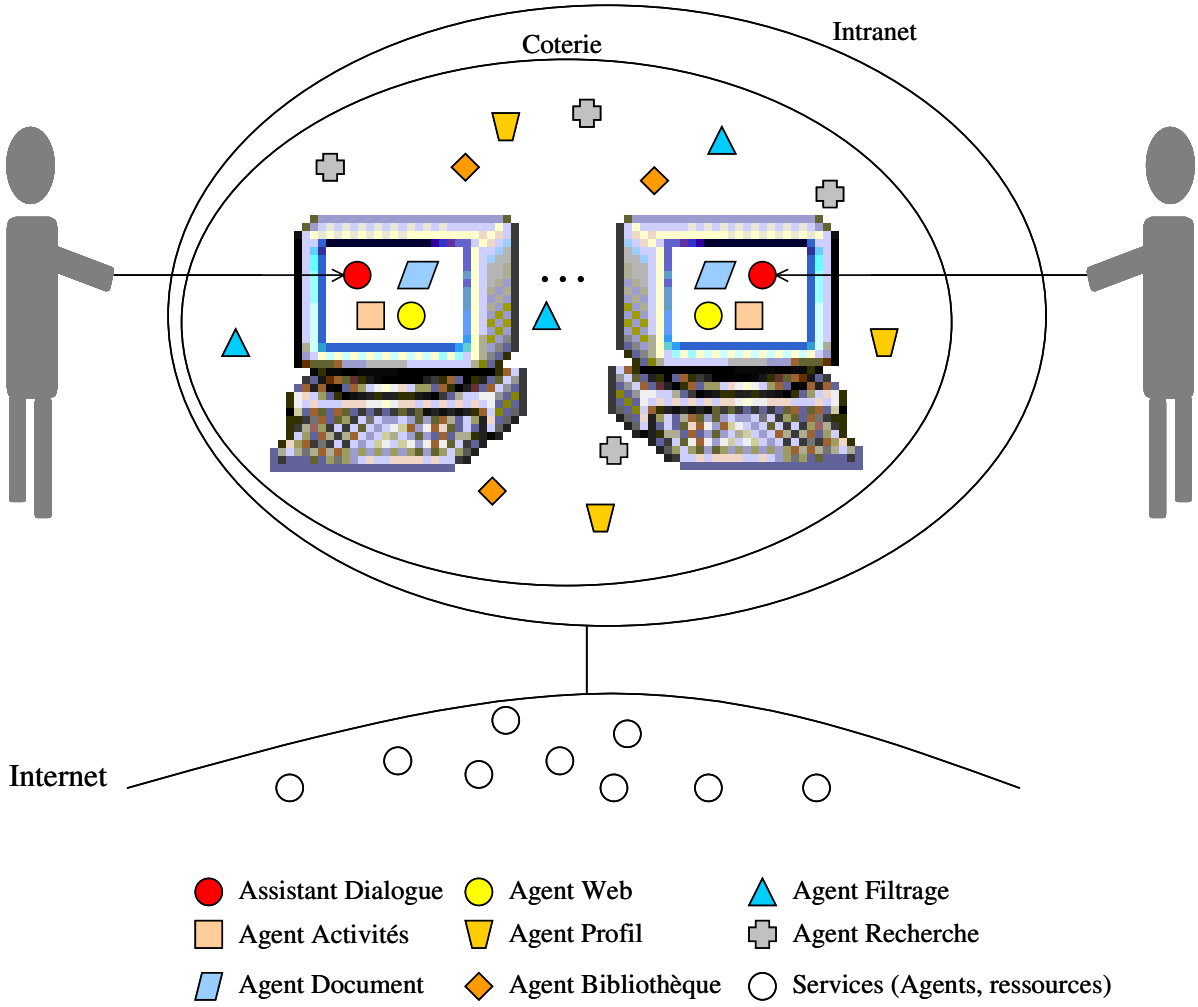

Figure 65 Un SMA personnalisé.

Sur la Figure 65, le staff de l'assistant est composé des agents suivants : *agent-activités*, *agent-document*, *agent-web*. Ces agents tournent sur la machine de l'utilisateur tandis que les agents service (*agent-profil*, *agent-décision*, *agent-bibliothèque*, *agents-filtrage*, *agents-recherche*) tournent dans une coterie et peuvent être partagés par d'autres SMAs.

L'architecture du SMA présenté sur la Figure 65 est adaptée à notre cas et peut être généralisée conceptuellement. Nous pensons que la définition d'un modèle générique peut favoriser la compréhension des mécanismes internes de notre architecture et faciliter le développement de nouvelles applications. Ainsi nous introduisons dans la section suivante un modèle formel de notre SMA.

7.4. Définition formelle d'un SMA personnalisé

Dans cette section nous présentons une représentation formelle d'un SMA personnalisé. Nous essayons d'abstraire toute donnée spécifique aux applications de façon que la représentation obtenue soit générique et donc utilisable dans n'importe quel domaine. Nous utilisons le système discuté dans la section 7.1 pour donner des explications à propos de la représentation et nous utilisons le langage *Z* (Spivey, 1992) pour déclarer les définitions nécessaires. Les définitions Z basiques sont présentées lorsqu'elles sont nécessaires.

7.4.1. Les agents du système

Nous avons soutenu au cours de cette thèse qu'un système destiné à la personnalisation de services ne doit pas être centralisé et composé par exemple, d'un seul assistant spécialiste. En revanche, nous soutenons que la personnalisation implique l'utilisation de mécanismes complexes d'interaction et création de modèles dynamiques. Cela peut être accompli plus facilement lorsque le système est décomposé en plusieurs agents qui peuvent être développés indépendamment les uns des autres.

Dans un SMA personnalisé nous pouvons distinguer trois types d'agents en fonction de la complexité de leurs compétences : *assistant* (compétences complexes), *coordination* (compétences moyennement complexes) et *service* (compétences simples).

L'*assistant* est responsable de l'interaction avec le maître. Il possède une représentation des tâches et des services que le système doit fournir. Il doit être spécialisé dans la communication avec le maître mais peut également faire appel aux services d'autres agents du système, normalement des agents de coordination.

Les agents de *coordination* sont spécialisés dans la résolution de problèmes complexes et qui font appel à plusieurs services. Pour cela, ils doivent également avoir une description des services qui précise comment chaque service peut être décomposé en sous-services et comment les requêtes concernant les sous-services doivent être envoyées. Pour personnaliser chaque service, les agents de coordination peuvent disposer d'un mécanisme d'adaptation spécifique à chaque type de service fourni. L'agent coordination doit ensuite utiliser les réponses aux sous-services pour produire la solution du problème initial demandé par l'assistant. Les sous-services sont réalisés par des agents *service* spécialisés à des problèmes simples. Chaque agent service contient un certain nombre de compétences (services) mises en

forme de procédures informatiques. Cette architecture ressemble beaucoup l'architecture des applications développées avec la plate-forme MAGIQUE (Bensaid et Mathieu, 1997), où les agents sont organisés de façon hiérarchique selon leurs rôles et compétences.

7.4.2. La définition formelle du système

Dans la définition présentée dans cette section nous utilisons des termes anglais pour garder une cohérence avec les éléments de notre système de dialogue discuté dans la section 3.3.5. Tout d'abord, nous avons besoin de quelques types de données basiques pour représenter les données du système. En *Z* ces éléments sont représentés de la manière suivante : [*AGENT*, *SERVICE-DESCRIPTION*, *AGENT-SERVICE*, *AGENT-IDENTIFICATION*, *TASK-STRUCTURE*, *SERVICE-INTERFACE*, *TERM*]. Nous présentons une définition de chacun de ces éléments dans le Tableau ci-dessons.

Type de Donnée	Définition
AGENT	Est un système informatique capable de fournir et/ou demander des services en utilisant un moyen de communication.
AGENT-SERVICE	Correspond à une compétence. Un agent peut avoir plusieurs compétences et donc être spécialisé dans plusieurs services.
AGENT-IDENTIFICATION	Identification d'un agent. Contient des informations comme le nom et l'adresse de l'agent
TASK-STRUCTURE	Est une structure utilisée par l'agent assistant pour représenter une tâche. Cette représentation joue un rôle similaire à l'AGENT-SERVICE car la tâche représente aussi un service fourni. Néanmoins les tâches concernent seulement l'agent assistant. Par ailleurs, la structure des tâches contient des informations additionnelles concernant les paramètres, par exemple, des questions pré-formulées. Ces informations sont utilisées par l'assistant pendant le dialogue avec l'utilisateur.
SERVICE-INTERFACE	Un service peut éventuellement avoir une interface utilisée par l'utilisateur pour saisir des informations et déclencher des

	services. L'assistant peut, par exemple, présenter des interfaces lorsqu'elles sont disponibles ou mener un dialogue en langage naturel.
TERM	Une chaîne de caractères.

Définition des types de données basiques.

Definition Z (Schémas). Un schéma Z sert à représenter des types abstraits de données et des fonctions, déclarer des variables ou encore définir les relations entre les variables et fonctions. Les schémas Z sont dénotés dans ce mémoire par un rectangle doté d'un nom.

Definition Z (Symbole $\mathbb{P}$). Dénote un ensemble.

Definition Z (Symbole $\mathbb{N}_1$). Dénote l'ensemble des nombres naturels à compter de 1.

Definition Z ($\mathbb{N}_1 \rightarrow$ *DATA-TYPE*). Dénote la déclaration d'un ensemble de données du type *DATA-TYPE* qui sont accessibles à partir d'une clé numérique. Du point de vue de l'implémentation, cela peut être mis en place sous forme d'un vecteur ou d'une liste.

Définition Z (Pré-conditions). Dans un schéma, une barre horizontale sépare les déclarations des pré-conditions. Les pré-conditions doivent être satisfaites pour la réalisation de l'opération.

Nous introduisons respectivement la définition formelle des trois types d'agents discutés dans la section précédente dans les schémas *AssistantAgent*, *CoordinationAgent* et *ServiceAgent*.

AssistantAgent

$name : TERM$
$agent\text{-}id : AGENT\text{-}IDENTIFICATION$
$service\text{-}descriptions : \mathbb{N}_1 \rightarrow SERVICE\text{-}DESCRIPTION$
$task\text{-}templates : \mathbb{N}_1 \rightarrow TaskTemplate$
$task\text{-}structures : \mathbb{N}_1 \rightarrow TASK\text{-}STRUCTURE$
$num\text{-}serv : \mathbb{N}$
$num\text{-}tasks : \mathbb{N}$
$run\text{-}task : TASK\text{-}STRUCTURE \nrightarrow ANSWER$
$run\text{-}service : SERVICE\text{-}DESCRIPTION \nrightarrow ANSWER$
$find\text{-}remote\text{-}service : (\mathbb{N}_1 \rightarrow TERM) \nrightarrow SERVICE\text{-}DESCRIPTION$

$task\text{-}structures = \text{dom } run\text{-}task$
$service\text{-}descriptions = \text{dom } run\text{-}service$
$service\text{-}descriptions = \text{ran } find\text{-}remote\text{-}service$
$\forall_i : 1.. num\text{-}tasks \bullet task\text{-}templates_i.name = task\text{-}structure_i.name$

Un agent assistant contient une série de descriptions de services, modèles de tâches et structures de tâches. Ces trois ensembles sont représentés respectivement avec les variables *service-descriptions*, *task-templates* et *task-structures*. Le nombre de services et de tâches est stocké respectivement dans les variables *num-serv* et *num-tasks*.

Dans le schéma *AssistantAgent* nous déclarons trois fonctions : *run-task*, *run-service* et *find-remote-service*. La fonction *run-task* reçoit comme paramètre une structure de tâche et doit rendre une réponse après avoir exécuté la tâche concernée. Le domaine de la fonction, c'est-à-dire, l'univers des tâches valides est donné par la variable *task-structure*. Cette fonction doit déclencher dans l'assistant un mécanisme qui permet de remplir la structure de tâche avec les informations nécessaires pour l'exécution de la fonction attachée à la tâche, par exemple, un mécanisme de dialogue en langage naturel comme nous avons vu dans la section 3.3. Lorsque les informations nécessaires ont été saisies l'assistant peut alors exécuter la fonction concernée, par exemple, envoyer un mail ou déclencher une application quelconque. La fonction *run-service* doit être utilisée pour déclencher l'exécution d'un service extérieur (qui appartient à des agents coordination). Ainsi, l'assistant n'est pas capable de réaliser le service tout seul et doit demander aux autres. Éventuellement, la description du service peut avoir une interface pour faciliter la saisie des informations nécessaires concernant le service. Nous avons ajouté aussi la fonction *find-remote-service*. Cette fonction doit identifier un certain service extérieur à partir d'une liste de termes. Les domaines des fonctions *run-task* et *run-service* correspondent respectivement au contenu des variables *task-structure* et *service-*

descriptions. La troisième pré-condition assure que le résultat de *find-remote-service* doit être un élément de *service-descriptions*. La quatrième pré-condition présentée dans le schéma *AssistantAgent* assure que les modèles et structures de tâches respectent le même ordre. La représentation des autres types d'agents (les schémas *CoordinationAgent* et *ServiceAgent*) est très simple et donc ne mérite pas une discussion plus approfondie.

CoordinationAgent

$name : TERM$
$agent\text{-}id : AGENT\text{-}IDENTIFICATION$
$service\text{-}descriptions : \mathbb{P}\, ServiceDescription$
$services : \mathbb{P}\, AGENT\text{-}SERVICE$

ServiceAgent

$name : TERM$
$agent\text{-}id : AGENT\text{-}IDENTIFICATION$
$services : \mathbb{P}\, AGENT\text{-}SERVICE$

ServiceDescription

$name : TERM$
$agent\text{-}id : AGENT\text{-}IDENTIFICATION$
$interface : SERVICE\text{-}INTERFACE$
$num\_params : \mathbb{N}$
$params : \mathbb{N}_1 \rightarrow TERM$
$verbs : \mathbb{N}_1 \rightarrow TERM$

Dans la définition de l'assistant et dans la définition des agents de coordination nous utilisons des descriptions de services (le schéma *Service-Description*). Un service normalement doit avoir un nom et une liste de paramètres nécessaires pour l'exécution du service. Il doit avoir également l'identification de l'agent capable de le réaliser et des informations additionnelles concernant chacun des paramètres. Dans notre spécification nous nous limitons à représenter des paramètres comme termes pour simplifier notre spécification. La variable *num_params* contient le nombre de paramètres du service.

SMAP

$assistant : AssistantAgent$
$coordination\text{-}agents : \mathbb{N}_1 \rightarrow CoordinationAgent$
$service\text{-}agents : \mathbb{N}_1 \rightarrow ServiceAgent$
$num\text{-}coord : \mathbb{N}$
$name : AGENT \nrightarrow TERM$

$\{assistant\} \cup coordination\text{-}agents \cup service\text{-}agents = \mathrm{dom}\ name$

À partir des structures déclarées précédemment nous pouvons finalement spécifier la structure générale du système. Pour cela nous introduisons le schéma *SMAP* (Système Multi-Agent Personnalisé). Un SMAP est composé par un agent assistant, un ensemble d'agents coordination et un ensemble d'agents de service. Nous déclarons aussi la variable *num-coord* qui contient le nombre d'agents coordination dans le système et la fonction *name*. La fonction *name* met en relation un agent et le terme qui représente son nom. Par définition, le domaine de la fonction *name* est constitué par tous les agents du système.

Si nous reprenons comme exemple le système de la section 7.1 nous avons le SMAP composé par les éléments suivants :

assistant = assistant-dialogue

coordination-agents = {agent-activités, agent-document, agent-web}

service-agents = {agent-profil, agent-décision, agent-bibliothèque, agent-filtrage$_1$, …, agent-filtrage$_m$, agent-recherche$_1$, …, agent-recherche$_n$}

num-coord = 3

Définition Z (Symbole Δ). Le symbole Δ dénote un changement de l'espace d'états d'une structure quelconque.

Définition Z (Opérateur ⊕). L'application de A ⊕ B produit un ensemble composé par les éléments de A avec des objets de B qui remplacent les versions des objets de A. C'est-à-dire, ⊕ met à jour A avec les éléments de B.

SMAP-AddCoordinationAgent

Δ *SMAP*
agent-id? : *AGENT-IDENTIFICATION*
service-descriptions? : $\mathbb{P}$ *ServiceDescription*

———————

$\forall_i$: 1.. *num_coord* • *agen-id?.name* ≠ *name*(*coordination-agents*(*i*))

num-coord' = *num-coord* + 1
coordination-agents' = *coordination-agents* ⊕ {*num-coord*' → *agent-id?*}
assistant.AssistantAgentAddService(*service-descriptions?*)

Du point de vue du système, une seule opération est possible : l'addition d'un agent coordination. Cette opération est spécifiée dans le schéma *SMAP-AddCoordinationAgent*. La fonction reçoit deux paramètres représentés par les variables *agent-id?* et *service-description?*. Ces deux variables concernent respectivement l'identification de l'agent à ajouter au système et la description du service qu'il fournit. La pré-condition dans le cas de *SMAP-AddCoordinationAgent*, consiste à assurer que le système ne possède pas un autre agent avec le même nom. Si la condition est vraie, le nombre d'agents coordination est augmenté, l'ensemble d'agents est mis à jour et l'assistant rajoute une nouvelle description de service dans sa base avec l'exécution de l'opération du schéma *AssistantAgentAddService*.

AssistantAgentAddService

Δ *AssistantAgent*
service-descriptions? : $\mathbb{P}$ *ServiceDescription*
num-serv! = $\mathbb{N}$

———————

$\forall$ *service-description*: *service-descriptions?*•
num-serv' = *num-serv* + 1
service-descriptions' = *service-descriptions* ⊕ {*num-serv*' → *service-description*}

Définition Z (Symbole Ξ). Le symbole Ξ dénote que la définition d'une opération ne change pas l'espace d'états de la structure concernée.

Du point de vue opérationnel, la fonctionnalité la plus importante est l'interprétation des requêtes de l'utilisateur. Le schéma *AssistantAgentExecuteQuery* montre le déclenchement des actions de l'agent assistant. Cette fonction reçoit comme paramètre le terme central de la requête et les termes concernant les paramètres du service ou de la tâche et le verbe utilisé. Par exemple, à partir de la requête « *send a mail to cesar* » nous identifions le terme central de la tâche « mail », le paramètre « to » et le verbe « send ». La décomposition grammaticale des phrases des utilisateurs ne fait pas partie de la spécification générique du système car elle

concerne l'aspect implémentation de celui-ci. Pour cela nous ne la discutons pas dans cette section. Cependant, nous avons étudié ce sujet dans la section 3.3.3.

AssistantAgentExecuteQuery

Ξ *AssistantAgent*
target-term? : *TERM*
verb? : *TERM*
params? : $\mathbb{N}_1 \rightarrow$ *TERM*
answer! : *ANSWER*
task-template! : *TaskTemplate*
task-structure! : *TASK-STRUCTURE*
service-description! : *ServiceDescription*

———————

((*task-template!*, *task-structure!*) = *AssistantAgentFindLocalTask*(*target-term?*, *verb?*, *params?*)
$\wedge$ *answer!* = *run-task* (*task-template!*, *task-structure!*))
$\vee$
(*service-description!* = *find-remote-service*({*target-term?*, *verb?*} $\cup$ *params?*)
$\wedge$ *answer!* = *run-service* (*service-description!*))

Avant de donner plus de détails sur le traitement de la requête de l'utilisateur nous introduisons le schéma *TaskTemplate*.

TaskTemplate

names : $\mathbb{N}_1 \rightarrow$ *TERM*
verbs : $\mathbb{N}_1 \rightarrow$ *TERM*
params : $\mathbb{N}_1 \rightarrow (\mathbb{N}_1 \rightarrow$ *TERM*)
num_params : $\mathbb{N}$
find-parameter : *TERM* $\nrightarrow$ {t, f}

———————

$\frown$/*params* = dom *find-parameter*

Une structure du type *TaskTemplate* contient des informations terminologiques concernant une tâche. Nous avons utilisé cette structure dans la conception du notre système de dialogue dans la section 3.3. La variable *names* correspond aux termes qui concernent la tâche, par exemple, pour la tâche « envoi de mails », *names* peut être {mail, email, e-mail, message, courrier}. La variable *verbs* contient les verbes utilisés pour déclencher l'action, par exemple {send, write}. La variable *params* correspond aux termes concernant les paramètres, par exemple, {{to, receiver}, {carbon-copy, cc, copy}, {subject, comments}, {text, contents, message}}. Ainsi, chaque paramètre peut-être référencé par un nombre de termes. La variable *num_params* contient le nombre de paramètres de la tâche et *find-parameter* est une fonction booléenne qui calcule si un terme fait référence à un paramètre. Dans le schéma *TaskTemplate*

le symbole ⌢/ représente la concaténation de tous les éléments d'une séquence, dans l'exemple précédant par exemple, ⌢/*params* = {to, receiver, carbon-copy, cc, copy, subject, comments, text, contents, message}. Le type *TaskTemplate* est utilisé dans la définition du schéma *AssistantAgentExecuteQuery* pour l'exécution d'une requête utilisateur.

Les paramètres du schéma *AssistantAgentExecuteQuery* (*target-term?*, *verb?*, *params?*) correspondent respectivement au terme central de la requête utilisateur, le verbe et la liste de paramètres référencés dans la requête. Nous utilisons quatre variables additionnelles : *answer!*, *task-template!*, *task-structure!* et *service-description!*. À partir des paramètres fournis l'assistant essaye de trouver la tâche concernée. Si une tâche est trouvée, alors il l'exécute. Sinon, il essaye de trouver un service extérieur (un service d'un agent coordination). Si un service est trouvé, alors il démarre l'exécution du service.

Le schéma *AssistantAgentFindLocalTask* montre comment l'assistant peut localiser une tâche qu'il est capable d'exécuter. La fonction reçoit trois paramètres (*name?*, *verb?* et *params?*). Nous utilisons une variable additionnelle : *value!*. Le résultat de la fonction est une paire ordonnée qui contient le modèle d'une tâche et la structure de cette tâche.

AssistantAgentFindLocalTask

Ξ *AssistantAgent*
name? : *TERM*
verb? : *TERM*
params? : $\mathbb{N}_1 \rightarrow$ *TERM*
value! : *(TaskTemplate, TASK-STRUCTURE)*

$\exists_i : 1..$ *num_tasks* • (*name?* $\in$ *task-templates*(*i*).*names*) $\wedge$ (*verb?* $\in$ *task-templates*(*i*).*verbs*)
$\forall$ *m* $\rightarrow$ *param* : *params?* • *task-templates*(i).*find-parameter*(*param*)

value! = (*task-template*(*i*), *task-structure*(*i*))

Nous pouvons constater que notre modèle formel favorise la distribution de services mais ne formalise pas l'interaction entre agents. Nous nous limitons ici au formalisme des structures internes de chaque agent, car nous pensons que les modèles d'interaction peuvent varier énormement selon le but recherché, comme la résolution distribuée de problème ou la simple distribution de tâches avec le Contract Net (le lecteur intéressé sur la spécification formelle du Contract Net peut consulter le chapitre VI de l'ouvrage de d'Inverno et Luck (2001)). Ainsi, le modèle d'interaction est spécifique au comportement souhaité.

Nous pensons que notre formalisme peut être utilisé pour le développement de SMA destinés à la personnalisation de services sur l'Internet. Aujourd'hui, une grande attention est donnée aux systèmes composés d'agents d'information qui favorisent la personnalisation en plusieurs domaines d'application. Nous discutons dans la section suivante dans quel contexte notre modèle de SMA peut s'inscrire et comparons notre approche à des travaux similaires.

7.5. Services distribués vs résolution distribuée de problèmes

Au début de ce mémoire nous avons dit comment des agents pouvaient résoudre de problèmes de façon distribuée. Ashri et al. 2003, prétendent que l'interaction entre agents peut être utile pour la résolution de trois catégories de problèmes : la *coordination* (les agents réalisent leurs activités individuelles de façon cohérente), la *collaboration* (les agents travaillent ensemble pour accomplir un objectif commun) et la *compétition* (les agents doivent partager des ressources limitées). Néanmoins, ces comportements sont souvent ignorés dans plusieurs applications, principalement celles concernant la toile.

Aujourd'hui, de nombreux travaux concernant les *agents d'information* ou « *gathering agents* » portent sur comment rendre disponibles des informations intéressantes pour des utilisateurs connectés à l'Internet. Le but est de concevoir des outils informatiques auxquels on peut accéder de n'importe quel endroit du réseau (ou du monde), qui soient capables de suivre le contexte d'un utilisateur et ainsi de fournir des informations utiles et personnalisées. Les applications « ACCESS » et « GraniteNights », présentées au workshop CIA'03 (Cooperative Information Agents) à Helsinki illustrent bien cette approche.

ACCESS (Muldoon et al., 2003) est une application multi-agents construite à partir de la plate-forme *Agent Factory*. Elle a été conçue pour aider une personne à se déplacer dans une ville inconnue. L'utilisateur arrivant dans une ville inconnue peut sur son PDA (Personnal Digital Assistant) consulter la carte numérique de la ville, vérifier sa position (capturée automatiquement par GPS), demander l'adresse d'un hôtel, visualiser le chemin à suivre, les taxis disponibles ou encore la position exacte des bus qu'il peut utiliser pour y arriver.

Le SMA GraniteNights (Grimnes et al., 2003) a été conçu pour organiser la soirée des utilisateurs. Le système met à disposition une interface web grâce à laquelle l'utilisateur qui veut organiser une soirée peut demander trois types de divertissements : restaurant, cinéma ou « pub ». Le système (qui connaît la localisation et a des informations sur les soirées

précédentes) fait des propositions. Comme le système ne dispose d'aucune méthode d'apprentissage du profil de l'utilisateur, les propositions sont simplement des listes de combinaisons possibles construites à partir de bases de données stockées sur l'agent restaurant, l'agent cinéma et l'agent pub. La plate-forme JADE a été utilisée pour la construction des agents.

Du point de vue de l'Intelligence Artificielle, ces applications n'ont rien d'intelligent. Elles reposent sur une énorme quantité de données stockées dans des agents qui simplement répondent à des demandes de service et ne sont pas capables de coopérer ni de s'adapter. Les données sont structurées sous formes d'ontologies qui peuvent ensuite être exploitées avec des langages de représentation comme RDF ou RDF Schéma.

Certaines applications web, moins nombreuses utilisent les concepts de résolution distribuée de problème. TAC (Trade Agent Competition) (TAC, 2002), par exemple, met en compétition plusieurs agents qui doivent organiser le voyage d'utilisateurs différents en achetant les billets d'avion, en réservant l'hôtel et en choisissant des options de divertissement, tout en essayant de minimiser les dépenses. Freitas et Bittencourt (2003) utilisent la collaboration entre agents pour rechercher des informations spécifiques sur la toile. Les agents ont des connaissances (patrons de pages, règles, et fonctions) sur des domaines spécifiques et sont capables de faire des propositions sur ces pages. Lorsqu'une recommandation d'un agent est acceptée, les autres agents rajoutent à leur connaissance les règles et concepts qui lui ont permis de faire cette recommandation. Par conséquent, les agents coopèrent pour résoudre le problème. Les agents sont utilisés pour la recherche d'appels à participation à des conférences et pour obtenir des articles scientifiques.

Liu (2003) prétend que dans un avenir proche, chaque utilisateur aura à sa disposition une communauté d'agents capables de fournir de nombreux services. Ce concept est similaire à celui d'un staff d'agents. Liu discute deux types d'agents : les *agents communauté* capables de planifier et distribuer les tâches et les *agents service* qui réalisent effectivement les tâches. Nous ne pouvons pas laisser passer inaperçue ici la similarité entre ces concepts et nos définitions d'*agent coordination* et d'*agents service* introduits dans les sections précédentes. Liu soutient également que les agents service devront être forcement coopératifs pour fournir des services intelligents et améliorer leur performance en échangeant des informations concernant le contexte de l'utilisateur.

Cependant, Liu discute les modes d'interaction entre l'utilisateur et le système d'agents très vaguement en disant seulement que le système devra être accessible à partir de plusieurs outils, comme Palms, notebooks, téléphones mobiles ou ordinateurs communs. À l'opposé de cette approche, nous favorisons l'utilisation d'un agent assistant personnalisé.

Concernant l'architecture de SMA personnalisé présenté dans la section 7.2, nous sommes conscients que nos agents ne sont pas très coopératifs et que le mécanisme de coordination est codé en dur dans les agents de filtrage. Cependant, nos tâches sont très spécifiques et ne sont pas vraiment propices à une résolution distribuée de problèmes, car les activités de chaque agent ne sont pas vraiment liées.

Enfin, ces agents devront partager des formes standard de représentation de connaissances et des formes de représentation de la sémantique des informations disponibles sur la toile utilisant des langages de marques (markup languages). Par conséquent, nous pouvons affirmer que le développement des agents intelligents sur la toile s'appuiera fortement sur les nouvelles technologies du « web sémantique » favorisant « l'intéroperabilité sémantique ».

7.6. La problématique et la solution apportée

Dans cette section nous analysons les mécanismes proposés tout au long de la thèse en fonction de la problématique concernant l'architecture proposée par Ramos.

En introduisant le modèle de Ramos nous avons mis en évidence trois problèmes :

- Les modules sont décrits seulement au niveau conceptuel ;
- L'interaction avec l'utilisateur n'est pas clairement définie ;
- L'architecture peut rencontrer des problèmes lors d'un passage à l'échelle.

Nous pensons que les deux premières questions sont clairement résolues avec les structures décrites par le modèle formel de la section 7.4 et le mécanisme de contrôle de dialogue de la section 3.3.4, respectivement. Concernant la troisième question nous pensons que le modèle distribué formalisé dans ce chapitre peut diminuer considérablement le nombre de calculs réalisés et la complexité de l'agent assistant personnel. Par ailleurs, cette distribution favorise la gestion des services qui peuvent être rajoutés ou supprimés du système sans nécessiter de changements concernant le mode d'interaction avec l'utilisateur. Si l'on reprend l'architecture

de Ramos nous aurons une nouvelle représentation des modèles internes de l'agent assistant personnel. Cette représentation est illustrée sur la Figure 66.

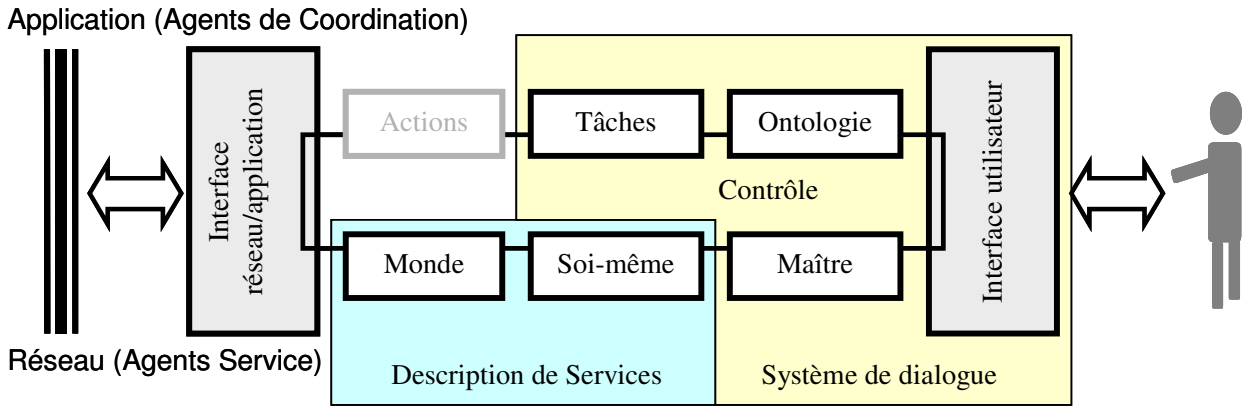

Figure 66 Architecture résultante d'agent assistant personnel.

Les modules d'ontologie, tâches et maître sont gérés par le système de dialogue. Dans notre cas, le modèle du maître illustré sur la Figure 66 concerne seulement celui de l'interaction. En revanche, les modèles de l'utilisateur spécifiques aux autres activités fournies par le système sont répartis selon les agents coordination du système. Les modèles du monde et soi-même contiennent l'identification des agents coordination ainsi que leurs descriptions de services. Puisque notre assistant ne fournit, à ce jour, pas de services à d'autres agents, nous n'utilisons pas la représentation des compétences (Actions). Cependant, rien n'empêche qu'une approche collaborative comme celle de (Lashkari et Maes, 1994) ne soit développée et donc, des actions soient construites. Dans l'application de Lashkari et Maes, les agents assistants collaborent pour gérer les mails d'un groupe de personnes à travers l'exécution de services.

Dans notre modèle nous avons besoin de distinguer le mode de communication entre *application* et *réseau*. Puisque notre agent assistant personnel communique seulement avec des agents de coordination et que ces agents tournent sur la machine de l'utilisateur, nous considérons que l'utilisation du réseau doit se limiter à la communication entre les agents coordination et les agents de service. La communication entre l'assistant et les agents coordination peut se faire sans utiliser le réseau si les agents évoluent dans le même environnement, par exemple dans la même Image Lisp.

7.7. Conclusions

Dans cette section nous avons présenté les architectures possibles des SMA qui peuvent servir à définir des agents assistants. Nous avons vu également, comment un SMA personnalisé peut

être mis en place pour aider l'utilisateur. Nous avons mis ensemble les différentes applications développées au cours de cette thèse sous forme d'une seul SMA. Pour concevoir le SMA nous nous sommes appuyés sur la métaphore du majordome. Selon cette métaphore, l'utilisateur interagit avec un seul agent assistant, capable de mener un dialogue convenable pour identifier ses besoins et gérer un groupe d'agents de coordination, sous la forme d'agents personnalisés adaptatifs.

Dans le SMA obtenu, les agents capables de personnaliser les services, c'est-à-dire, les agents adaptatifs, ont été groupés sous forme d'un staff d'agents qui tournent sur la machine de l'utilisateur, tandis que des agents de service simples sont distribués dans la coterie. Ensuite nous avons développé une représentation formelle du système. Nous nous sommes concentrés sur la formalisation des mécanismes internes de chaque agent et nous n'avons pas formalisé le mécanisme de négociation entre agents, car celui-ci peut être développé avec des protocoles simples comme par exemple, le Contract Net.

Cette formalisation sert à renforcer l'un des arguments présentés au cours de cette thèse qui avance l'idée que dans des domaines complexes, l'assistance ne peut pas être accomplie avec un seul agent assistant isolé car ce dernier devient à son tour trop complexe. En revanche, un SMA peut être utilisé pour personnaliser les services en distribuant ceux-ci à d'autres agents.

8 Conclusions Générales

Dans cette thèse nous avons étudié principalement deux problèmes :

- Le développement de l'architecture proposée par Ramos pour la construction d'un agent assistant personnel ;
- Le manque de méthodes d'apprentissage adaptées à la conception d'agents assistants personnels adaptatifs.

Nous avons introduit dans ce mémoire une architecture de SMA destinée à personnaliser les services et l'interaction avec l'utilisateur. Nous avons présenté et discuté plusieurs applications destinées à différentes sortes de services, et constaté que même un service apparemment simple (comme la recherche sur le web) pouvait nécessiter plusieurs mécanismes complexes. En conséquence, il est difficile d'imaginer comment un seul agent assistant pourrait accommoder les nombreux services que lui sont nécessaires et dont certains contiennent des mécanismes complexes. Ce problème limite l'architecture proposée par Ramos à des domaines simples, susceptibles d'être traités par une petite collection statique de services simples. Notre conclusion est qu'un SMA composé de l'agent assistant personnel et d'un staff d'agents de service est indispensable pour la distribution des services et la mise en œuvre de mécanismes plus sophistiqués, ce qui simplifie le développement d'un agent assistant personnel dont le role est alors restreint à la personnalisation de l'interaction avec l'utilisateur.

Par ailleurs, l'assistant de Ramos ne peut pas gérer automatiquement des services dynamiques qui sont rajoutés ou supprimés au fil de l'eau, car le dialogue avec l'utilisateur doit être

recodé à chaque départ ou arrivée. Nous avons donc ajouté, à notre SMA, un module de dialogue générique capable de s'adapter dynamiquement à différents services.

L'évolution d'un agent assistant personnel sert à améliorer l'interaction avec l'utilisateur et à personnaliser les résultats des services. Nous avons pu constater que l'adaptation de l'interaction est un problème complexe. Concernant les modes classiques d'interaction, l'adaptation du point de vue système est souvent négligée à cause de la complexité des connaissances nécessaires à la création des modèles initiaux de l'utilisateur. Lorsque l'adaptation de l'interaction est effectivement mise en place, elle utilise des techniques de modélisation automatique de l'utilisateur. Nous avons pu constater également, que l'adaptation de l'interaction à partir du langage naturel et de systèmes de dialogue est encore plus difficile, car les mécanismes d'apprentissage actuels ne sont pas capables d'apprendre convenablement à partir d'informations non-structurées ou faiblement structurées.

Nous avons pu constater que les mécanismes d'adaptation des agents autonomes adaptatifs ne sont pas tout à fait adaptés à la conception d'un agent assistant personnel. Nous avons donc proposé une nouvelle méthode capable de résoudre ce problème. La méthode incrémentale d'apprentissage présentée dans ce mémoire (ELA) est performante, nécessite peu de calculs pour classer des exemples et peut apprendre à partir de données peu nombreuses. D'autres avantages de la méthode proposée sont la capacité de prise en compte immédiate de feedback, ainsi que la capacité d'utilisation de modèles définis au préalable. Ces techniques ont été utilisées dans une application de recherche d'information sur la toile, où l'utilisateur peut ajouter de nouvelles pages aux favoris (feedback) et utiliser des centres d'intérêts publics (modèles définis au préalable). Ces caractéristiques rendrent la méthode générique et adaptée à la conception d'agents assistants personnels.

Enfin, nous avons conçu une représentation générique d'un système multi-agent personnalisé (SMAP) qui résout les problèmes inhérents à l'architecture centralisée de Ramos et conserve la personnalisation au niveau de l'interaction et de l'exécution des services.

Les perspectives de travaux futurs

Concernant la méthode d'apprentissage développée, nous pouvons dire que la procédure de mesure de performance ne permet pas de caractériser entièrement son potentiel ni ses points faibles. Pour avoir une idée plus précise du comportement de la méthode, il faudra réaliser beaucoup plus d'analyses à partir d'autres bases de données, y compris sur des bases non

symboliques. Cela n'a pas été fait dans ce travail, car l'analyse de performance sur des données symboliques était suffisante pour nos problèmes.

Une autre perspective de recherche consiste à adapter la méthode ELA à des données non-symboliques. Dans ce cas, la méthode devra incorporer une discrétisation dynamique de données pour transformer les variables quantitatives en variables qualitatives. Pour cela, des méthodes de groupement supervisé ou non-supervisé ou d'autres méthodes utilisées dans des algorithmes d'arbres de décision pourront être considérées.

Une autre voie de recherche concerne le développement d'une architecture d'agent autonome à partir des études réalisées. Je compte construire une architecture d'agent autonome suffisamment générique pour pouvoir tester et développer des méthodes de résolution distribuée de problèmes. Ces agents devront collaborer et seront appliqués à résolution de plusieurs problèmes, comme par exemple, la simulation de traffic routier. À partir d'une architecture fiable d'agent autonome, j'espère rendre les agents adaptatifs en utilisant des méthodes d'apprentissage comme celle qui a été présentée dans ce mémoire. Le but sera d'évaluer dans quelle mesure l'adaptation peut aider un agent autonome dans un environnement collaboratif.

Finalement, je compte continuer à développer des applications d'agents assistants personnels, notamment celles concernant le web comme par exemple, la recherche distribuée d'information, la gestion automatisée de documents (organisation, filtrage et classification) et les SMA personnalisés.

Références bibliographiques

AAMODT, A. ; PLAZA, E., *Case-Based Reasoning : Foundational Issues, Methodological Variations, and System Approaches*, AICom - Artificial Intelligence Communications, IOS Press, Vol. 7: 1, pp. 39-59, 1994.

ABEL, M., *Diffusion de connaissances et dialogue*, Technique et sciences informatiques, vol X, nº X/2000, pp Y-Z.

AGRAWAL, R.; SRIKANT, R., *Mining Sequential Patterns*, Proceedings of the 11th International Conference on Data Engineering – ICDE, IEEE Press, Philip S. Yu and Arbee L. P. Chen (eds.), 6 October, 3-14 pg., 1995. ISBN 0-8186-6910-1

AHA, D. W.; KIBLER, D.; ALBERT, M.K., *Instance-Based Learning Algorithms*, Machine Learning, 6, pp. 37-66, 1991.

ALLEN, J. F., *Natural Language Understanding*, The Benjamin/Cummings Publishing Company, Inc, Menlo Park, California, 1986. ISBN 0-8053-0330-8

ALLEN, J. F.; Miller, B. W.; Ringger, E. K.; Sikorski, T., *Robust Understanding in a Dialogue System*, Proc. 34 th. Meeting of the Association for Computational Linguistics, June 1996.

Agentcities Task Force, http://www.agentcities.org.

ARDISSONO, L., FELFERNIG, A., FRIEDRICH, G., GOY, A., JANNACH, D., MEYER, M., PETRONE, G., SCHÄFER, R., SCHÜTZ, W., ZANKER, M., *Personalising On-Line Configuration of Products and Services*, ECAI'02, F. van Harmelen (ed.), IOS Press, pp.225-229,2002. ISBN 1-58603-257-7.

ARMSTRONG, R; FREITAG, D.; JOACHIMS, T.; MITCHELL, T., *WebWatcher: A Learning Apprentice for the World Wide Web*, AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments, march, 1995.

ASHRI, R., LUCK, M., D'INVERNO, M., On Identifying and Managing Relationships in Multi-Agent Systems, Proceedings of IJCAI'03, Morgan Kaufmann Publishers, pp. 743-748, Georg Gottlob and Toby Walsh (eds.), 2003. ISBN 0-127-05661-0

de AZEVEDO, H., *Contribution à la modélisation des connaissances à l'aide des systèmes multi-agents*. Thèse de Doctorat, Université de Technologie de Compiègne, juin, 1997.

BARTHES, J-P. A., *MOSS 3.2*, Memo UTC/GI/DI/N 111, Université de Technologie de Compiègne, Mars, 1994.

BARTHES, J-P. A., *OMAS v 1.0 Technical Reference*, Memo UTC/GI/DI/N 151, Université de Technologie de Compiègne, Département de Génie Informatique, Jannuary, 2002.

BARTHÈS J-P. A, TACLA C. *Agent-supported portals and knowledge management in complex R&D projects*. Computers in Industry. **48**, 3-16, 2002.

BAY, S. D.; PAZZANI, M. J., Detecting Change in Categorical Data: Mining Contrast Sets, Knowledge Discovery and Data Mining, pp. 302-306, 1999.

BENSAID N., MATHIEU, P., *A Hybrid and Hierarchical Multi-Agent Architecture Model*, Proceedings of the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'97), 1997.

BENSON S.: *Inductive Learning of Reactive Action Models*. ICML'95, pp. 47-54, 1995.

BILLSUS, D.; PAZZANI, M., *A Hybrid User Model for News Story Classification,* Proceedings of the Seventh International Conference on User Modeling (UM '99), Banff, Canada.

BLAKE, C.L.; MERZ, C.J., *UCI Repository of machine learning databases* [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science, 1998.

BOISSIER, O., *Modèles et Architectures d'agents*, In : Briot, J-P.; Demazeau, Y. (eds.), Principes et architecture des systèmes multi-agents, Lavoisier, 2001. ISBN 2-7462-0336-7.

BOOKER, L. B.; GOLDBERG, D. E.; HOLLAND, J. H., *Classifier systems and genetic algorithms*. Artificial Intelligence, n. 40, pp. 235-282, 1989.

De BRA, P. ; BRUSILOVSKY, P. ; HOUBEN, G-J., *Adaptive Hypermedia : From Systems to Framework*, ACM Computing Surveys, n. 31, vol 4, December 1999.

BRATMAN, M.E.; ISRAEL, D.J.; POLLACK, M.E., *Plans and Resource-Bounded Practical Reasoning*, Computational Intelligence, vol. 4, pg. 349-355, 1988.

BRENNER, W.; ZARNEKOW, R.; WITTIG, H., *Intelligent Software Agents*, Springer-Verlag, 1998. ISBN 3-540-63411-8

BRIFFAULT, X. ; GUESSOUM Z. ; OCCELLO M., *Environnement de développement de systèmes multi-agents*, Technique et Science Informatiques RSTI, série TSI, vol.21, nº 4, 2002.

BRIGHTON, H.; MELLISH, C., *Advances in Instance Selection for Instance-Based Learning Algorithms*, Data Mining and Knowledge Discovery, Kluwer Academic Publishers, n. 6, pp. 153-172, 2002.

BROOKS, R.A., *A Robust Layered Control System for a Mobile Robot*, IEEE Journal of Robotics and Automation, RA-2, April, 1986.

BROOKS, R.A., *Intelligence without Reason*, Computers and Thought lecture, Proceedings of IJCAI-91, Sidney, Australia, 1991.

BRUSILOVSKY, P.; SU, H.-D. *Adaptive Visualization Component of a Distributed Web-based Adaptive Educational System*. In: Intelligent Tutoring Systems. Vol. 2363, (Proceedings of 6th International Conference on Intelligent Tutoring Systems, ITS'2002, Biarritz, France, June 2-7, 2002) Berlin: Springer-Verlag, pp. 229-238, 2002.

BUENO, D.; DAVID, A. A., *METIORE: A Personalized Information Retrieval System*, Proceedings of the 8th International Conference User Modeling 2001, Springer-Verlag, Sonthofen, Germany, July, 2001. pp 168-177.

CALVO, R. A., *Procesamiento Inteligente de Documentos*, Tutorial of the Simposio Argentino de Inteligencia Artificial ASAI'00, Tandil, Argentina, 4-9 septiembre, 2000.

CALLAN, J., *Distributed information retrieval*. In W.B. Croft (ed.), *Advances in information retrieval*, chapter 5, pages 127-150. Kluwer Academic Publishers, 2000.

CLARK, K. L.; LAZAROU, V. S., *A Multi-Agent System for Distributed Information Retrieval on the World Wide Web*. In WETICE97, Collaborative Agents in Distributed Web Applications, IEEE Computer Society Press, 1997.

CRABTREE, I. B., *Adaptive Personal Agents*, In: Personal Technologies Journal, n. 3, vol. 2, pp. 141-151, 1998.

CHAPELLE, J.; Simonin, O.; Ferber, J., *How Situated Agents can Learn to Cooperate by Monitoring their Neighbors' Satisfaction*, ECAI'02, F. van Harmelen (ed.), IOS Press, pp.68-72, 2002. ISBN 1-58603-257-7

CHAN LAM, V., *Conception et réalisation d'une base de données ensembliste*, Thèse de Doctorat, Université de Technologie de Compiègne, 1979.

CHEN, L ; SYCARA, K., *Webmate: A Personal Agent for Browsing and Searching*, Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98), ACM Press, (K. Sycara and M. Woodridge eds.), pp. 132-139, New York, 1998.

CORCHUELO, R. ; ARJONA, J.L. ; RUIZ, A., *Automatic Extraction of Semantically-Meaningful Information from the Web*, Upgrade The European Online Magazine for the IT Professional, (Ricardo Bayeza-Yates, Peter Schäuble eds.), n. 3, vol III, June 2002.

CUNNINGHAM, P.; SMYTH, B.; VEALE, T., *On the limitations of Memory Based Reasoning*, In: Proceedings of the 2nd European Workshop on Case-Based Reasoning, 1994, Paris.

DELGADO, J.; ISHII, N.; URA, T., *Intelligent Collaborative Information Retrieval*, IBERAMIA, pp 170-182, 1998. http://citeseer.nj.nec.com/delgado98intelligent.html

De VELL, O.; NESBITT, S.A., *Collaborative Filtering Agent System for Dynamic Virtual Communities on the Web, Working Notes of Learning from Text and the Web*, Conf. Automated Learning and Discovery (CONALD-98), Carnegie Mellon University, Pittsburg, 1998. http://www.cs.cmu.edu/~conald/

DIMAKOPOULOS, V. V., PITOURA, E., *A Peer-to-Peer Approach to ressource Discovery in Multi-Agent Systems*, Cooperative Information Agents VII, Springer-Verlag, LNAI 2782, M. Klusch, A. Omicini, S. Ossowski, H. Laamanen (eds.), pp. 62-77, Helsinki, August, 2003. ISBN 3-540-40798-7

DORER, K., *Behavior Networks for Continuous Domains using Situation-Dependent Motivations*, Proceedings of the IJCAI'99, pg. 1233-1238, 1999.

http://citeseer.nj.nec.com/dorer99behavior.html

EKLUND, P. W.; HOANG, A., *A Performance Survey of Public Domain Supervised Machine Learning Algorithms*, Technical Report. http://www.kvocentral.com/kvopapers/7paper.pdf

ELZER, S.; CHU-CARROLL, J.; CARBERRY, S., *Recognizing and Utilizing User Preferences in Collaborative Consultation Dialogues*, In UM 94 Proceedings of the Fourth International Conference on User Modeling, Hyannis, Massachusetts, 1994, pp 19-24. http://citeseer.nj.nec.com/elzer94recognizing.html

Encyclopaedia Universalis France S/A, vol 1, pg. 255, 1989.

ENEMBRECK, F.; BARTHES, J-P., *Communication entre OMAS et d'autres plates-formes d'agents; le concept d'agent postier*. Rapport Technique HEUDIASYC 2001/45, Université de Technologie de Compiègne, 2001.

ENEMBRECK, F.; BARTHÈS, J-P., *Document Classification in Many-Classes Environments*, Proceedings of The International Symposium on Document Technologies - STD, Kival Weber and Celso Kaestner (eds.), pg. 31-36, 17-19 September, São Paulo, Brazil, 2002a.

ENEMBRECK, F.; BARTHÈS, J-P. , *Personnel Assistant to improve CSCW*, Proceedings of The Seventh International Conference on CSCW in Design, pg. 329-335, COPPE/UFRJ, 25-27 September, Rio de Janeiro, Brazil, 2002. ISBN 85-285-0050-0

ENEMBRECK, F.; BARTHÈS, J-P., *Agents for Collaborative Filtering*, Cooperative Information Agents VII, Springer-Verlag, LNAI 2782, M. Klusch, A. Omicini, S. Ossowski, H. Laamanen (eds.), pp. 184-191, Helsinki, August, 2003. ISBN 3-540-40798-7

ERMAN, L. D., LESSER, V. R., *A multi-level organization for problem solving using many diverse cooperating sources of knowledge*, Proceedings of the 4th International Joint Conference on Artificial Intelligence, Tbilisi, USSR, pp. 483-490, 1975.

ETZIONI, O., WELD, D., *A Softbot-Based Interface to the Internet*, In :Readings in Agents, M. N. Huhns et M. P. Singh (eds.), Morgan Kaufmann Publishers Inc., San Francisco, EUA, 1998. ISBN 1-55860-495-2

FALTINGS, B., *Qualitative Models as Indices for Memory-Based Prediction*, IEEE Expert, pp. 47-53, may-june, 1997.

FERBER, J., *Les Systèmes Multi-Agents*, InterEditions, Paris, 1995. ISBN 2-7296-0572-X

FLYCHT-ERIKSSON, A., *A Survey of Knowledge Sources in Dialogue Systems*, Proceedings of the (IJCAI)-99 Workshop on Knowledge and Reasoning in Practical Dialogue Systems, International Joint Conference on Artificial Intelligence, Murray Hill, New Jersey, Jan Alexandersson (ed.), pp 41-48, 1999. http://citeseer.nj.nec.com/328028.html

FIKES, R.E.; NILSSON, N.J., *STRIPS: a New Approach to the Application of Theorem Proving to Problem Solving*, Artificial Intelligence, n. 2, vols 3-4, pp. 189-208, 1971.

FIPA, *Foundation for Intelligent Physical Agents*, http://www.fipa.org.

FIPAa, *Interaction Protocol Library Specification*, http://www.fipa.org/specs/fipa00025/PC00025C.html.

FIPAb, *Agent Management Specification*, http://www.fipa.org/specs/fipa00023/XC00023G.pdf.

FIPAc, FIPA *Ontology Service Specificatio*n, http://www.fipa.org/specs/fipa00086/XC00086D.pdf.

FRANKLIN, S., *Autonomous Agents as Embodied AI*, Cybernetcs and Systems, n. 28, vol 6, pg. 499-520, 1997.

FREITAS, F. L. G., BITTENCOURT, G., *An Ontology-based Architecture for Cooperative Information Agents*, Proceedings of IJCAI'03, Morgan Kaufmann Publishers, pp. 37-42, Georg Gottlob and Toby Walsh (eds.), 2003. ISBN 0-127-05661-0

FRIEDMAN, J. H.; KOHAVI, R.; YUN ,Y., *Lazy Decision Trees*, Proc. of the Thirteenth National Conference on Artificial Intelligence, Howard Shrobe and Ted Senator (eds.), AAAI Press, pg. 717-724, 1996.

GASPARETTI, F., MICARELLI, A., *Adaptive Web Based on a Colony de Cooperative Distributed Agents*, Cooperative Information Agents (CIA) VII*, Springer-Verlag, LNAI 2782*, M. Klusch, A. Omicini, S. Ossowski and H. Laamanen (eds.), pp. 168-183, Helsinki, August, 2003. ISBN 3-540-40798-7

GEUTNER, P.; DENECKE, M.; MEIER, U.; WESTPHAL, M.; WAIBEL A., *Conversational Speech Systems for On-Board Car Navigation and Assistance*, In Proceedings of The 5th International Conference on Spoken Language Processsing ICSLP'98, December, 1998. http://www.ri.cmu.edu/pubs/pub_2982_text.html

GEORGEFF,, M.P., LANSKY, A.I., *Procedural knowledge*. In Proceedings of the IEEE Special Issue on Knowledge Representation, Vol. 74, pp. 1383-1398, 1986.

GONZALES, J. A.; HOLDER, L. B.; COOK, D., J., *Graph-Based Concept Learning*, Proceedings of the Fourteenth Annual Florida AI Research Symposium, pp. 377-381, 2001.

GOLLER, C.; LÖNING J.; WILL, T.; WOLF, W., *Automatic Document Classification: A thorough Evaluation of various Methods*, IEEE Intelligent Systems, nº 14, vol 1, pp. 75-87, 2000.

GRABOWICZ, P., *Better Internet Search Engines (Part 1): Intelligent Agents*, Online Journalism Review, march, 2002. http://www.ojr.org/ojr/technology/1017967102.php

GRIMNES, G. A. A., CHALMERS, S., EDWARDS, P., PREECE, A., *GraniteNights – A Multi-agent Visit Scheduler Utilising Semantic Web Technology*, Cooperative Information Agents (CIA) VII, Springer-Verlag, LNAI 2782, M. Klusch, A. Omicini, S. Ossowski, H. Laamanen (eds.), pp. 137-151, Helsinki, August, 2003. ISBN 3-540-40798-7

GRUBER, T. R., *A translation approach to portable ontologies*. Knowledge Acquisition, n. 5, vol 2, pp.199-220, 1993.

HEWITT, C., *Viewing Control Structures as Patterns of Passing Messages*, Artificial Intelligence, vol. 8, n. 3, pg 323-364, 1977.

HOLLAND, J.H., *Escaping Brittleness: the Possibilities of General-Purpose Learning Algorithms applied to Parallel Rule-Based Systems*, In: Machine Learning, an Artificial Intelligence Approach, vol 2, R.S. Michalski, J.G. Carbonell, T.M. Mitchell (eds.), Morgan Kauffmann, 1986.

HOLMES, J. H.; LANZI, P. L.; STOLZMANN, W.; WILSON, S. W., *Learning classifier systems: New models, successful applications*. Information Processing Letters, vol 82, ed. 1, pp. 23-30, 2002.

HOYLE, M. A.; LUEG, C., *Open Sesame !: a Look at Personal Assistants*, Proceedings of the International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 97), London, 21-23.4.97, pp 51-60.

HUHNS, M. N.; SINGH, M. P. (ed.), Readings in Agents. Morgan Kaufman Publishers Inc., 1998.

D'INVERNO, M.; KINNY, D.; LUCK, M.; WOODRIDGE, M., *A Formal Specification of dMARS*, Proceedings of the Fourth International Workshop on Agent Theories, Architectures and Languages, Springer-Verlag, Singh, Rao and Woodridge (eds.), pp. 155-176, 1998.

D'INVERNO, M. ; LUCK, M., *Understanding Agent Systems*, Spinger Verlag, Berlin, 2001. ISBN 3-540-41975-6

INDYK, P. ; MOTWANI, R., *Approximate nearest neighbors : Towards removing the curse of dimensionality*. In. Proceedings of the 30th ACM Symposium on Theory of Computing, pg. 604-613, 1998.

JADE, *Java Agent DEvelopment Framework*, http://sharon.cselt.it/projects/jade/

JENNINGS, N.; SYCARA, K.; WOOLDRIDGE, M., *A Roadmap of Agent Research and Development,* Autonomous Agents and Multi-Agent Systems, Vol. 1, No. 1, pp. 7 – 38, July, 1998.

JING, Y.; TAYLOR, N.; BROWN K., *An Intelligent Inference Approach to user Interaction Modeling in a Generic Agent Based Interface System*, Proceedings of the ECAI'02, F. van Harmelen (ed.), IOS Press, pp. 103-107, 2002. ISBN 1-58603-257-7

JOACHIMS, T. *A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization*, Proceedings of ICML-97, 14th International Conference on Machine Learning, Morgan Kaufmann Publishers, (ed. Douglas H. Fisher) San Francisco, US, pp. 143-151, 1997.

KASIF, S.; SALSZBERG, S.; WALTZ, D.; RACHLIN, J.; AHA, D., *A Probabilistic Framework for Memory-Based Reasoning*, Artificial Intelligence, vol 104, nº 1-2, pg. 287-311, 1998.

KASSEL, G.; ABEL, M-H.; BARRY, C.; BOULITREAU, P.; IRASTORZA, C.; PERPETTE, S., *Construction et exploitation d'une ontologie pour la gestion des connaissances d'une equipe de recherche*. In Actes de la Conférence en Ingénierie des Connaissances : IC'2000, Toulouse, pp 251-259.

KIPP, M.; ALEXANDERSSON, J.; REITHINGER, N., *Understanding Spontaneous Negotiation Dialogue*, Linköping University Electronic Press: Electronic Articles in Computer and Information Science, ISSN 1401-9841, vol. 4, nº 027, 1999. http://www.ep.liu.se/ea/cis/1999/027/

KISS, A.; QUINQUETON, J., *Machine Learning of User Preferences in a Corporate Knowledge Management System*, Proceedings of the Eighth International Symposium on the Management of Industrial and Corporate Knowledge ISMICK'01, Compiègne, France, October 22-24, 2001.

KNOBLOCK, C. A., AMBITE, J. L., *Agents for Information Gathering*. In Bradshaw, J.M. (ed.) Software Agents. MIT Press, 1997. pp. 347-373. ISBN 0-262-52234-9

KOUBARAKIS, M., *Multi-agent Systems and Peer-to-Peer Computing : Mehtods, Systems and Challenges*, Cooperative Information Agents VII, Springer-Verlag, LNAI 2782, M. Klusch, A. Omicini, S. Ossowski, H. Laamanen (eds.), pp. 46-61, Helsinki, August, 2003. ISBN 3-540-40798-7

De KRETSER, O.; MOFFAT, A.; SHIMMIN, T.; ZOBEL, J., *Methodologies for Distributed Information Retrieval*, Int. Conf. on Distributed Computing Systems, pg. 66-73, 1998. http://citeseer.nj.nec.com/dekretser98methodologies.html

KASS, R. ; FININ, T., *Modeling the user in natural language systems*. Computational Linguistics, vol 14, n. 3, pg 5-22, 1988.

KRULWICH, B., *Lifestyle Finder*, AI Magazine, vol. 18, nº 2, Summer 1997, pp. 37-46.

KOBSA, A., *Generic User Modeling Systems*, User Modeling and User-Adapted Interaction, Kluwer Academic Publishers, nº 11, pp 49-63, Netherlands, 2001.

KOCH, M. ; LACHER, M. S., *Integrating Community Services – A Common Infrastructure Proposal*, Proc. of the Knowledge-Based Intelligent Engineering Systems and Allied Technologies, pp. 56-59, Brighton, UK, September, 2000.

KOHAVI, R., *Bottom-Up Induction of Oblivious Read-Once Decision Graphs: Strengths and Limitations*, Proceedings of the National Conference on Artificial Intelligence, pp. 613-618, 1994.

KÖLZER, A., *Universal Dialogue Specification for Conversational Systems*, Linköping University Electronic Press: Eletronic Articles in Computer and Information Science, ISSN 1401-9841, vol. 4, nº 028, 1999. http://www.ep.liu.se/ea/cis/1999/028/

KOSALA, R.; BLOCKEEL, H., Web Mining Research: A Survey, ACM SIGKDD Explorations, vol. 2, ed. 1, pp. 1-15, July 2000.

KOZIEROK, R.; MAES, P., *A learning interface agent for scheduling meetings*. In Proceedings of the Fourteenth International Workshop on Intelligent User Interfaces, pages 81--88, Orlando, USA, 1993.

LACHER, M. S., *An Agent-based Knowledge Management Framework*, AAAI Workshop on Bringing Knowledge to Business Process, 20-22 March, 2000.

LACHER, M. S.; GROH, G., *Facilitating the exchange of explicit knowledge through ontology mappings*, 14 th International FLAIRS conference, AAAI Press, Key West, FL, 21-23 may, 2001.

LASHKARI, Y.; METRAL, M.; MAES P., *Collaborative Interface Agents*, In Proceedings of the Twelfth National Conference on Artificial Intelligence, vol. 1, AAAI Press, Seattle, WA, 1994. http://citeseer.nj.nec.com/lashkari94collaborative.html

LAWRENCE, S.; GILES, C. L., *Context and Page Analysis for improved Web Search*, IEEE Internet Computing, PP. 38-46, July-august, 1998.

LIEBERMAN, H., *Autonomous Interface Agents*, Proceedings of the ACM Conference on Computer and Human Interface, Atlanta, Georgia, March 1997.

LIEBERMAN, H., *Integrating User Interface Agents with Conversational Applications*, Proceedings of the ACM Conference on Intelligent User Interfaces, San Francisco, January, 1998.

LIEBERMAN, H.; VAN DYKE, N.; VIVACQUA, A., *Let's browse: a collaborative browsing agent*, Elsevier Science B. V., vol 12, decembe 1999, pp 427-431.

LIEBERMAN, H., *Intelligent Profiling by Example*, Proceedings of the International Conference on Intelligent user Interfaces (IUI 2000), Santa Fé, January 14-17, 2001.

LIU, J., *Web Intelligence (WI) : What Makes Wisdom Web ?*, Proceedings of IJCAI'03, Morgan Kaufmann Publishers, pp. 1596-1601, Georg Gottlob and Toby Walsh (eds.), 2003. ISBN 0-127-05661-0

MAES, P., *Learning Behavior Networks from Experience*, In: Toward a Practice of Autonomous Systems, Proceedings of the Firs European Conference on Artificial Life, F.J. Varela & P. Bourgine (eds.), MIT Press/Bradford Books, 1992.

MAES, P., *Modeling Adaptative Autonomous Agents*, Artificial Life Journal, n. 1, vol 1-2, MIT Press, pp.135-162, 1994.

MAGNINI, B. ; STRAPPARARA, C., *Improving User Modeling with Content-Based Technique*, Proceedings of the 8th International Conference User Modeling 2001, Springer-Verlag, Sonthofen, Germany, July, 2001. pp 74-83.

MAKAR, R.; MAHADEVAN, S.; GHAVAMZADEH, M., *Hierarchical Multi-Agent Reinforcement Learning*, Proceedings of the Fifth International Conference on Autonomous Agents, (J. P. Müller, E. Andre, S. Sen, C. Frasson eds.), ACM Press, pp. 246-253, 2001. ISBN: 1-58113-326-X

MARCU D.; BOICU M.; TECUCI, G., *User-Agent Interactions in Mixed-Initiative Learning*, Proc. of the 14th International FLAIRS Conference, Special track on Machine Learning, May 2001.

MATHER, L. A.; NOTE, J., *Discovering Encyclopedic Structure and Topics in Text*, KDD-2000 Workshop on Text Mining, August 20, Boston, 2000.

MAULSBY, D.; WITTEN I. H., *Teaching Agents to Learn: From User Study to Implementation*, IEEE Computer, vol 30, nº 11, 1997, pp 36-44.

MCROY, S.; ALI, S. S., *A practical, declarative theory of dialog*. Eletronic Transactions on Artificial Intelligence, vol. 3, Section D, 1999, 18 pg.

MENCZER, F., PANT, G., SRINIVASAN, P, RUIZ, M. E., *Evaluating Topic-Driven Web Crawlers*, Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 241-249, 2001.

MICHIE, D.; SPIEGELHALTER, D. J.; Taylor, C.C., *Machine Learning, Neural and Statistical Classification*, Ellis Horwood, 1994. ISBN 0-13-106360-X.

MIDDLETON, S. E., *Interface agents: A review of the field*, Technical Report, ECSTR-IAM01-001, University of Southampton, 2001, UK.

MINUT, S.; MAHADEVAN, S., *A Reinforcement Learning Model of Selective Visual Attention*, Proceedings of the Fifth International Conference on Autonomous Agents, (J. P. Müller, E. Andre, S. Sen, C. Frasson eds.), ACM Press, pp. 457-464, 2001. ISBN: 1-58113-326-X

MITCHELL, T. M., *The need for biases in learning generalization*, Readings in Machine Learning, J. W. Shavlik and T. G. Dietterich (eds.), pp. 184-191, Morgan Kaufmann Publishers, San Mateo, California, 1990. ISBN 1-55860-143-0

MITCHELL, T. M., *Machine Learning*, McGraw Hill, 1997. ISBN 0070428077

MLADENIC, D.; J. Stefan Institute, *Text-Learning and Related Intelligent Agents: A Survey*, IEEE Intelligent Systems, pp. 44-51, july/august 1999.

MOBASHER, B.; COOLEY, R.; SRIVASTAVA, J., *Automatic Personalization Based on Web Usage Mining*, Communications of the ACM, n. 8, vol. 43, august, 2000.

MORRIS, J.; MAES, P., SARDINE*: An Agent-facilitated Airline Ticket Bidding System*, In Proceedings of the Fourth International Conference on Autonomous Agents (Agents 2000), Barcelona, Catalonia, Spain, June 2-7, 2000.

MOULIN, B. ; CHAIB-DRAA, B., *An Overview of Distributed Artificial Intelligence*, In : Foundations of Distributed Artificial Intelligence, G. M. P. O'Hare and N. R. Jennings (eds.), John Wiley & Sons Inc., Canada, pp.3-56, 1996. ISBN 0-471-00675-0

MULDOON, C., O'HARE, G. M. P., PHELAN, D., STRAHAN, R., COLLIER, R.W., *ACCESS: An Agent Architecture for Ubiquitous Service Delivery*, Cooperative Information Agents (CIA) VII, Springer-Verlag, LNAI 2782, M. Klusch, A. Omicini, S. Ossowski and H. Laamanen (eds.), pp. 1-15, Helsinki, August, 2003. ISBN 3-540-40798-7

MULLER, H.; HILBRICH, T.; KUHNEL, R., *An Assistant Agent*, Journal Fundamenta Informaticae, vol 39, nº 3, 1999, pp 327-336. http://citeseer.nj.nec.com/muller99assistant.html.

NEGROPONTE, N., *Agents: From Direct Manipulation to Delegation*. In Bradshaw, J.M. (ed.) Software Agents. MIT Press, 1997. pp. 57-66. ISBN 0-262-52234-9

NILSSON, N.J., *Teleo-reactive programs for agent control*, Journal of Artificial Intelligence Research, vol 1, pp 139-158, 1994.

ODELL, J.; PARUNAK, H. van D.; Bauer, B., *Extending UML for Agents*. In: AOIS Workshop at AAAI, 2000.

OKAMOTO, M. ; YANG, Y. ; ISHIDA T., *Wizard of Oz method for Learning Dialog Agents*, Cooperative Information Agents V, LNAI 2182, Springer Verlag, M. Klusch and F. Zambonelli (eds.), 2001. ISBN 3-540-42545-4

OLIVEIRA, E., *Agents, Advanced Features for Negotiation and Coordination*, In: ACAI 2001, LNAI 2086, M. Luck (ed.), Springer-Verlag, pp. 173-186, 2001.

OMG, *Object Manager Group*, http://www.omg.org

PANNU, A. S.; SYCARA, K., *A Learning Personal Agent for Text Filtering and Notification*, Proceedings of the International Conference on Knowledge Based Systems, 1996.

PARAISO, E. C., BARTHES, J-P, *Voice Activated Information Entry: Technical Aspects*, Heudiasyc, Université de Technologie de Compiègne, Rapport Technique 04/2003, 2003.

PAUROBALLY, S. ; CUNNINGHAM, J., *Verification of Protocols for Automated Negotiation*, 15th European Conference on Artificial Intelligence ECAI2002, Frank van Harmelen (ed.), IOS Press, pp. 43-47, July 21-26, 2002. ISBN 1 58603 257 7

PAYNE, T. R.; EDWARDS, P.; GREEN, C. L., *Experience with Rule Induction and k-Nearest Neighbour Methods for Interface Agents that Learn*, Proceedings of the Workshop on Agents that learn from Other Agents, International machine Learning Conference, 1995.

PIER, J.H., Learning Classifier Systems: New Models, Successful Applications, http://citeseer.nj.nec.com/517270.html.

QUINLAN, J. R., *Induction of Decision Trees*, Machine Learning, vol 1. Kluwer Academic Publishers, pp. 81-106, Netherlands, 1986.

QUINLAN, J. R., *C4.5 : Programs for Machine Learning*, Morgan Kauffman Publishers, San Mateo, California, 1993.

RAMOS, M. P., *Structuration et évolution conceptuelles d'un agent assitant personnel dans les domaines techniques*. Thèse de Doctorat, Université de Technologie de Compiègne, décembre, 2000.

RAO, A.S.; GEORGEFF, M., *BDI Agents: from theory to practice*. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), pp. 312-319, San Francisco, CA, June, 1995.

RENNIE, J. D. M., ifile : An Application of Machine Learning to E-mail Filtering, KDD-2000 Text Mining Workshop, Boston MA, USA, August 20, 2000.

RICH, C.; SIDNER, C. L., *COLLAGEN: When Agents Collaborate with People*, Proceedings of the First International Conference on Autonomous Agents, Marina Del Rey, CA, USA, February, 1997, pp 284-291.

RICH, C.; SIDNER, C. L.; Lesh, N., *COLLAGEN: Applying Collaborative Discourse Theory to human-Computer Interaction*, AI Magazine, Special Issue on Intelligent User Interfaces, vol 22, issue 4, pp. 15-25, Winter 2001.

RICKEL, J. et.al., *Task-Oriented Tutoral Dialogue: Issues and Agents*, Fall Symposium on Building Dialogue Systems for Tutorial Application, AAAI, November, 2000.

RICKEL, J. et.al., *Building a Bridge between Intelligent Tutoring and Collaborative Dialogue Systems*, In. Proceedings of Tenth International Conference on AI in Education, pp. 592-594, IO Press, May, 2001.

RIESBECK, C. K.; SCHANK R. C., *Inside Case-Based Reasonning*. Lawrence Erlbaum Associates, ISBN 0-89859-767-6, EUA, 423p., 1989.

SALTON, G., *Automatic Text Processing: The Transformations, Analysis, and Retrieval of Information by Computer*, Addison-Wesley, 1989.

SARNER, M. H. ; CARBERRY, S., *Generating Tailored Definitions Using a Multifaceted User Model*. In : User Modeling and User-Adapted Interaction, vol 2, n. 4, pp 331-366, 1992.

SARWAR, B. M.; KARYPIS, G.; KONSTAN, J. A.; REIDL, J., *Item-based collaborative filtering recommendation algorithms*, In. World Wide Web, pp. 285-295, 2001. http://citeseer.nj.nec.com/sarwar01itembased.html

SAVOY, J., *Information Retrieval on the Web: A New Paradigm*, Upgrade The European Online Magazine for the IT Professional, (Ricardo Bayeza-Yates, Peter Schäuble eds.), n. 3, vol III, June 2002.

SCALABRIN, E. E., *Conception et réalisation d'environnement de développement de systèmes d'agents*. Thèse de Doctorat, Université de Technologie de Compiègne, Décembre 1996.

SCHANK, R. C., ABELSON, R. P., *Scripts, Plans, Goals and Understanding*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1977.

SHANKAR, S.; KARYPIS, G., *A Feature Weight Adjustment Algorithm for Document Categorization*, KDD-2000 Workshop on Text Mining, Boston, USA, August 2000.

SCHLICHTER, J.; KOCH, M. ; BÜRGER, M., *Workspace Awareness for Distributed Teams,* Proc. Coordination Technology for Collaborative Applications – Organizations, Processes and Agents, (Wolfram Conen, Gustaf Neumann eds.), Springer Verlag, Berlin, Singapore, 1997, pp. 199-218.

SEARLE, J. R., *Speech Acts*, Cambridge University Press, 1969.

SENEFF, S.; POLIFRONI, J., *Dialogue Management in the Mercury Flight Reservation System*, Satellite Dialog Workshop ANLP-NAACL, Seattle, April 2000.

SHAPIRO, D.; LANGLEY, P.; SHACHTER, R., *Using Background Knowledge to Speed Reinforcement Learning in Physical Agent*, Proceedings of the Fifth International Conference on Autonomous Agents, (J. P. Müller, E. Andre, S. Sen, C. Frasson eds.), ACM Press, pp. 254-261, 2001. ISBN: 1-58113-326-X

SHEN, W. ; NORRIE, D. H. ; BARTHÈS, J-P., *Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing*. Taylor & Francis, ISBN 0-7484-0882-7, Great Britain, 2001.

SHOENAUER, M.; SEBAG, M., *Incremental Learning of Rules and Meta-rules*, In Porter B. W. and Mooney R. (eds.), Machine Learning, Morgan Kaufmann, Los Altos/Palo Alto/San Francisco, pp.49-57, 1990.

da SILVA, J.L.T.; DEMAZEAU, Y, *Distributed Agenda Management through Decentralised Vowels Co-ordination Approach*. IBERAMIA 2002, Springer, Lecture Notes in Computer Science, vol. 2527, pp. 596-605, 2002. ISBN 3-540-00131-X

SILVERSTEIN, C., Henzinger, M. R., Marais, J, Moricz, M., *Analysis of a very large AltaVista query log*, ACM SIGIR, n. 33, pp. 6-12, 1999.

SMITH, R. G., *A Framework for Distributed Problem Solving*, In the Proceedings of the 6Th International Joint Conference on Artificial Intelligence. Tokyo, Japan, 1979. pp. 836-841.

SPIVEY, J. M., *The Z Notation: A Reference Manual*, 2 ed., Prentice Hall International (UK) Ltd, 1992.

STOCK, O., *Language-Based Interfaces and Their Application for Cultural Tourism*, American Association for Artificial Intelligence AAAI, Spring, 2001, pp 85-97.

SYCARA, K. ; PAOLUCCI, M. ; VAN VELSEN, M. ; GIAMPAPA, J. A., *The RETSINA MAS Infrastructure,* Technical Report CMU-RI-TR-01-05, Robotics Institute, Carnegie Mellon University, March, 2001.

TAC *Trade Agent Competition*, http://auction2.eecs.umich.edu/game.html.

TACLA, C.A., *Ba in the Construction of KM Systems Based on Agents,* Proceedings of ISMICK'01, October 2001, Compiègne, France, Tsuchiya, S. et Barthès, J.P. (eds.), pp. 167-176.

THOUVENIN, I.; ABEL, M. H. ; RAMOND, B.; QAMHIYAH, A*., Environment improvements for a better cooperation in international collaborative mechanical design*, Computers in Industry, 2003. (to appear).

TSAPARAS, P., *Nearest Neighbor in Multidimensinal Spaces*, Depth Oral Report 31902, Dept. of Computer Science, University of Toronto, 1999.

UTGOFF, P. E., *An Improved Algorithm for Incremental Induction of Decision Trees*, International Conference on Machine Learning, pp. 318-325, 1994.

WATKINS, C., *Learning from Deplayed Rewards*, PhD Thesis, King's College, Cambridge, 1989.

WEBB, G. I.; PAZZANI, M. J.; BILLSUS, D., *Machine Learning for User Modeling*, User Modeling and User-Adapted Interaction, Kluwer Academic Publishers, nº 11, pp 19-29, Netherlands, 2001.

WELD, D. S., ANDERSON, C., DOMINGOS, P., ETZIONI, O., GAJOS, K., LAU, T., WOLFMAN, S., *Automatically Personalizing User Interfaces*, Invited Talk in IJCAI'03, Morgan Kaufmann Publishers, pp. 37-42, Georg Gottlob and Toby Walsh (eds.), 2003. ISBN 0-127-05661-0.

WILSON, D. R.; MARTINEZ, A. R., *Instance pruning techniques*. Machine Learning, vol. 38, n. 3, pp 257-286, 2000.

WU, S.; GHENNIWA, H.; SHEN, W., *User model of a personal assistant in collaborative design environments, First* International Workshop on Agents in Design WAID'02, Key Centre of Design Computing and Cognition, Cambridge MA, 29-31 August, 2002. ISBN 1 864 87 1202

ZUKERMAN, I.; ALBRECHT, D. W., *Predictive Statistical Models for User Modeling*, User Modeling and User-Adapted Interaction, Kluwer Academic Publishers, nº 11, pp 5-18, Netherlands, 2001.

ZUKERMAN, I.; LITMAN, D., *Natural Language Processing and* User *Modeling: Synergies and Limitations*, User Modeling and User-Adapted Interaction, Kluwer Academic Publishers, nº 11, pp 129-158, Netherlands, 2001.

Annexes

Annexe 1 – Conception des SMA

Annexe 2 – Grammaire de la langue anglaise

Annexe 3 - Représentation de l'ontologie avec MOSS

Annexe 3A - Création d'objets MOSS

Annexe 3B - Création de l'ontologie de tâches avec MOSS

Annexe 4 - Le langage MAIS-L

Annexe 1 - Conception des SMA

Dans cette section nous discutons quelques concepts généraux rencontrés dans la mise en place d'un SMA.

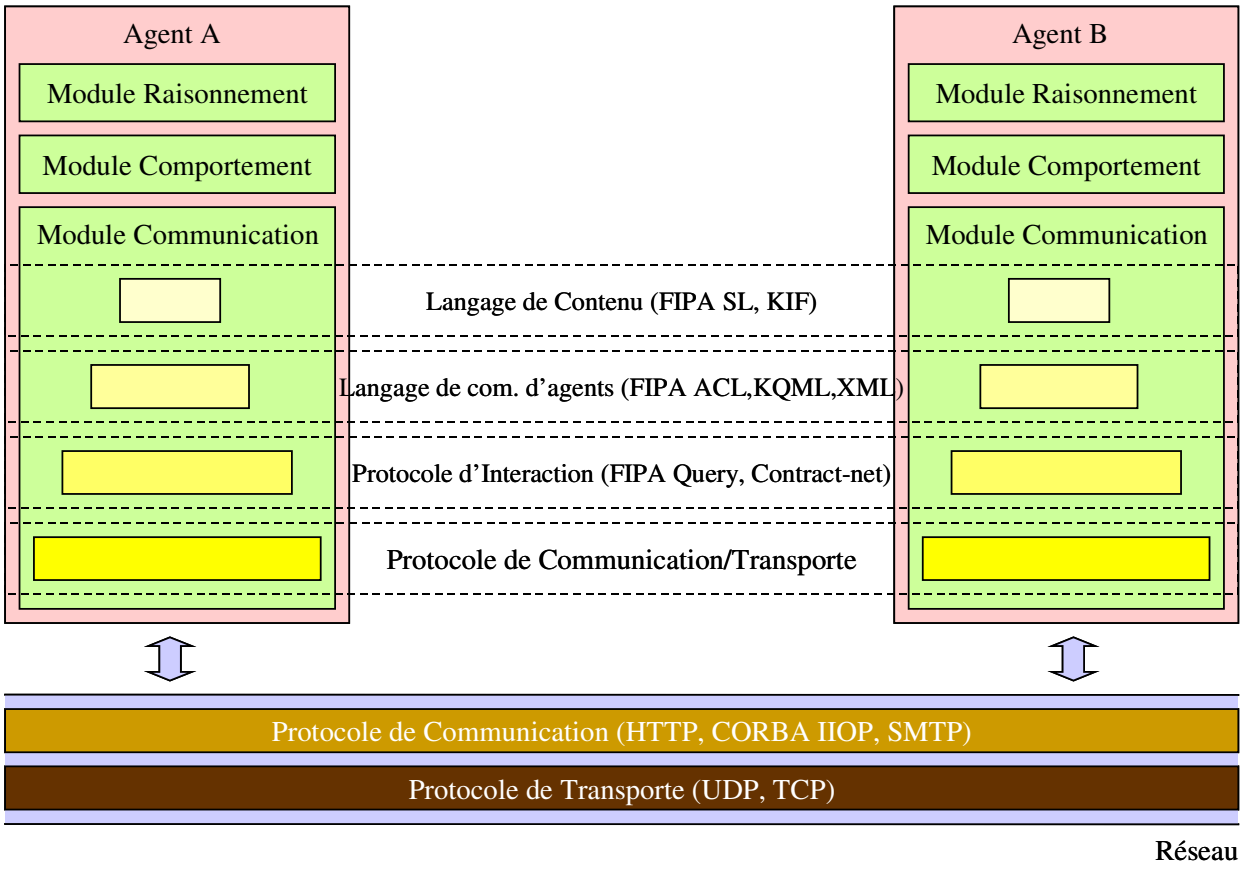

Figure 67 Communication entre agents d'un SMA.

La Figure 67 présente quelques concepts qui sont souvent utilisés dans la construction d'un SMA. Généralement, un SMA possède des agents distribués dans un environnement réseau.

Nous avons abstrait dans la Figure 67 les couches logicielles qui constituent le modèle réseau OSI car ceci ne fait pas partie du contexte de ce mémoire. Le réseau sert de moyen de transport aux messages transmis par un protocole de transport. Celui-ci peut être connecté comme par exemple TCP (Transfer Control Protocol) ou non-connecté comme l'UDP (User Datagram Protocol). Les protocoles connectés demandent une architecture client/serveur au niveau de communication et dans un SMA sont souvent mis en œuvre avec un canal de communication, parfois nommé *serveur de noms* ou *facilitateur*. Le canal de communication est utilisé pour centraliser l'adresse des agents et jouer le role d'intermédiaire pour la transmission de messages, évitant ainsi que chaque agent ne soit obligé de connaître le nom et l'adresse des autres agents. La communication avec un protocole non-connecté comme UDP est faite en partageant une porte unique de communication pour tous les agents. UDP est généralement utilisé dans des communications courtes, car il ne contient pas d'algorithme de vérification de données, tandis que TCP sert de base à un nombre de protocoles de communication pour la transmission à longue distance, car il est capable d'assurer que la transmission se fait sans perte d'information.

Au niveau des agents, chacun doit avoir un module d'interprétation et de codification du protocole de communication et/ou de transport. Dans le cas d'un modèle de communication connecté, cela se fait généralement avec un client mail pour SMTP, un client HTTP ou encore un ORB (Object Request Broker) dans le cas de CORBA (voir référence OMG). Les agents doivent partager la même sémantique pour le type de chaque message qui est transmis dans un processus de communication ainsi que les changements provoqués par ces messages à l'intérieur de l'agent. Pour cela ils utilisent un protocole d'interaction commun, par exemple, le Contract Net. La structure de chaque message est donnée par un langage de communication entre agents (ACL). Finalement, l'interprétation d'un message s'achève avec l'analyse du contenu du message représenté dans un langage donné.

Nous pouvons voir que dans chaque couche logicielle concernant la communication, un nombre énorme d'outils est disponible (et il augmente chaque jour). Par conséquent, il est très difficile d'imaginer comment deux SMAs développés avec des plates-formes[14] d'agents différentes pourront communiquer. Pour cela, la FIPA (Foundations for Intelligent Physical Agents) (FIPAa) propose des normes et des outils standards à utiliser dans chaque couche

[14] Une plate-forme d'agents est composée par les resources logicielles partagées par les agents. Une plate-fome sert à mettre en place des SMAs.

présente sur la Figure 67. Nous avons discuté cet aspect dans la section 2.5 pour introduire notre plate-forme d'agent.

Annexe 2 - Grammaire de la langue anglaise

Les fonctions présentées sous la forme BNF dans cette section sont utilisées par l'analyseur syntaxique du système de dialogue présenté dans la section 3.3. La grammaire anglaise est composé de 15 règles syntaxiques (S1-S15) et utilise un algorithme de matching non-déterministe (fonction *matching*). Le résultat de l'analyse syntaxique est une structure syntaxique similaire à celle présentée par Allen (1986) (voir section 3.3.3).

Opérateurs utilisés pour la définition de la grammaire	
Opérateur	Définition
|	Disjonction
||	Conjonction
\+	Négation

La grammaire

;=====================================
; S (Sentence, an english sentence)
;=====================================
S ← S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8 | S9 | S10 | S11 | S12 | S13 | S14 | S15

S1 ← VP-ACTION NP

S2 ← VP

S3 ← AUX NP VP

S4 ← VP NP

S5 ← PP AUX NP VP/PP

S6 ← PP NOUN AUX NP VP/PP

S7 ← WH-WORD

S8 ← PP VP

S9 ← ADV

S10 ← NP AUX VP

S11 ← NP VP

S12 ← NP

S13 ← EXPLA AUX NP VP

S14 ← EXPLA NP VP/PP

S15 ← EXPLA VP NP

;===
; U (Unknown)
;===
U ← U1 | U2 | U3 | U4

U1 ← UNKNOWN U

U2 ← NOUN UNKNOWN

U3 ← UNKNOWN NOUN

U4 ← UNKNOWN

;===
; VP (Verb Phrase)
;===
VP ← VP1 | VP2 | VP3

VP1 ← SIMPLE-VP

VP2 ← PRE SIMPLE-VP

VP3 ← SIMPLE-VP PPS

;===
; VP/PP (Verb Phrase)
;===
VP/PP ← SIMPLE-VP PPS/PP

```
;=======================================
; SIMPLE-VP (Simple Verb Phrase)
;=======================================
SIMPLE-VP ← SIMPLE-VP1 | SIMPLE-VP2

SIMPLE-VP1 ← VERB

SIMPLE-VP2 ← VERB NP

;=======================================
; VP (Verb Phrase - Action)
;=======================================
VP-ACTION ← VPA1 | VPA2 | VPA3

VPA1 ← SIMPLE-VPA

VPA2 ← PRE SIMPLE-VPA

VPA3 ← SIMPLE-VPA PPS

;=======================================
; VP/PP (Verb Phrase - Action)
;=======================================
VP/PPA ← SIMPLE-VPA PPS/PP

;=======================================
; SIMPLE-VP (Simple Verb Phrase - Action)
;=======================================
SIMPLE-VPA ← SIMPLE-VPA1 | SIMPLE-VPA2

SIMPLE-VPA1 ← VERB-ACTION

SIMPLE-VPA2 ← VERB-ACTION NP

;=======================================
; NP (Noun Phrase)
;=======================================
NP ← NP1 | NP2 | NP3 | NP4 | NP5 | NP6 | NP7 | POSSESSIVE

NP1 ← ART NOUN

NP2 ← ART NOUN PPS

NP3 ← PROPER-NAME

NP4 ← PRO

NP5 ← U
```

NP6 ← ART U

NP7 ← ART U PPS

;===
; Possessive
;===
POSSESSIVE ← POSS1 | POSS2 | POSS3 | POSS4 | POSS5 | POSS6 | POSS7

POSS1 ← PROPER-NAME AP NOUN

POSS2 ← ART U AP U

POSS3 ← U AP U

POSS4 ← U AP PROPER-NAME

POSS5 ← ART U AP PROPER-NAME

POSS6 ← PROPER-NAME AP U

POSS7 ← U AP NOUN

;===
; NP/PP (Noun Phrase / Propositional Phrase)
;===
NP/PP ← ART NOUN PPS/PP

;===
; PPS (Propositional Phrase *)
;===
PPS ← PPS1 | PPS2

PPS1 ← PP

PPS2 ← PP PPS

;===
; PPS/PP (Propositional Phrase *) / Propositional Phrase
;===
PPS/PP ← PPS/PP1 | PPS/PP2

PPS/PP1 ← null

PPS/PP2 ← PPS

;=======================================
; PP (Propositional Phrase)
;=======================================
PP ← PP1 | PP2 | PP3 |

PP1 ← PRE NP

PP2 ← WH-WORD

PP3 ← PRE WH-WORD

;=======================================
; Basic function applied on the **first** word of the sentence
; INPUT: word
; OUTPUT: true or false
; GOAL: tell if the word belongs to “unknown” syntactic category
;=======================================
UNKNOWN ← \+ ART || \+ AUX || \+ ADV || \+ WH-WORD
|| \+ PROPER-NAME || \+ PRO || \+ NOUN || \+ PRE
|| \+ VERB || \+ EXPLA || \+ “S”

;=======================================
; Basic functions applied on sentences containing **one single word**
; INPUT: word
; OUTPUT: true or false
; GOAL: tell if the word belongs to a given syntactic category
;=======================================
WH-WORD ← member of the list of “wh” words. Ex.: (why, what, when, …)

EXPLA ← member of (why, how)

PRE ← member of the list of prepositions. Ex. : (pos, from, in, on, out, up, to, over, under, at, of, for, with)

AP ← '

ADV ← member of the list of adverbs. Ex.: (yes, no, sure, ok, none, nobody, any)

ART ← member of the list of articles. Ex.: (a, an, the)

NOUN ← member of the list of nouns. Ex.: (seat, house, mail, text, morning, age, e-mail, address, name, document, title, paper, file, article, information, flight, time, baby, box, corner, dialog, task, subject, carbon-copy, message, address, arrival, depart, author, date, year, subject, theme, today, morning, page, webpage, web-page, web)

PROPER-NAME ← member of the list of proper names. Ex.: (mary, boston, cesar, marco, barthes, fabricio)

PRO ← member of the list of pronouns. Ex.: (I, you, he, she, it, we, they, me, them, this, these, those, that, my, our, your)

VERB ← member of the list of verbs. Ex. : (set, can, book, do, does, is, exit, like, works, see, eat, am, work, find, locate, search, return, execute, leave, carrying, put, send, excuse, go, burn, hidden, start, abort, cancel, want, know, write, compose, arrive, teach, teaches, means, mean, produce, produces, build, allow, allows, create, creates, look)

VERB-ACTION ← member of the list of verbs used in actions. Ex.: (search, look, give, return, show, compute, leave, arrive, go, does, do, work, teach, teaches, teach, means, create, creates, produce, build, allow, make)

AUX ← member of the list of modal verbs. Ex.: (can, do, does, did, should, may, might, must, could)

```
;;;=====================================
;;;  Non-deterministic matching algorithm
;;;=====================================
(defun matching (structs terms-list)
  (if (and (not structs) terms-list)
      nil
    (if (and structs (equal (car structs) 'pps/pp))
        (matching (cdr structs) terms-list) ;matching
      (if (and structs (not terms-list))
          nil  ;;
        (if (and (not structs) (not terms-list))
            '(t)
          (if (terminal (car structs))
              (let ((res nil)
                    (val1  (matching (cdr structs) (cdr terms-list)))
                    (val (car (eval `(,(car structs) ',(list (car terms-list)))))))
                (if (and val val1)
                    (progn
                      (setf val1 (remove '(t) val1 :test #'equal))
                      (setf val (remove '(t) val :test #'equal))
                      (dolist (v val1)
                           (setf res (append res (insert-first1  val v (car structs) (cdr structs)))) ;insert
first
                        )
                      )
                  )
                RES)

            (let ((val nil) (val1 nil) (temp nil) (cont 0))
              (loop
                (if (> cont (length terms-list))
                    (return))
                (setf val (remove '(t) (eval `(,(car structs) ',(subseq terms-list 0 cont))) :test #'equal))
                (if val
                    (progn
```

```
                (setf val1 (matching (cdr structs) (subseq terms-list cont (length terms-list))))
;ok
                (if (and val val1)
                    (progn
                      (setf val (remove '(t) val :test #'equal))
                      (setf val1 (remove '(t) val1 :test #'equal))
                      (dolist (v val)
                        (dolist (v1 val1)
                          (setf temp  (append temp (insert-first1 v v1 (car structs) (cdr structs))) )
;insert first
                          )
                        )
                      )
                  )
                )
              )
             (setf cont (+ 1 cont)))
            (remove '(t) temp :test #'equal))
          )
        )
      )
    )
  )
 )
```

Annexe 3 - Représentation de l'ontologie avec MOSS

Annexe 3A – Création d'objets MOSS

Ces fonctions ont été construites pour faciliter l'utilisation de MOSS. Elles constituent une couche au-dessus MOSS et servent à créer dynamiquement des objets et propriétés.

```
;;;=====================================
;;;  Functions for creating objects on MOSS
;;;=====================================

(defun make-object (&key label synonyms isa slots instance)
  (if isa
      (make-object1 :label label :synonyms synonyms :isa isa :slots slots :instance instance)
    (make-object1 :label label :synonyms synonyms :slots slots :instance instance))
  )

(defun make-object1 (&key label synonyms isa slots instance)
  (verify-create-slots slots)
  (let ((l nil))
    (if isa
        (if instance
            (setf l (eval `(m-definstance ,isa ontology (has-name ,(format nil "~S" label)) (has-
synonyms ,@synonyms) ,@slots)))
          (if synonyms
              (progn
                (setf l (eval `(m-defclass ,label ,(gentemp (read-from-string (subseq (format nil
"~A" label) 0 3))) ontology (:is-a ,isa) (:tp name) (:tp SYNONYMS (:entry)) ,@slots)))
                (add-slot l 'synonyms synonyms 'ontology))
            (setf l (eval `(m-defclass ,label ,(gentemp (read-from-string (subseq (format nil "~A"
label) 0 3))) ontology (:is-a ,isa) (:tp name) (:tp SYNONYMS (:entry)) ,@slots)))))
      (if instance
          (setf l (eval `(m-defobject ontology (has-name ,label) (has-synonyms ,@synonyms)
,@slots)))
        (if synonyms
            (progn
```

```
          (setf l (eval `(m-defclass ,label ,(gentemp (read-from-string (subseq (format nil "~A"
label) 0 3))) ontology (:is-a ,isa) (:tp name) (:tp SYNONYMS (:entry)) ,@slots)))
          (add-slot l 'synonyms synonyms 'ontology))
        (setf l (eval `(m-defclass ,label ,(gentemp (read-from-string (subseq (format nil "~A"
label) 0 3))) ontology (:is-a ,isa) (:tp name) (:tp SYNONYMS (:entry)) ,@slots)))))
     )
   l)
 )

;;;=====================================
;;;  Knowledge Base - Vacuum Cleaner - Concepts
;;;=====================================

(make-object :label 'parallel-line :isa 'non-animate  :synonyms '("parallel-line" "line")
        :slots '((has-definition "This is the definition of parallel line"))
        :instance t)

(make-object :label 'power-cord :isa 'non-animate  :synonyms '("cord" "power-cord")
        :slots '((has-definition "This is the definition of power cord"))
        :instance t)

(make-object :label 'vacuum-cleaner :isa 'non-animate
        :synonyms '("vacuum" "vacuum-cleaner")
        :slots '((has-definition "This is the definition of vaccum cleaner") (has-production
"vacuum-production"))
        :instance t)

(make-object :label 'ac-motor :isa 'non-animate
        :synonyms '("ac-motor" "motor" "ac-electric-motor" "ac-drive-motor")
        :slots '((has-definition "This motor must turn a fan unit on one side of the armature and
a belt drive on the other side of the armature.") (has-production "ac-motor-production"))
        instance t)

(make-object :label 'dc-motor :isa 'non-animate
        :synonyms '("dc-motor" "motor" "dc-electric-motor" "dc-drive-motor")
        :slots '((has-definition "This motor is used to drive the wheels on the back forward or
reverse.") (has-production "dc-motor-production"))
        :instance t)

(make-object :label 'suction-fan :isa 'non-animate  :synonyms '("fan" "suction-fan")
        :slots '((has-definition "This is the definition of a suction fan"))
        :instance t)

(make-object :label 'belt-drive :isa 'non-animate  :synonyms '("belt-drive" "belt" "drive")
        :slots '((has-definition "This is the definition of a belt drive"))
        :instance t)

(make-object :label 'armature :isa 'non-animate  :synonyms '("armature")
        :slots '((has-definition "This is the definition of an armature"))
        :instance t)
```

```
(make-object :label 'ac-dc-converter :isa 'non-animate
        :synonyms '("converter" "ac-converter" "dc-converter" "ac-to-dc-converter" "ac-dc-
converter" "power-supply" "power-supplies" "wall-converter")
        :slots '((has-definition "This is the definition of an ACto DC converter")
(has-production "power-converter-production"))
        :instance t)

(make-object :label 'mesure :isa 'abstract  :synonyms '("mesure"))

(make-object :label 'ac :isa 'mesure
        :synonyms '("ac" "alternative-current" "current" "vac" "power" "voltage")
        :slots '((has-definition "This is the definition of AC"))
        :instance t)

(make-object :label 'dc :isa 'mesure
        :synonyms '("dc" "direct-current" "current" "vdc" "power" "voltage")
        :slots '((has-definition "This is the definition of DC"))
        :instance t)

(make-object :label 'receiver-box :isa 'non-animate  :synonyms '("receiver" "receiver-box")
         :slots '((has-definition "This is the definition of receiver box"))
        :instance t)

(make-object :label 'remote-control :isa 'non-animate
        :synonyms '("remote-control" "remote-unit" "remote-controls" "remote-control-unit"
"remote-control-unit-kit" "remote-control-units")
        :slots '((has-definition "This is the definition of remote control") (has-production
"remote-control-production"))
        :instance t)

(make-object :label 'hand-held-control-unit :isa 'non-animate
        :synonyms '("hand-held" "hand-held-control" "hand-held-control-unit")
        :slots '((has-definition "This is the definition of hand held control"))
        :instance t)

(make-object :label 'signal-receiving-unit :isa 'non-animate
        :synonyms '("signal-receiving" "signal-receiving-unit" "signal-unit")
        :slots '((has-definition "This is the definition of signal receiving unit"))
        :instance t)

(make-object :label 'servo :isa 'non-animate  :synonyms '("servo" "servos")
        :slots '((has-definition "This is the definition of signal receiving unit"))
        :instance t)

(make-object :label 'drive-wheel-motor :isa 'non-animate
        :synonyms '("wheel" "drive-wheel-motor")
        :slots '((has-definition "This is the definition of drive-wheel-motor"))
        :instance t)
```

```
(make-object :label 'switch :isa 'non-animate
        :synonyms '("switch" "manufactured-switch" "switches")
        :slots '((has-definition "This is the definition of switch"))
        :instance t)

(make-object :label 'pivot :isa 'non-animate :synonyms '("turning-pivot" "pivot")
        :slots '((has-definition "This is the definition of pivot"))
        :instance t)
```

Annexe 3B – Création de l'ontologie de tâches avec MOSS

```
;====================================
;  Knowledge Base - Vacuum Cleaner - Tasks
;====================================

(make-object :label 'task :isa 'abstract
        :synonyms '("task" "activity" "job")
        :slots '((:tp justify) (:tp explanation) (:tp subtasks)))

(make-object :label 'vacuum-production :isa 'task :synonyms '("vacuum-production")
         :slots '((has-subtasks "ac-motor-production" "power-converter-production" "remote-
control-production" "servos" "dc-motor-production"))
        :instance t)

(make-object :label 'ac-motor-production :isa 'task
        :synonyms '("ac-motor-production")
        :slots '((has-justify "The production of an AC motor for a Suction Cleaner needs take
into acount a relative speed, high torque to turn a fan unit and the belt drive.")
(has-explanation "The production of an AC motor must be as..."))
        :instance t)

(make-object :label 'power-converter-production :isa 'task
        :synonyms '("converter-production")
        :slots '((has-justify "The 120 VAC or 220 VAC power voltage must be converted
around 5 VDC needed to power ther drive the motor and receiver box.") (has-explanation
"Normally, a converter is made..."))
        :instance t)

(make-object :label 'remote-control-production :isa 'task
        :synonyms '("remote-control-production")
        :slots '((has-justify "The production of a remote control is important because...")
                (has-explanation "Normally, a remote control is made..."))
        :instance t)

(make-object :label 'dc-motor-production :isa 'task
        :synonyms '("dc-motor-production")
        :slots '((has-justify "The motor is used to drive the wheels on the bak forward or
reverse.") (has-explanation "A switch is needed to physically alternate the polarities from the
power supply so the motor runs in a clockwise or CCW direction."))
        :instance t)
```

Annexe 4 - Le langage MAIS-L

La fonction (USER-LOGIN User-Name)

Nom de la Fonction :	USER-LOGIN
Type ou valeur de retour :	Une structure du type *User-Profile*
Valeur d'erreur :	""
Définition :	Pour exécuter des requêtes, il faut d'abord créer un utilisateur. Pour cela nous pouvons tout simplement exécuter la commande "USER-LOGIN". À ce jour, il n'y a pas de sécurisation (mot de passe). "USER-LOGIN" va créer un nouvel utilisateur s'il n'existe pas et rendre comme réponse une structure du type "User-Profile" ou rendre comme réponse le profil de l'utilisateur déjà existant.
Paramètres	
Nom	Type
User-Name	une structure *User-Name*
Exemple	
(USER-LOGIN "fabricio")	

La fonction (VERIFY-IF-RESULT User-Name ID)

Nom de la Fonction :	VERIFY-IF-RESULT
Type ou valeur de retour :	:done
Valeur d'erreur :	""
Définition :	Vérifie si une recherche sous l'identificateur *ID*, déclenchée par un utilisateur *User-Name*, a été déjà faite.
Paramètres	
Nom	Type
User-Name	une structure *User-Name*
ID	une structure Query-ID

La fonction (REMOVE-QUERY-FROM-PROFILE User-Name ID)

Nom de la Fonction :	REMOVE-QUERY-FROM-PROFILE
Type ou valeur de retour :	:done
Valeur d'erreur :	""
Définition :	Si la recherche sous l'identificateur *ID* de l'utilisateur *User-Name* a été déjà faite alors la recherche est supprimée du profil de l'utilisateur. Si la recherche est encore en cours, elle est annulée.
Paramètres	
Nom	Type
User-Name	une structure *User-Name*
ID	une structure Query-ID

La fonction (GIVE-QUERY-RESULT User-Name ID)

Nom de la Fonction :	GIVE-QUERY-RESULT
Type ou valeur de retour :	Query-Answer
Valeur d'erreur :	""
Définition :	Si la recherche sous l'identificateur *ID* de l'utilisateur *User-Name* a été déjà faite alors la fonction retoune le résultat de la recherche.
Paramètres	
Nom	Type
User-Name	une structure *User-Name*
ID	une structure Query-ID

La fonction (UPDATE-PROXY Server Port User-Name)

Nom de la Fonction :	UPDATE-PROXY
Type ou valeur de retour :	:done
Valeur d'erreur :	""
Définition :	Sert à mettre à jour le serveur proxy et le numéro de port dans le profil de l'utilisateur. Ces informations sont utilisées quand un client standalone est utilisé. Si nous utilisons les services à partir d'un agent ou d'une plate-forme à distance, nous n'avons pas besoin de l'utiliser.
Paramètres	
Nom	Type
Server	une structure *Server*
Port	une structure *Port*
User-Name	une structure *User-Name*
Exemple	
(UPDATE-PROXY "sigma-si.utc.fr" 3128 "fabricio")	

La fonction ADD-FOLD-ON-PROFILE

Nom de la Fonction :	ADD-FOLD-ON-PROFILE
Type ou valeur de retour :	:done
Valeur d'erreur :	""
Définition :	Sert à ajouter un dossier dans le profil de l'utilisateur.
Paramètres	
Nom	Type
Fold-Name	une string
User-Name	une structure *User-Name*

La fonction (ADD-SITE-ON-PROFILE Site-Item Fold-Name User-Name)

Nom de la Fonction :	ADD-SITE-ON-PROFILE
Type ou valeur de retour :	:done
Valeur d'erreur :	""
Définition :	Sert à ajouter une adresse web dans le profil de l'utilisateur. Notez l'existence du symbole « ' » devant la liste. Il est obligatoire.
Paramètres	
Nom	Type
Site-Item :	une structure Site-Item
Fold-Name	une string
User-Name	une structure *User-Name*
Exemple	
(ADD-SITE-ON-PROFILE '("http://www.fipa.org" "FIPA Home Page") "Standards-Agents" "fabricio")	

La fonction (RENAME-FOLD-ON-PROFILE New-Fold-Name Fold-Name User-Name)

Nom de la Fonction :	RENAME-FOLD-ON-PROFILE
Type ou valeur de retour :	:done
Valeur d'erreur :	""
Définition :	Sert à changer le nom d'un dossier existant dans le profil de l'utilisateur.
Paramètres	
Nom	Type
New-Fold-Name :	une string
Fold-Name	une string
User-Name	une structure *User-Name*

La fonction (REMOVE-SITE-FROM-PROFILE Fold-Name Site User-Name)

Nom de la Fonction :	REMOVE-SITE-FROM-PROFILE
Type ou valeur de retour :	:done
Valeur d'erreur :	""
Définition :	Sert à supprimer un site existant dans un dossier du profil de l'utilisateur.
Paramètres	
Nom	Type
Fold-Name:	une string
Site	une structure *Site*
User-Name	une structure *User-Name*

La fonction (REMOVE-FOLD-FROM-PROFILE Fold-name User-Name)

Nom de la Fonction :	REMOVE-FOLD-FROM-PROFILE
Type ou valeur de retour :	:done
Valeur d'erreur :	""
Définition :	Sert à supprimer un dossier existant dans le profil de l'utilisateur.
Paramètres	
Nom	Type
Fold-Name:	une string
User-Name	une structure *User-Name*

La fonction (START-REQUEST User-Name Words Number-Sites Domains Date Query-ID Read)

Nom de la Fonction :	START-REQUEST
Type ou valeur de retour :	:done
Valeur d'erreur :	""
Définition :	Déclenche une recherche de sites. La recherche est faite de la manière suivante : L'agent « searcher@OMAS » envoie une requête de recherche vers l'agent *Filtrage*. Cet agent, à son tour, demande des pages à des agents qui recherchent sur Google, Altavista et Lycos. Chaque moteur de recherche possède un agent spécifique. Ensuite l'agent *Filtrage* ramasse les pages, les ordonne selon le profil de l'utilisateur et enregistre la réponse dans l'espace de travail de l'utilisateur. À ce jour, le nombre de requêtes traitées en parallèle est égal au nombre d'agents du type *Filtrage* existant dans la plate-forme OMAS. Tous ces agents sont complètement distribués. La fonction *START-REQUEST* reçoit 7 paramètres: • Le nom de l'utilisateur qui fait la recherche ; • Une liste de termes ; • Le nombre de pages voulues dans la réponse ; • Une liste de domaines : pour l'instant la liste de domaines est très réduite. Parmi les domaines déjà modélisés dans le système on en trouve *ai-adaptative-agents*, ai-*autonomous-agents*, *ai-mbr*, *ai-dialogue-systems*, *ai-agents-and-ontologies*, *ai-personal-assistants* ; • Query-ID : une string unique qui sert à identifier la requête ; • Date : un numéro de 10 chiffres ; • Read : "No" Notez l'existence du symbole « ' » devant la liste. Il est obligatoire. Le résultat des requêtes doit normalement être différent selon les pages ajoutées par l'utilisateur à son profil et les domaines choisis dans la recherche. Normalement une recherche dure plusieurs minutes (10, 20 ou 30). Cela dépend du nombre de pages demandés, du nombre de pages existantes dans les favoris et du nombre de domaines choisis.

Paramètres	
Nom	Type
User-Name :	une structure *User-Name*
Words :	une structure *Words*
Number-Sites :	un numéro
Domains :	une structure *Domains*
Date :	une structure *Date*
Query-ID :	une structure *Query-ID*
Read :	“No”
Exemple	
(START-REQUEST “fabricio” ‘(personal assistant) 10 ‘(AI-MBR AI-Autonomous-Agents) 3859457894 “fab-001”)	

Nom de la Structure :	User-Profile
Exemple :	(:DOMAINS (AI-AGENTS AI-PERSONAL-ASSISTANT AI-AUTONOMOUS-AGENTS AI-ADAPTATIVE-AGENTS) :HTTP-SERVER "sigma-si.utc.fr" :HTTP-PORT 3128 :FOLDERS (("lessons-learned" (("http://www.aic.nrl.navy.mil/AAAI00-ILLS-Workshop/" "AAAI-00 Workshop: Intelligent Lessons Learned Systems"))) ("CBR" (("http://www.coginst.uwf.edu/projects/caseBasedCapture/index.html" "Adapting an Attention Direction Piloting Display") ("http://orgwis.gmd.de/projects/fabel/fabel.html" "GMD-FIT, FABEL project") ("http://www.cc.gatech.edu/aimosaic/faculty/kolodner.html" "Janet Kolodner") ("http://www.aic.nrl.navy.mil/~aha/aaai99-kmcbrw/papers/" "AAAI-99 KM/CBR Workshop: Papers") ("http://www.cbr-web.org/CBR-Web/?info=welcome&menu=1" "The CBR Homepage") ("http://www.aic.nrl.navy.mil/~aha/" "David W. Aha: Home Page") ("http://www.ai-cbr.org/theindex.html" "Case-Based Reasoning [the AI-CBR Homepage]") ("http://online.loyno.edu/cisa494/material.html" "Case-based Reasoning: Course Materials"))) ("knowledge-management" (("http://www.bus.utexas.edu/kman/" "Knowledge Management Home Page")))))

Nom de la Structure :	Query-Answer
Exemple :	"(((\"http://www.xe.com/pca/\" \"XE.com - The Personal Currency Assistant (tm)\" \"home\") 8.948293) ((\"http://www.vipdesk.com/\" \"Personal Assistant and Corporate Concierge Services - VIPdesk\" \"home\") 8.948293) ((\"http://www.bigdates.com/\" \"Your FREE personal assistant. Reminders, Greeting Cards, Egreetings, Gifts, Shopping, Birthdays, Holidays...\" \"CBR\") 8.879593) ((\"http://www.cisco.com/en/US/products/sw/voicesw/ps2026/\" \"Cisco Personal Assistant - Cisco Systems\" \"CBR\") 8.870356) ((\"http://www.siteexperts.com/assist/about.asp\" \"SiteExperts.com - Personal Assistant\" \"CBR\") 8.710824))"