Une architecture d'agents pour la simulation

Le modèle YAMAM et sa plate-forme Phoenix

THÈSE

présentée et soutenue publiquement le 27/06/2003
pour l'obtention du

Doctorat de l'INSA de Rouen
(Spécialité informatique)

par

Marc Savall

Composition du jury

Thierry Paquet	Université de Rouen	Président
Jean-Marc Ogier	Université LaRochelle	Rapporteur
Philippe Mathieu	Université de Lille	Rapporteur
Jean-Pierre Pécuchet	INSA de Rouen	Directeur
Nathalie Chaignaud	INSA de Rouen	Examinateur

REMERCIEMENTS

Le travail de recherche présenté dans ce mémoire a été mené au sein de l'équipe PSI de l'INSA de Rouen.

Je tiens à remercier tout particulièrement les personnes ayant permis l'aboutissement de mes travaux de thèse :

Je remercie donc vivement Jean-Pierre Pécuchet, professeur à l'INSA de Rouen de m'avoir proposé ce sujet de thèse et de m'avoir encadré conjointement pendant celle-ci. Je remercie Catherine Barry et Habib Abdulrab pour leurs nombreux conseils. Je remercie également Mhamed Itmi pour son aide en matière de simulation. Je les remercie de leur disponibilité, de leur réactivité à la lecture et à la correction de mes documents, et de leur soutien.

Mes remerciements iront naturellement vers tous ceux qui ont accepté avec bienveillance de participer au jury de thèse :

Je remercie Jean-Marc Ogier, Professeur à l'université de La Rochelle et Philippe Mathieu, Professeur à l'université de Lille, d'avoir accepté de rapporter avec soin ce mémoire. Je salue également Thierry Paquet, Professeur à l'Université de Rouen pour avoir présidé le jury.

Je remercie Nathalie Chaignaud, maître de conférences à l'INSA, de sa participation au jury et d'avoir manifesté un grand intérêt à mes travaux et pour son soutien.

J'adresse mes remerciements à toutes les personnes de l'équipe PSI et aux participants du groupe de travail SMA de l'Université de Rouen, qui ont contribué de près ou de loin à ce travail de recherche. Je tiens à remercier en particulier Samuel Morisset, doctorant CIFRE chez ARAKNYDE et rattaché au PSI, pour son aide et sa disponibilité lors de la rédaction de cette thèse.

Mes pensées vont aussi vers Nicolas Delestre, maître de conférence à l'INSA de Rouen qui m'a accordé son temps pour de nombreuses discussions techniques et de multiples réunions improvisées

Carole Tambrun, Sylvianne Henocq et Nathalie Raffray, secrétaires de l'équipe, pour leur accueil et leur gentillesse. Je remercie vivement également tous ceux qui ont contribué à développer une ambiance de travail agréable. En particulier, un grand merci à tous mes compagnons au sein du laboratoire: Gaetan et Patrick, ainsi que les nombreux stagiaires qui sont passés par le laboratoire et qui ont contribué à la mise au point de nos travaux. A tous ceux qui n'ont pas encore terminé, je souhaite une agréable fin de thèse.

Enfin un grand merci à tous mes amis qui m'ont encouragé de près ou de loin pendant la fin de ma thèse, je pense particulièrement à tous les membres de ma famille, à ma femme et ma petite fille à qui j'ai volé beaucoup de temps.

RESUME

Cette thèse s'inscrit dans le cadre d'un projet de recherche industriel. Elle présente la modélisation et la conception d'un logiciel générique orienté agent. Il se base sur un modèle organisationnel qui propose la gestion du cycle de vie d'un ensemble d'entités autonomes, les agents, qui tentent de réaliser des rôles en fonction des *compétences* qu'ils possèdent. La décomposition de *rôles* en *tâches* permet aux agents de n'exécuter que certaines parties de leurs rôles, en demandant à d'autres agents la réalisation des tâches manquantes par délégation. La particularité de notre modèle repose sur l'utilisation d'un système d'échange d'*énergie*, véritable moteur engendrant la dynamique et l'*auto-organisation* du système. Chaque agent oriente ses actions de façon à maximiser son énergie. La contrainte énergétique favorise l'*émergence* de comportements plus efficaces en éliminant les agents inaptes ou inefficaces. Aux agents normaux, dits réactifs, s'ajoutent les agents cognitifs, plus élaborés, bénéficiant de compétences particulières. Certains agents sont pourvus de facultés d'*apprentissage* et pourront donc acquérir de nouvelles compétences. L'intérêt du logiciel est de permettre le développement rapide d'applications multi-agents, et de bénéficier de la découpe intuitive et performante des rôles pour autoriser un maximum de modularité, de spécialisation des agents et d'évolution. Pour résumer, c'est une plate-forme de développement de programmes multi-agents, intégrant les technologies réseau les plus récentes. Ses champs d'applications sont la simulation, l'informatique autorégulée, la supervision, la résolution de problèmes…. Nous présentons à titre d'exemple la première application industrielle de notre étude : un *simulateur* pour la formation opérationnelle des officiers de la Sécurité Civile.

MOTS-CLES

Agent, rôle, compétence, tâche, middleware, émergence, auto-organisation, simulation, formation, sécurité civile, modèle organisationnel

TABLE DES MATIERES

Chapitre I. Intelligence Artificielle et Intelligence Artificielle Distribuée 12
I.1. La naissance de l'I.A. 12
I.2. Positionnement de l'intelligence artificielle 15
I.2.1. Positionnement de l'I.A. par rapport aux domaines de recherche connexes 16
I.2.2. Notre positionnement par rapport à l'IA 17
I.3. Introduction à l'IAD 18
I.4. La complexité et l'I.A. 20
I.5. Systémique 21
I.6. Emergence d'auto-organisation et émergence de chaos 22
I.6.1. Vers des systèmes autonomes et clos 24
I.6.2. Les systèmes autopoiétiques 25
I.6.3. Notion d'émergence 25
I.6.4. Exemples d'émergence 26
I.6.5. Critique de l'émergence 29
I.6.6. Intérêt de cette approche 29
I.6.7. Une tentative de formalisation de l'émergence 30
I.6.8. Dynamique et Chaos 31
I.6.9. Notion d'attracteur et de système chaotique 34
I.7. Conclusion 35
Chapitre II. Agents et SMA 36
II.1. Introduction 36
II.2. Les agents : définitions 39
II.2.1. Définition de Erceau 39
II.2.2. Définition de Ferber 39
II.2.3. Définition de Maes 39
II.3. D'autres types d'agents 40
II.4. Cas particulier des animats 41
II.5. Typologie des agents 42
II.5.1. Agents réactifs 43
II.5.2. Agents cognitifs 43
II.5.3. Agents hybrides 44
II.6. Le rôle 44
II.7. Le comportement 45
II.7.1. Modélisation et représentation du comportement 46
II.7.2. Modélisation en fonction du type d'agent 47
II.7.3. Prévision d'un comportement 47
II.8. L'environnement 48
II.9. L'autonomie 50
II.9.1. L'autonomie par rapport à ses propres buts 50
II.9.2. L'autonomie par rapport à ses motivations 50
II.9.3. La vie artificielle 51
II.10. L'agent comme entité d'un système 51
II.10.1. Définition d'un SMA 51
II.10.2. Vers des systèmes multi-agents adaptatifs 52
II.10.3. Inconvénients des SMA 55
II.11. Interaction et coopération 56
II.12. Coordination d'actions 58
II.12.1. Une coordination sans communication 60
II.12.2. Le comportement de grégarisme 60
II.13. Conclusion 62

Chapitre III. Méthodologies et modèles SMA 63
III.1. Avant-Propos 63
III.2. Intérêt d'une méthodologie 64
III.3. Les méthodologies d'analyse des SMA 64
III.4. MAS et AMAS 66
III.5. Phases de construction des SMA 67
III.6. Comparaisons des méthodes 68
III.7. Méthodes retenues 68
III.7.1. GAIA 69
III.7.2. AALAADIN 70
III.7.3. Cassiopée 73
III.7.4. MaSE 74
III.7.5. VOYELLES 76
III.7.6. SCALA 77
III.8. Vers une méthode de représentation des agents 78
III.9. Conclusion 81
Chapitre IV. Le modèle YAMAM 83
IV.1. Avant-Propos 83
IV.2. Présentation du modèle YAMAM 84
IV.2.1. Ontologie liée au modèle 85
IV.2.2. Le modèle organisationnel simplifié 86
IV.2.3. Vers un modèle organisationnel complet 87
IV.2.4. Des systèmes régies par l'énergie 95
IV.2.5. Application de la notion d'énergie à notre modèle d'organisation 96
IV.2.6. Réutilisation du concept dans notre modèle 99
IV.2.7. La coordination entre les agents 102
IV.2.8. Les groupes d'agents 104
IV.2.9. Les objets et les ressources 105
IV.2.10. L'environnement 105
IV.3. Méthodologies applicables au modèle YAMAM 106
IV.4. Conclusion 106
Chapitre V. Etude comparative des plates-formes existantes 108
V.1. Avant-Propos 108
V.2. Un standard international 109
V.3. Caractéristiques requises 110
V.4. Etude des produits retenus 111
V.5. Conclusion 116
Chapitre VI. Choix technologiques 118
VI.1. Introduction 118
VI.2. Recueil des fonctionnalités 119
VI.3. Comparaison de J2EE et .Net 120
VI.3.1. Architecture J2EE pour les services web 120
VI.3.2. Architecture .NET pour les services web 121
VI.4. Le langage de développement 122
VI.4.1. Les avantages de Java 123
VI.4.2. Caractéristiques 124
VI.5. La distribution 125
VI.5.1. RMI 126
VI.5.2. RMI over IIOP 126
VI.5.3. CORBA 127
VI.6. Les JAVA Beans 131
VI.7. Les Entreprise JavaBeans 132
VI.8. XML 134
VI.9. Sécurisation des communications 135

VI.9.1. Encodage SSL ... 136
VI.9.2. Négociation SSL ... 137
VI.9.3. Principales vulnérabilités du SSL ... 137
VI.9.4. Cryptage des données ... 137
VI.9.5. L'authentification des protagonistes ... 138
VI.10. Les Web Services ... 138
VI.10.1. SOAP ... 139
VI.10.2. WSDL ... 140
VI.10.3. UDDI ... 140
VI.10.4. Création d'un service (côté serveur) ... 140
VI.10.5. Utilisation d'un service (côté client) ... 141
VI.10.6. Sécurité des web services ... 141
VI.10.7. Choix d'une architecture de serveur web ... 141
VI.11. La communication et le partage de ressources ... 142
VI.12. Le grid computing ... 144
VI.13. SMP ... 146
VI.14. L'Autonomic Computing ... 146
VI.15. Le P2P ... 147
VI.15.1. Une nouvelle philosophie d'architecture ... 147
VI.15.2. Avantages et inconvénients ... 149
VI.15.3. Vers des protocoles P2P standard ? ... 150
VI.16. Mobilité et Persistance des Threads ... 151
VI.16.1. Mobilité ... 151
VI.16.2. Persistance ... 151
VI.16.3. Une solution élégante ... 151
VI.17. Les techniques d'apprentissage ... 152
VI.17.1. Raisonnement à base de règles (rule-based reasoning) ... 153
VI.17.2. Réseaux Bayésiens ... 153
VI.17.3. Réseaux de neurones ... 154
VI.17.4. Q-Learning ... 154
VI.18. Conclusion ... 154
Chapitre VII. Conception du middleware Phoenix ... 156
VII.1. Introduction ... 156
VII.2. Fonctionnalités globales du Middleware ... 156
VII.3. Positionnement de la plate-forme ... 157
VII.3.1. Caractéristiques générales : ... 157
VII.3.2. Configuration logicielle et matérielle : ... 157
VII.4. Conception de la plateforme ... 158
VII.4.1. Vue d'ensemble des packages ... 158
VII.4.2. Modélisation de la plateforme ... 158
VII.4.3. Modélisation des agents ... 162
VII.4.4. Les messages ... 164
VII.4.5. La boite aux lettres ... 169
VII.4.6. Le market ... 169
VII.4.7. Communication inter-plates-formes ... 171
VII.4.8. Migration d'agents ... 171
VII.4.9. Implémentation de la migration ... 175
VII.4.10. Une architecture P2P ... 177
VII.4.11. Utilisation des WebServices ... 178
VII.5. Le Rôle ... 183
VII.5.1. Le nombre d'instances d'un rôle ... 183
VII.5.2. La délégation de tâches ... 183
VII.5.3. Choix d'un rôle par un agent ... 185
VII.5.4. Exécution d'un rôle par un agent ... 186
VII.5.5. Estimation du prix de vente d'un rôle ... 187
VII.5.6. Choix d'un rôle par un agent cognitif ... 187
VII.6. Les tâches ... 189

VII.6.1. Définition du package Script dans Phoenix 190
VII.6.2. Lancement d'un script 190
VII.6.3. Composition d'un script 191
VII.6.4. Analyse d'un rôle 193
VII.6.5. Interprétation du rôle 195
VII.7. Les compétences 200
VII.7.1. Modélisation et utilisation 200
VII.7.2. L'acquisition de compétences 201
VII.7.3. La recherche de compétence : un problème sémantique ? 202
VII.8. L'état mental 202
VII.8.1. Les mémoires impliquées dans l'état mental 203
VII.8.2. Apprentissage 205
VII.9. Fonctionnement de la plate-forme 206
VII.9.1. Agent Noyau 206
VII.9.2. Energie 206
VII.9.3. Ressource 207
VII.9.4. Environnement 208
VII.9.5. Réseau d'accointances 210
VII.9.6. Le market 210
VII.9.7. Le modèle graphique 210
VII.9.8. Délégation 210
VII.9.9. Un mode de fonctionnement distribué 211
VII.9.10. Sécurisation des transactions 211
VII.9.11. Certification d'une compétence ou d'un agent 212
VII.9.12. La persistance 214
VII.10. Une passerelle vers le Grid 220
VII.10.1. Les services de gestion 221
VII.10.2. Un protocole de transfert de fichiers 221
VII.10.3. Accès au Grid par les Web Services 222
VII.10.4. Adaptation de Phoenix et du modèle FIPA à l'accès au Grid 222
VII.11. Conclusion 222
Chapitre VIII. Une application professionnelle du Middleware Phoenix : Le projet ANTIC 225
VIII.1. Avant-Propos 225
VIII.2. Terminologie 225
VIII.3. Etude de l'existant - simulation pour la sécurité 226
VIII.3.1. Système d'information pour la gestion de crise 227
VIII.3.2. Système d'aide à la gestion de crise 227
VIII.3.3. Système de réalité virtuelle pour la formation aux plans d'intervention 227
VIII.3.4. Système de détection de feux 228
VIII.3.5. Système de communication de groupe 228
VIII.3.6. Système de météorologie 229
VIII.3.7. Système d'aide à la décision 229
VIII.3.8. Système de main courante 229
VIII.4. ANTIC I 229
VIII.4.1. Les partenaires du projet 230
VIII.4.2. Technologies mises en oeuvre 230
VIII.4.3. Ontologie ANTIC I 230
VIII.4.4. Vues et définitions des liens sémantiques 231
VIII.4.5. Les composantes du système 231
VIII.4.6. Le simulateur 234
VIII.4.7. Fonctionnement d'un scénario 235
VIII.4.8. Perspective de ANTIC I 237
VIII.5. Le projet ANTIC II 237
VIII.5.1. Les partenaires du projet 238
VIII.5.2. Solutions techniques retenues 238
VIII.5.3. Utilisation du modèle YAMAM pour le projet ANTIC2 239
VIII.5.4. Choix d'un scénario 240

VIII.5.5. Mode d'utilisation 241
VIII.5.6. Architecture de communication entre Serpolet et Phoenix 241
VIII.5.7. Implémentation 242
VIII.5.8. Fonctionnement du logiciel 246
VIII.6. Conclusion 248

INTRODUCTION GENERALE

Avant-Propos et motivation

De nombreux domaines utilisent la simulation pour essayer d'expliquer et de prévoir le comportement des systèmes complexes. Grâce à sa puissance de traitement, l'ordinateur est un moyen adapté et efficace pour reproduire un processus du monde réel modélisé. Le modèle du monde ainsi représenté est volontairement réducteur. Souvent, ce processus est formalisé par des équations complexes qui décrivent le comportement du système. On parle alors de simulation numérique. L'informatique permet par exemple d'appliquer des lois physiques pour simuler un réacteur nucléaire avec une précision acceptable. Un tel outil de simulation est utilisé lorsque l'expérimentation est coûteuse, contraignante, voire impossible.

Les techniques de simulation numériques ne sont cependant pas toujours de nature à permettre l'étude des systèmes complexes comme l'évolution des colonies d'insectes. Depuis quelques années, la science s'intéresse à une autre approche : la modélisation multi-agents. Cette approche consiste à représenter un système avec des entités plus ou moins autonomes, les agents, qui interagissent. Un agent possède des caractéristiques individuelles et un comportement particulier, motivé par des buts et conditionné en partie par la perception de son environnement. On considère alors l'évolution globale du système est la résultante de la composition des comportements individuels. On étudie donc au niveau macro un système dont l'évolution est le résultat d'interactions au niveau micro. L'intérêt d'une simulation multi-agents est d'étudier comment, à partir d'interactions très simples, on peut observer l'émergence d'un comportement global du système. Cette notion d'émergence fait toujours l'objet de recherches.

Ce domaine souffre cependant d'un manque de formalisme au niveau méthodologique et au niveau des techniques de simulations utilisées et de la représentation des interactions. Il n'existe pas l'équivalent des lois physiques pour modéliser le comportement d'une entité individuelle. Percevoir, communiquer et agir sont des concepts difficiles à exprimer d'un point de vue mathématique et sont à la charge du programmeur. Un modèle multi-agents ne spécifie pas la façon dont il doit être implémenté (par définition du modèle) et cela a une influence considérable sur la pertinence du résultat obtenu. Un modèle théorique sur papier peut donner des résultats très différents selon le programme utilisé pour simuler. La technique et la méthodologie utilisées pour simuler un SMA sont donc indissociables des résultats obtenus. Nous nous sommes intéressés dans cette thèse à la fois aux modèles multi-agents et aux plates-formes multi-agents pour la simulation.

Un modèle et des applications

La simulation trouve des applications dans des domaines variés comme celui de la formation interactive. Elle constitue un outil précieux pour mettre en situation un apprenant. L'avantage direct est de proposer une approche pédagogique interactive sans recourir à une logistique chère et complexe, propre à la situation réelle. La simulation a donc un intérêt

incontestable dans l'apprentissage du pilotage et du commandement. Les militaires utilisent par exemple de nombreux simulateurs pour former leur personnel. La réalité virtuelle permet même d'immerger le stagiaire dans un monde virtuel en trois dimensions, ce qui accroît l'interaction entre l'homme et la machine.

Notre projet de recherche s'inscrit dans un projet industriel d'enseignement et d'évaluation par la simulation. Il a pour objectif de concevoir un simulateur de catastrophe pour la Sécurité Civile qui puisse améliorer qualitativement la formation opérationnelle des officiers Sapeurs-Pompiers. Le logiciel doit apporter une réactivité et une certaine plasticité de façon à ce que la simulation d'une situation à risque ne se déroule pas de manière statique et préprogrammée. Des événements nouveaux peuvent être introduits en cours de simulation et des acteurs doivent pouvoir intégrer un rôle dans la simulation à n'importe quel moment du scénario. Le programme informatique doit donc être capable de prendre en compte ces événements et de s'adapter de lui-même en réaction à un changement de contexte. Le projet impose une approche largement distribuée de manière à ce que les scénarii de simulation puissent être joués par des utilisateurs rassemblés en groupes locaux ou distants. Le laboratoire d'informatique de l'INSA de Rouen est donc chargé d'intégrer cet aspect dynamique pour rendre la simulation plus réelle, moins prévisible et moins contraignante.

Nous souhaitons également que notre système puisse être réutilisé en dehors du cadre de la Sécurité Civile. C'est la raison pour laquelle nous nous sommes fixé l'objectif d'utiliser une couche logicielle générique pour abstraire davantage nos travaux et permettre une bonne réutilisation. Nous adoptons une approche pragmatique consistant à choisir parmi de nombreuses technologies logicielles, celle qui nous pourrait la plus appropriée pour répondre à notre problématique de simulation.

Notre choix s'est porté sur les techniques multi-agents pour ce qui concerne la dynamique comportementale du système de simulation.

Une étude des différentes méthodologies et plates-formes nous a conduit à entreprendre le cœur de notre travail de recherche et d'ingénierie : la mise au point d'un nouvel environnement d'agents.

Nous souhaitons en effet intégrer un système natif permettant d'auto-organiser la plate-forme. Le fonctionnement d'un système est en effet assuré par le résultat des interactions entre les entités le constituant. Ces entités sont socialement dépendantes. Nous considérons qu'un système complexe ne peut s'organiser librement sans un substrat, traduit ici sous la forme d'énergie, qui relie les entités le constituant. Outre la grande modularité, c'est en grande partie ce que notre approche apporte de nouveau scientifiquement.

Nous décrivons dans ce document quel modèle nous avons mis au point et comment sa structure inhérente autorise l'auto-organisation de ses parties. La conception d'un outil logiciel, basé sur ce modèle d'organisation est présentée en détail, ainsi qu'une application professionnelle dans le but de valider son fonctionnement.

Organisation du mémoire

Nous présentons un historique de l'intelligence artificielle et des systèmes distribués dans le premier chapitre et présentons ensuite notre vision du paradigme agent et des systèmes multi-agents. Ces deux premiers chapitres positionnent notre étude par rapport aux grandes

tendances de l'informatique d'aujourd'hui et définissent le vocabulaire employé tout au long de ce document.

Nous exposons dans le chapitre trois un état de l'art sur les modèles et les méthodologies existants. Nous précisons pourquoi nous avons choisi de répondre à notre problématique de simulation avec un nouveau modèle d'organisation d'agents, nommé YAMAM, décrit en détail au chapitre quatre.

Les plates-formes multi-agents disponibles actuellement font l'objet d'une étude dans le chapitre cinq. Nous mettons en exergue leurs caractéristiques et domaines d'application privilégiés et montrons pourquoi aucune d'entre elles ne peut répondre simplement à nos spécifications. Ce constat nous conduit au développement de notre propre plate-forme.

Nous exposons dans les deux chapitres suivants quelles technologies nous pensons adéquates pour le développement de notre propre outil logiciel ainsi que la conception détaillée de notre plate-forme Phoenix.

Le projet ANTIC, pierre angulaire de notre étude, est décrit dans le chapitre huit. Ce projet constitue notre première application industrielle construite sur le logiciel générique Phoenix. ANTIC est présenté dans son ensemble, mais nous insistons sur les fonctionnalités de simulation proprement dites qui mettent en exergue les grands concepts du modèle YAMAM.

Nous concluons avec une synthèse de nos travaux et des perspectives d'évolution de nos recherches.

Chapitre I. Intelligence Artificielle et Intelligence Artificielle Distribuée

« Il serait salutaire de ne pas oublier que si les diverses parcelles de savoir que nous possédons nous rendent assez dissemblables, dans notre infinie ignorance nous sommes tous égaux » – Karl R.Popper.

Résumé : Ce chapitre présente un historique de l'intelligence artificielle et plus particulièrement des systèmes distribués. Il a pour but de positionner notre étude par rapport aux domaines de recherches connexes à l'I.A et l'IAD.

I.1. La naissance de l'I.A.

Dans l'histoire du développement technologique de ces XX^{e} et XXI^{e} siècles, l'informatique et les ordinateurs tiennent une place de choix. Cela est vrai non seulement au point de vue matériel où les immenses progrès de l'électronique ont permis de construire des machines de plus en plus performantes, mais aussi en ce qui concerne les méthodes et les modèles, mettant à disposition des utilisateurs des environnements de plus en plus faciles à utiliser et des outils de plus en plus efficaces et perfectionnés. En à peine plus de cinquante ans, nous sommes passés du premier ordinateur fonctionnant avec des tubes à vide et des cartes perforées, aux transistors et traitements par lots puis aux circuits intégrés et à la programmation. L'évolution a permis aussi le passage du traitement monoprocesseur et monoprocessus au traitement multiprocessus, puis multiprocesseur, parallèle, concurrent et distribué. Pendant ce temps, les méthodes ont évolué du langage machine aux langages procéduraux. Après ces langages procéduraux, sont apparus des langages de plus haut niveau puis des langages de type fonctionnel, logique et objet…

Nous sommes aujourd'hui dans une ère où la communication et l'information jouent des rôles de plus en plus importants dans la vie quotidienne. L'Homme est de plus en plus dépendant de la technologie et développe en conséquence des systèmes de plus en plus rapides, fiables et de plus en plus sophistiqués. La science est devenue sa religion.

L'épistémologie recentrée sur l'étude des systèmes d'information inclus les systèmes artificiels « intelligents » qui ont pour vocation d'assister l'homme dans son travail quotidien. Ce domaine d'étude tend à donner à la machine un comportement « humain », capable notamment de s'adapter à des situations nouvelles et d'apprendre en permanence.

Le problème de la dualité entre le cerveau physique et l'esprit est posé depuis fort longtemps et est un problème typique de la philosophie. La manière d'appréhender les choses est dépendante de l'époque, de la discipline et du chercheur. C'est certainement un des problèmes les plus ardus que se soit posé l'Homme dans la mesure où son domaine de définition et ses caractères ne sont toujours pas clairement identifiés [Cardon, 1996a]. Devant la dimension du problème, certains adoptent une position radicale considérant l'esprit comme une entité impénétrable qu'aucune simulation sur calculateur symbolique ne pourra reproduire [Searle, 1992].

Le terme Intelligence Artificielle (I.A.) a pourtant été proposé en 1956 au congrès de Dartmouth pour désigner un domaine de recherche qui s'ouvrait alors. L'intelligence artificielle peut être définie comme une branche des sciences informatiques qui concerne l'automatisation de « comportement intelligent ». Cette définition suppose qu'on ait préalablement défini le terme d'intelligence. Comme pour préserver l'ambiguïté de la définition, certains définissent l'I.A. comme « une collection de problèmes et de méthodologies étudiées par des chercheurs en I.A ». A la différence de la Philosophie ou de la Psychologie, qui sont aussi concernées par l'étude de l'intelligence, l'I.A. projette de construire des artefacts intelligents. On trouve deux approches : la première consiste à dire qu'une machine sera considérée comme intelligente, si elle produit d'une manière ou d'une autre un résultat ou une décision qui pourrait être attribué à une action humaine. La seconde spécifie qu'une machine est considérée comme intelligente si elle reproduit le fonctionnement d'un être humain.

L'idée de construire un être artificiel doué d'intelligence remonte au moins à L'Iliade [Homere, …] (chant XVIII) où Héphaïstos, le Dieu forgeron, crée des femmes en or qui ont la capacité de parler, de travailler… : « Des servantes s'empressaient pour soutenir le prince, toutes d'or, mais semblables à de jeunes vivantes ; elles ont un esprit dans leur diaphragme ; elles ont la voix, la force, et les immortels leur ont appris à agir. »

L'invention de la machine à calculer de Pascal fût certainement la première construction mécanique produisant des travaux dont on pensait que seul l'homme était capable d'effectuer. Les questions sur la nature de la pensée et sur les processus reproductibles par les machines se posaient déjà à cette époque. Pour Pascal et Descartes, cela montre que l'arithmétique échappe au domaine du raisonnement. Hobbes considère au contraire que le raisonnement n'est jamais qu'un calcul dont la nature mécanique est confirmée par des constructions telles que la machine de Pascal. On constate aujourd'hui que les clivages sont toujours d'actualité et que ce vaste débat reste ouvert. Le livre « L'homme-machine » de La Mettrie est publié en 1947 [LaMettrie, 1974]. La thèse défendue est particulièrement matérialiste puisque seule la matière existe, la pensée n'en est qu'une propriété et l'Homme est donc réduit à une machine : « Le corps humain est une machine qui remonte elle-même ses ressorts… ». En 1950, Alan Turing placera dans la faculté humaine à converser, la frontière qui permettra de dire si une machine est intelligente ou non [Turing, 1950] et [Turing, 1983]. Il fut un des pionniers de l'I.A. Il avait commencé sa carrière par des travaux sur la logique mathématique. A cette époque l'I.A. s'était fixée comme objectif la reproduction ou la modélisation du raisonnement humain par des relations logiques. Jusqu'au milieu du XIX^{e} siècle, la logique faisait encore partie de la philosophie et les références à Aristote étaient inévitables. Les mathématiques et la géométrie n'étaient d'ailleurs pas entièrement formalisées ; c'est seulement en 1899 que David Hilbert proposa une formalisation de la géométrie. Il proposa l'année suivante, au 2^{e} congrès international des mathématiciens, de formaliser les mathématiques de telle sorte que tout énoncé mathématique soit décidable. Zermelo proposa en 1906 un système formel (Z) qui, après quelques modifications apportées par Fraenkel en 1922 (ZF), permettra de décrire la quasi-totalité des mathématiques telles qu'elles se font encore actuellement.

Mais avant cela, pour De Morgan (1806-1871) et Boole (1815-1864), la logique migrait dans le champ d'investigation des mathématiques et les travaux sur les propositions logiques commençaient réellement. Frege (1848-1925) fondait la science des langages formels. Un système formel peut être décrit par un langage, des axiomes et des règles. Le langage correspond à un alphabet et des règles permettant de construire des expressions bien formées. Les axiomes constituent un sous-ensemble de l'ensemble des expressions. Les règles permettent d'inférer des expressions à partir d'autres.

Les premiers systèmes à I.A. utilisaient cette logique pour formaliser un domaine d'expérience (médecine, géologie…). Les expressions bien formées codent les expressions pertinentes du domaine, qu'elles soient vraies ou non, les axiomes correspondent aux vérités de base et les règles d'inférence permettent de déduire toutes les vérités du domaine et elles seules. A cette époque, l'intelligence artificielle était indissociable de la formalisation. La raison avancée était qu'un ordinateur fonctionnait en binaire et ne pouvait donc traiter que des suites de 0 et de 1, c'est-à-dire des données syntaxiques. Les chercheurs en I.A. considéraient que les comportements humains ne sont rien d'autre que le résultat d'un calcul portant sur des données plus ou moins complexes et que tout calcul pouvait être simulé par une machine de Turing universelle.

Jusqu'aux années 40, cette thèse était de surcroît purement théorique car on ne pouvait valider cela sur des machines réelles. En 1833 Charles Babbage avait construit une « machine à différence » capable d'effectuer des séquences d'opérations arithmétiques. En 1842, il entreprend de construire la « machine analytique » dans laquelle le contrôle du calcul est programmé et peut dépendre de résultats intermédiaires grâce à des instructions de branchement conditionnel. Malheureusement cette machine, qui nécessitait plus de 50.000 pièces mécaniques de haute précision, ne put jamais être construite. Les limites de la mécanique étaient atteintes. En 1945, les Etats-Unis achevaient la construction de l'ENIAC, un des derniers grands calculateurs électriques qui pesait 30 tonnes et occupait une surface de 160 m^2. Cette année là, John Von Neumann décrivait dans un texte d'une dizaine de pages les grands principes qui sont encore à la base des ordinateurs actuels. Les premiers transistors sont mis au point en 1947 et remplacent les lampes à vide dans les années 50. Après 1950, les progrès de la technologie électrique, puis électronique ont permis de construire les premiers ordinateurs. Le tout premier langage de programmation, comportant une cinquantaine d'instructions fut conçu par Alan Turing [Hodges, 1988]. L'ordinateur Mark1 les traduisait en binaire. Le langage FORTRAN fut écrit entre 1953 et 1956. Le LISP écrit par McCarthy et considéré comme le premier vrai langage de l'I.A, apparu aussi en 1956. Puis l'ALGOL et le COBOL apparurent en 1960, suivis par le PL1 et le PASCAL en 1964. Le PROLOG, deuxième grand langage de l'I.A., fut conçu par Colmerauer et Kowalski et apparut dix ans plus tard.

Grâce à ces progrès rapides, les premières réalisations logicielles voient le jour. En 1956, Newell, Shaw et Simon créent un démonstrateur automatique de théorème : « Logic Theorist ». Cette même année, fut présenté le traducteur Anglais-Russe. Deux ans plus tard, Newell et Simon prédisent qu'avant dix ans, l'intelligence artificielle pourra battre le champion du monde d'échecs et pourra démontrer un théorème important en mathématique. En 1959, les mêmes Newell et Simon donnent naissance au GPS (General Problem Solving), qui s'attaque à la résolution de problème en général et non dans quelques domaines particuliers.

Cette première décennie a vu la réalisation de nombreux travaux en I.A. mais les résultats sont néanmoins décevants et pas tout à fait en phase avec les déclarations très optimistes des chercheurs. Parallèlement aux constatations d'échecs des chercheurs, le philosophe américain Hubert Dreyfus publie son livre « Intelligence Artificielle – Mythes et limites » [Dreyfus, 1984] qui présente son analyse très critique sur le développement de l'I.A.

Face à l'explosion combinatoire souvent exponentielle des problèmes à traiter, la communauté de l'I.A. dégage un certain nombre de grands principes pour faire évoluer son domaine. Un algorithme doit

- Incorporer des modules d'apprentissage pour améliorer les performances
- Combiner des approches combinatoires et heuristiques

- Distinguer la connaissance sur le problème et la méthode de résolution.

Les années 60 et 70 voient la réalisation de projets beaucoup plus ciblés et de meilleurs principes méthodologiques, supportés en parallèle par les progrès de la miniaturisation et l'augmentation de la puissance des ordinateurs. En 1970, Terry Winograd conçoit le logiciel SCHRDLU qui simulait la manipulation de blocs géométriques et générait automatiquement des plans. On vit à cette époque l'apparition des systèmes experts parmi lesquels DENDRAL en 1969 qui analysait les résultats d'une spectrographie de masse et MYCIN en 1974 qui se spécialisait dans le diagnostic médical de maladies infectieuses. Un peu plus tard, de nouvelles voies de recherche s'ouvrent en parallèle sur les réseaux de neurones artificiels et sur l'intelligence artificielle distribuée (IAD) qui proposait de découpler les systèmes en plusieurs modules de connaissance et de traitement. C'était la naissance des BlackBoards. Quelques soient les spécialités de recherche, les résultats arrivent tout doucement mais les grands espoirs placés dans l'I.A. aboutissent quasiment tous à des déceptions malgré l'événement du 29 août 1994 où le programme CRESS GENIUS 2.0 bat le champion du monde Gary Kasparov. Ce constat d'échec des systèmes à base de logique formelle a conduit à l'émergence d'approches radicalement différentes, basées sur des concepts d'organisation dynamique qui font l'objet de recherches actives de nos jours.

Aujourd'hui les domaines de l'intelligence artificielle sont les systèmes experts, le calcul formel, la représentation des connaissances, la simulation de raisonnement humain, le traitement du langage naturel, la résolution de problèmes, la reconnaissance de la parole, la reconnaissance de l'écriture, la reconnaissance des visages, la robotique, l'apprentissage, les réseaux neuronaux et les systèmes complexes adaptatifs. Ces derniers regroupent dans un seul vocable les algorithmes génétiques, les SMA et les modèles de vie artificielle. Il s'agit d'étudier comment les populations soumises à des lois simples et naturelles convergent naturellement vers des formes organisées. Les systèmes multi-agents adaptatifs se situent au carrefour de nombreuses disciplines. Les deux plus importants sont l'IAD et la vie artificielle que nous allons étudier dans la prochaine section. Les SMA représentent avec les réseaux de neurones et les algorithmes génétiques les nouveaux espoirs des chercheurs en intelligence artificielle. Nos travaux se situent précisément dans cette voie.

I.2. Positionnement de l'intelligence artificielle

« L'intelligence artificielle appartient au domaine de l'informatique et concerne les approches informatiques ayant pour but de faire apparaître un comportement intelligent »[1]. Cette définition fait évidemment elle-même référence au terme « intelligence » qui peut être défini ainsi : extrait du *nouveau Petit Robert*, 1993 :

1. *Faculté de connaître, de comprendre.*
2. *L'ensemble des fonctions mentales ayant pour objet la connaissance conceptuelle et rationnelle (abstraction, conception, entendement, intellect).*
3. *Aptitude (d'un être vivant) à s'adapter à des situations nouvelles, à découvrir des solutions aux difficultés qu'il rencontre.*
4. *Qualité de l'esprit qui comprend et s'adapte facilement; caractère d'une personne intelligente (capacité, discernement, jugement, perspicacité, réflexion).*
5. *Action de s'entendre mutuellement; son résultat.*

L'intelligence artificielle demeure un vaste chantier non balisé et au cahier des charges particulièrement flou. Ce chantier puise même certaines de ses grandes orientations dans d'autres disciplines qui approchent le problème de façons très différentes. L'approche

[1] Définition extraite du Web Dictionary of Cybernetics and Systems de F. Heylighen.

psychanalytique et l'approche des neurosciences s'opposent radicalement sur des problèmes d'ordre conceptuel [Hochmann & Jeannerod, 1996].

La première d'un niveau plus haut s'intéresse à l'étude de comportements issus du système neuronal central. Elle est divisée en plusieurs écoles mais pose globalement son domaine avec une ambiguïté volontaire considérant l'objet de l'étude comme régi par d'obscures forces traduisant des tendances comportementales observables extérieurement. L'autre approche, celle notamment de la neurobiologie reste fondamentalement objective et s'intéresse au niveau presque atomique. Elle considère d'abord les constituants de base du cerveau physique au niveau cellulaire et donc biochimique, pour déterminer des fonctions agrégatives multiples. Elle permet de dresser de véritables cartes cognitives du cerveau physique et donc se fixe l'objectif d'expliquer l'émergence d'un état mental.

L'informaticien s'intéresse plus fortement à l'organisation, et l'intelligence artificielle oriente ses investigations sur le processus de prise de conscience, sur le phénomène générant l'acte mental. Elle rejoint ici une troisième discipline qui est la philosophie, le passage de la matière à l'esprit.

Nous allons tenter dans les prochains paragraphes de positionner l'I.A. par rapport à ces deux domaines de recherche.

I.2.1. Positionnement de l'I.A. par rapport aux domaines de recherche connexes

L'approche des neurosciences est très objective [Delacour, 2001]. Elle consiste à dénombrer, classer et localiser les éléments physiques fondamentaux de ce qui constitue le phénomène de conscience localisé dans le cerveau. Cette approche est dite montante car elle part des constituants de base, au niveau cellulaire, pour en déterminer des observations fines et des fonctions agrégatives multiples. Nous faisons ici allusion aux cartes cognitives et à leurs rôles ainsi qu'aux fonctions des relations entre ces agrégations. Cette approche est très scientifique et ne propose pas de manipuler le processus de prise de conscience par des symboles mais revendique plutôt une approche sub-symbolique. Dans l'histoire de l'I.A., il y eut bien entendu plusieurs courants. Les premiers approchaient l'I.A. avec des systèmes de manipulation de symboles à grande dose de logique formelle, et proposaient des applications liées à la résolution de problèmes. Ensuite des approches neuronales consistaient à plagier le neurone biologique, ce qui par voie de conséquence, était très proche des neurosciences. L'approche des SMA, outre le phénomène de mode, propose de mettre en exergue l'aspect auto-organisateur des systèmes distribués complexes et postule une émergence d'organisation [Varela, 1989b] et de sens de part les multiples interactions des composants du système. Il est possible d'adopter aussi une approche montante ; les SMA ont beaucoup de points communs avec les réseaux neuronaux et moins avec les systèmes symboliques.

L'I.A. est un domaine de recherche fortement lié à celui de la psychologie cognitive. Notre approche résolument systémique et émergentiste[2] peut être positionnée par rapport au behaviorisme, au cognitivisme, au connexionnisme et à l'interactionnisme [Weil-Barais, 2001]. Elle est tournée en grande partie vers le connexionnisme, bien que cela ne signifie pas pour autant réfuter les thèses de l'interactionnisme, qui nous semblent complémentaires. Il est

[2] [Hempel & Oppenheim, 1948] proposent une définition de l'émergence comme « L'occurrence d'une caractéristique W d'un objet w est émergente relativement à une théorie T, à une relation de partie PT, à une classe d'attributs G des parties, si cette occurrence ne peut pas être déduite des parties de W données par PT respectivement à tous les attributs G ». [Teller, 1992], quant à lui propose : « Une propriété est émergente si elle n'est pas réductible aux propriétés non relationnelles de ses parties ».

vrai que l'homme vit en société et qu'il se développe grâce aux interactions avec ses congénères. L'environnement influe sur l'homme et l'homme influe sur son environnement. Malgré tout, il perçoit les choses de son environnement par des signaux qui sont des stimuli reçus au niveau du cortex cérébral. C'est ensuite, que le système neuronal réagit et qu'une nouvelle organisation émerge des interactions des neurones qui produit du sens. C'est là une position assez radicale.

I.2.2. Notre positionnement par rapport à l'IA

Se positionner dans l'intelligence artificielle contraint à se positionner par rapport aux différentes disciplines afférentes à l'I.A. Bien que chaque approche présente un certain nombre d'intérêts, nous allons choisir de nous atteler à la tâche difficile de l'émergence d'intelligence par auto-organisation[3]. Cela sera considéré comme un axiome dans toute notre recherche.

Cette forme d'émergence s'appuie simultanément sur des concepts neurobiologiques et psychanalytiques en les unifiant sous leur aspect systémique et organisationnel. Nous ne pouvons évacuer en effet l'aspect organisationnel puisque l'informatique, dans son essence même, s'intéresse à l'organisation et l'interaction de composants.

Nous nous intéressons aussi de près à la vie artificielle [Meyer, 1998], c'est-à-dire aux animats. L'approche Animat [Pfeifer & al., 1998] recherche dans quelle mesure il est possible de comprendre la cognition humaine en l'abordant dans une perspective évolutionniste. Plus précisément, elle consiste à étudier en quoi l'intelligence de l'homme s'explique à partir des processus adaptatifs les plus simples hérités des animaux. Pour ce faire, elle conçoit ou utilise des animats - c'est-à-dire des animaux simulés sur ordinateur ou des robots autonomes - dont les lois de fonctionnement sont inspirées de la biologie, et qui doivent "survivre" dans des environnements plus ou moins imprévisibles et dangereux. De tels animats doivent être équipés de senseurs et d'effecteurs adaptés aux défis que leur pose l'environnement, ainsi que d'une structure[4] de contrôle qui leur permette de choisir quelle action effectuer et à quel moment. L'AnimatLab[5] s'intéresse à trois techniques adaptatives qui sont les réseaux de neurones [Doncieux, 1999], les systèmes de classeurs [Gérard & al., 2001] et les algorithmes évolutionnistes [Flacher & al., 2002]

Notre désir sera dans un premier temps d'approcher une forme d'organisation à reconformation dynamique dans un système informatique. Les systèmes multi-agents adaptifs répondent parfaitement à notre présente problématique. Dans le cadre de ce travail, nous allons nous attacher à construire un système primaire auto-organisé possédant des propriétés de reconformation dynamique. Cela constitue une première étape déjà considérable. Ce travail est résolument tourné vers l'I.A. distribuée qui est présentée dans le prochain paragraphe.

[3] L'auto-organisation est un terme qui caractérise des systèmes réagissant « à des effets aléatoires de l'environnement de façon à augmenter leurs capacités de réponses à de nouveaux stimuli, c'est-à-dire leurs capacités de régulation » [Atlan, 1979].

[4] Une structure évoque une organisation des parties d'un système, qui lui donne sa cohérence et en est la caractéristique permanente ; par exemple la structure d'un État ou d'une entreprise. Ces organisations peuvent être vues comme des systèmes complexes considérés dans leurs éléments fondamentaux. Le sens philosophique parle d'un ensemble abstrait, ordonné et autonome d'éléments interdépendants aux rapports régis par des lois, faisant fonction de modèle d'intelligibilité des objets étudiés, dans les diverses sciences humaines.

[5] Vous pourrez retrouver plus d'information sur le site web de l'AnimatLab sur http://animatlab.lip6.fr/

I.3. Introduction à l'IAD

Sur le plan théorique, l'intelligence artificielle des années 70 à 90 s'était engagée sur une voie trop réductrice. Les systèmes experts ont marqué leur époque en exploitant massivement des relations logiques pour donner l'illusion d'intelligence alors que, fondamentalement, le système n'effectuait qu'une inférence sur de la connaissance structurée. Ce vaste domaine de recherche est bien sur divisé en plusieurs sous-domaines de recherche qui contribuent notamment à entretenir le clivage conceptuel des différentes approches.

Un de ces sous-domaines, très en vogue aujourd'hui est l'Intelligence Artificielle Distribuée (IAD). En effet, certains problèmes informatiques ne peuvent plus être résolus grâce à des systèmes possédant une expertise centralisée aussi sophistiquée soit-elle. Une approche de solution propose de répartir les éléments de connaissances sur un certain nombre d'entités distinctes. L'IAD a pour objectif de résoudre certains problèmes en utilisant la collaboration de telles entités [Gasser, 1992]. L'IAD se décompose en plusieurs sous-domaines qui sont la résolution distribuée de problèmes, les systèmes multi-agents et l'intelligence artificielle parallèle. L'un était souvent complémentaire de l'autre.

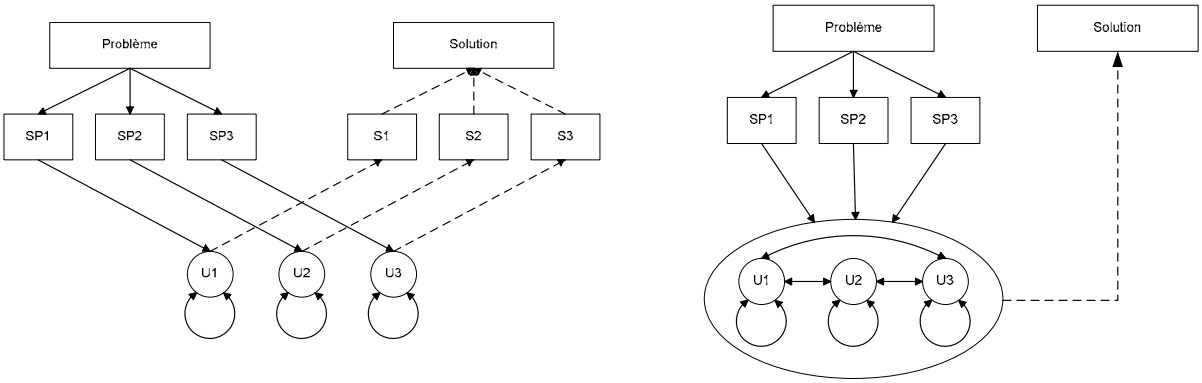
Les principes de base de l'IAD et par extension, du paradigme multi-agents, consistent en la distribution des connaissances et des informations nécessaires à la résolution d'un problème parmi les différentes composantes en interaction d'un système [Chevrier, 1993]. Le terme d'Intelligence Artificielle Décentralisée a également été utilisé pour désigner cette méthodologie permettant de trouver une solution à un problème général. Cette branche de l'IAD s'occupe donc de la résolution de problèmes spécifiques en choisissant de répartir sur plusieurs modules les connaissances et les éléments nécessaires à la résolution du problème. Cette branche fait suite historiquement aux systèmes de tableaux noirs [Erman & al., 1980] et [Hayes-Roth, 1985]. Les systèmes issus de ce type d'architecture conservent un certain degré de centralisation ne serait ce que dans le but global de résolution du problème posé.

Il existe une branche de l'IAD, qui est plutôt destinée à modéliser des processus sociaux ou des artefacts. C'est celle dans laquelle se positionne notre travail. Les composants de ces systèmes sont des entités plus ou moins intelligentes désignées par le terme "agent". Un agent est une des entités actives d'un système. Ce terme d'entité active sert à signaler le fait qu'un agent ne correspond pas à la notion de module statique, mais plutôt à une notion d'objets concurrents et autonomes ayant le propre contrôle de leur activité [Hewitt, 1977]. Ce premier concept constitue l'élément de base du paradigme multi-agents, nous y consacrons donc entièrement le prochain chapitre.

Par analogie avec les sciences sociales, le terme de société est utilisé pour désigner l'ensemble des agents constituant un système multi-agents. C'est ici que réside pour nous le principal intérêt du paradigme multi-agents : ne pas uniquement concevoir des entités intelligentes, mais aussi mettre ces entités en relation de manière "intelligente". Cette approche transparaît également au travers de deux autres concepts communément employés dans le domaine : les interactions et l'organisation des agents.

Le terme interaction désigne l'enchaînement d'échanges d'informations ou d'influences ayant lieu entre des agents. La mise en place de ces interactions permet aux agents de résoudre ensemble un problème grâce à leurs capacités respectives. Ce concept est l'un des aspects les plus importants de l'IAD [Decker & Lesser, 1995] dans la mesure où il permet de relier les agents pour qu'ils constituent un système. L'approche des problèmes est alors telle que leur résolution est au-delà de la complexité de chacun des agents du système mais réside dans la multitude des interactions entre les constituants du système.

Une résolution distribuée de problème va opérer selon le schéma ci-dessous à gauche. Une résolution de problème par un SMA va être traitée par une société d'agents qui vont interagir pour proposer une solution globale. Le schéma de résolution est montré sur la figure de droite :



Mais ce concept d'interaction ne permet de mettre en relation qu'un sous-ensemble des agents composant un système. Il est alors nécessaire d'utiliser un autre concept plus général pour exprimer les relations liant tous les membres de la société : l'organisation.

Le concept d'organisation a pour objectif principal de contrôler et de coordonner l'ensemble des interactions présentes entre les agents de la société, afin de structurer les activités des agents et permettre ainsi au système de se comporter comme un tout efficace et cohérent, capable de conduire à une solution du problème. Le fondement du paradigme multi-agents est évidemment le concept d'agent qui définit les caractéristiques des composants de base d'un système. Mais, afin de permettre une meilleure approche de la conception de systèmes multi-agents, nous pensons qu'il est nécessaire de prendre en compte également les concepts d'interaction et d'organisation dans une approche conceptuelle.

Le constat d'échec après de nombreuses années de recherche en I.A. sur les systèmes experts a montré qu'il est vain de considérer aujourd'hui une forme d'intelligence basée uniquement sur une base de connaissance aussi grande soit-elle. Il semble que l'intelligence ne soit pas une entité façonnable à souhait ni modulable, mais la résultante d'une émergence de propriétés issues d'un système dynamique, réactif et autonome. C'est probablement en cultivant ce caractère d'autonomie organisationnelle que de véritables systèmes intelligents pourront probablement voir le jour. C'est de la multitude d'interactions entre différentes entités autonomes ou semi-autonomes que peut naître une certaine cohérence du système. L'objectif n'est plus d'entrer de « l'intelligence » dans une machine mais de préparer un environnement pour que l'intelligence puisse émerger. Nous sommes dans le cadre de la théorie des systèmes avec toute la complexité inhérente à ses concepts sous-jacents.

Les différents chapitres suivants ont pour but de présenter les caractéristiques respectives de ces concepts d'organisation, d'interaction, d'autonomie… autour desquels s'organisera toute la problématique de ce mémoire. Nous proposons également un dictionnaire ontologique de façon à minimiser les incompréhensions dues aux divergences des définitions de concepts. Nous verrons que l'IAD fait souvent référence à la systémique et de ce concept découle d'autres concepts sous-jacents comme l'émergence d'auto-organisation, le chaos, les attracteurs etc.

I.4. La complexité et l'I.A.

L'homme s'intéresse à l'Intelligence Artificielle depuis longtemps car il est plongé dans un monde intrinsèquement complexe et il se trouve donc confronté à résoudre des problèmes complexes. Cette complexité fait effectivement partie de notre univers et est présente dans les logiciels [Booch, 1994]. L'homme cherche donc à construire des modèles[6] de systèmes, plus faciles à manipuler que les systèmes réels. Il existe quelques principes pour casser la complexité et mieux la maîtriser tels que la décomposition, l'abstraction et la hiérarchisation.

On peut recenser un grand nombre de processus dans la nature que nous ne maîtrisons pas bien. Comment, par exemple, décider d'un trajet optimum pour aller faire du shopping, comment s'orienter dans une pièce mal éclairée, comment se concentrer sur une seule conversation dans la cacophonie d'une soirée ? Aucun de ces problèmes n'a de solution algorithmique simple. Le choix d'une route optimale ou problème du « voyageur de commerce » est un problème np-complet bien connu dont on a bien du mal à briser la complexité (passer d'une complexité exponentielle à une complexité polynomiale ou logarithmique). Les chercheurs en IA ont étudié ces problèmes depuis de nombreuses années. Les recherches dans ce domaine nécessitent souvent la construction de systèmes « intelligents » qui simulent parfois certains aspects du comportement humain. Erman, Lark et Hayes-Roth [Erman & al., 1988] soulignent que les systèmes intelligents diffèrent des systèmes conventionnels par un ensemble d'attributs, qui ne sont pas toujours tous présents :

- Ils suivent des buts qui peuvent changer dans le temps
- Ils incorporent, utilisent et modifient des connaissances
- Ils exploitent divers sous-systèmes ad hoc qui mettent en œuvre un ensemble de méthodes choisies
- Ils interagissent de façon intelligente avec les utilisateurs et d'autres systèmes
- Ils allouent leurs propres ressources et centres d'intérêts.

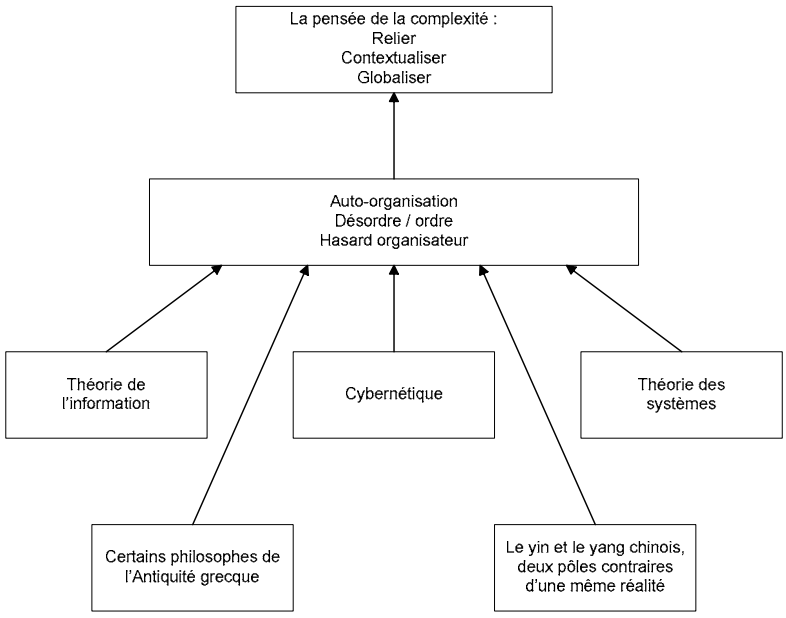
Chacune de ces propriétés est suffisamment complexe pour rendre très difficile la création de systèmes intelligents. Pendant plusieurs années, les recherches se sont orientées sur les systèmes experts et les systèmes black-boards (ou tableaux noirs), ce qui a notamment permis de progresser sur la représentation de la connaissance et sur des architectures de résolution de problèmes. Néanmoins nous étions encore bien éloignés d'une véritable intelligence artificielle. L'évolution des premiers systèmes experts aux modèles de tableaux noirs constituait le pas à franchir vers un premier type de systèmes distribués. Depuis les recherches plus récentes se sont portées sur la distribution de la connaissance et de l'intelligence notamment dans le cadre des systèmes multi-agents qui répondent convenablement aux attributs sus-cités (buts, intérêt, ressources…). Nous reviendrons sur les caractéristiques des SMA dans le prochain chapitre.

[6] « Un modèle est un système abstrait représentatif d'un système réel » (Alain Cardon). Cependant, il faut convenir que toute représentation est simplificatrice et il devient nécessaire de fait, d'exclure l'illusion d'un recensement exhaustif. Un système ne peut être défini d'un point de vue macroscopique par quelques relations globales car ces relations ne rendent pas compte de la genèse de sa structure inhérente. Il est préférable de chercher des techniques et des lois susceptibles de guider la sélection d'agrégats tenus pour pertinents. Tout système est modélisable mais il est impossible de représenter toutes les interactions du système réel. Dans le cadre d'une modélisation informatique, on choisit donc un niveau de granularité adéquat entre fiabilité et précision du modèle d'une part et coût de réalisation et de fonctionnement d'autre part.

I.5. Systémique

Selon la vision matérialiste du monde, qui sous-tend encore plus ou moins directement la plupart de nos décisions, la réalité objective et ultime peut se réduire à des structures matérielles et des échanges d'énergie dans l'espace et le temps. Même si certaines disciplines s'occupant de systèmes[7] complexes, telles que les sciences biologiques, écologiques, sociales, économiques ou cognitives, ont développé chacune des outils conceptuels moins matérialistes et mieux adaptés à leur propre champ de préoccupation, il manque encore une épistémologie commune, générale, bien adaptée à rendre intelligibles tous les systèmes naturels; et ceci malgré l'évidente continuité historique de l'évolution terrestre du stade physico-chimique aux stades biologique, socio-écologique, cognitif puis conscient. Les conflits entre les approches neurophysiologique et psychanalytique du fonctionnement de l'esprit humain sont une démonstration de l'absence de cohérence entre les diverses épistémologies en usage.

Toutefois, depuis une cinquantaine d'années, parallèlement à la tendance à la spécialisation observée dans la plupart des secteurs, se poursuit un effort de recherche de concepts plus larges et plus pertinents pour appréhender les processus fondamentaux à l'oeuvre dans le fonctionnement et l'évolution de la nature. L'approche systémique ou "science des systèmes" a essentiellement pour origine la rencontre, dans les années 1940, entre la biologie et l'électronique qui va se traduire par la naissance de la cybernétique ou étude des régulations chez les êtres vivants et les machines avec idées de boucles, de rétroactions, de régulations (texte fondateur de Wiener, 1948). Von Bertalanffy, biologiste, va exprimer une vision unitaire du monde avec, partout, des systèmes ouverts (1950). Cette approche s'est ensuite élargie, par le développement d'applications à des situations concrètes complexes pour devenir finalement un "mouvement systémique" [Bertalanffy, 1973].



Concrètement, l'approche systémique présente trois aspects :

[7] Alain Cardon défini un système comme « une construction théorique rendant compte d'un certain domaine du réel. Il est composé d'entités distinctes et répond à un principe d'existence qui est défini par son action apparente ». Cette définition permet de représenter, c'est à dire de présenter pour l'observation et l'analyse, un domaine du réel, rendu compréhensible pour l'homme. Les échanges réalisés dans ce système lui assurent une certaine autonomie et lui permettent de subir des modifications plus ou moins profondes dans le temps tout en conservant une certaine permanence.

1. Un aspect scientifique, c'est-à-dire la recherche des régularités (invariants) dans l'évolution des systèmes naturels, en particulier l'apparition spontanée d'ordre dans les systèmes complexes (auto-organisation); dans cette perspective, on étudie les propriétés émergentes globales ("systémiques") qui ne peuvent se réduire aux propriétés des composants du système.
2. Un aspect pratique, à savoir l'incidence de ces connaissances concernant la dynamique des systèmes naturels sur l'intervention et l'action dans des situations complexes: organisation de projets industriels importants, conception et management d'entreprises et d'institutions, ainsi qu'interventions dans des groupes humains (thérapie) et des systèmes sociaux, économiques, politiques et écologiques complexes. Depuis quelques années les connaissances sur les systèmes naturels complexes débouchent également sur la conception et le développement de systèmes complexes artificiels: intelligence artificielle et vie artificielle.
3. L'approche systémique ne se réduit pas à un corps de connaissances scientifiques et d'indications pratiques; elle génère et s'inscrit dans une vision globale du monde conduisant à questionner, consolider, ou modifier les fondements de nos connaissances et de notre façon de les rechercher.

Cette approche est davantage transdisciplinaire qu'interdisciplinaire car elle tente d'aller au-delà des disciplines en recherchant des concepts communs, plus englobants; elle est signalée par des mots-clefs tels que système, structure, interdépendance, communication, interaction, rétroaction, organisation, complexité, émergence, auto-organisation, auto-régulation, auto-production, auto-référence, autonomie… Ces termes seront définis à mesure de leur utilisation.

I.6. Emergence d'auto-organisation et émergence de chaos

Nous avons vu dans les précédents paragraphes quelques notions qui vont nous permettre d'appréhender les théories de l'organisation[8], de l'auto-organisation, de l'émergence et du chaos. Ces notions sont des points clés dans une simulation par SMA. Nous avons élaboré notre discours autour d'une publication de Encyclopédie Philosophique Universelle, Vol. Notions, tome 1, PUF, Paris 1990.

Autour du maître-mot auto-organisation, de 1960 à nos jours, se sont définies, concertées et affinées des conceptions originales, destinées à résoudre des problèmes venus d'horizons divers : chimie des « corps impurs », physique des phénomènes irréversibles, thermodynamique des systèmes loin de l'équilibre, biochimie, biophysique, génétique, ethnologie économique, intelligence artificielle connexionniste, psychologie cognitive, neurophysiologie, éthologie, géométrie fractale, etc. Cette interdisciplinarité est inséparable d'une remise en cause, plus ou moins profonde, d'aspects fondamentaux de la science traditionnelle et de ses divisions. Il est difficile d'en prévoir les conséquences philosophiques. Cependant, l'idée développée par les empiristes classiques, selon laquelle l'habitude (coutume chez Hume) stabiliserait en une structure cognitive fixe le flux d'informations en provenance

[8] Edgar Morin [Morin, 1977] définit l'organisation comme un « agencement de relations entre composants ou individus qui produit une unité, ou système, dotée de qualités inconnues au niveau des composants ou individus. L'organisation lie de façon interrelationnelle des éléments ou événements ou individus divers qui dès lors deviennent les composants d'un tout. Elle assure solidarité et solidité relative, donc assure au système une certaine possibilité de durée en dépit de perturbations aléatoires ».

de l'environnement, serait peut-être une des premières applications de l'auto-organisation en matière de théorie de la connaissance.

Ce paradigme est apparenté à la cybernétique des années 1950. C'est ainsi qu'on découvre que le bruit (défini comme évènement fortuit, dû à la « rencontre de séries causales indépendantes », Cournot) n'est destructeur que pour des systèmes simples ; dans d'autres, il provoque l'apparition d'un ordre, d'une complexité, jusque là contenus dans les *possibles* prédéterminés par ce système (points fixes de Varela). De tels systèmes, dits auto-organisateurs, ont même besoin de bruit pour évoluer. Leur autonomie par rapport à l'environnement [Varela, 1989b], est corollaire de leur caractère ouvert [Atlan, 1979] : ouverts, ils subissent l'influence des évènements extérieurs, et leur autonomie fait que ces évènements sont pour eux du bruit et non des données qu'ils savent déjà traiter. Le rôle de l'observateur est mis en valeur : information, complexité, ordre, sont mesurés par rapport à lui. On pourra se reporter au modèle stochastique de Blackwell-Kendall ou aux automates cellulaires (le jeu de la vie).

Ces systèmes, où est apparu un ordre dû au bruit (order from noise, Von Foerster), présentent une résistance à la destruction, ou hystérésis, qui assure à cet ordre une certaine durabilité. Pour pouvoir survivre au bruit, et donc pour pouvoir l'utiliser à son bénéfice, comme l'avait déjà montré Von Neumann, il est nécessaire qu'un système ait au départ une importante redondance, qui s'exprime sous la forme d'une multiplicité d'éléments au départ indifférenciés (copies), et d'une délocalisation des fonctions (qui doivent pouvoir être assurées par plusieurs circuits). C'est sur ces bases qu'Atlan [Atlan, 1979] élabore le principe de complexité par le bruit. Le bruit diminue la redondance d'un système auto-organisateur, différenciant ses éléments et ses fonctions ; il augmente ainsi la capacité du système à répondre à son environnement spécifique, mais diminue son adaptabilité à d'autres environnements. Susceptible d'être différencié, le système auto-organisateur doit pouvoir périodiquement « se recharger en redondance[9] », pour rétablir un ensemble de possibles. Toute organisation est donc une « suite de désorganisations rattrapées ». Plusieurs des phénomènes étudiés semblent jeter un pont entre la matière soumise à l'entropie[10] et les systèmes vivants : certaines matières hétérogènes sont capables d'une certaine forme d'auto-organisation. Le second principe de la thermodynamique est donc sérieusement ébranlé ; Atlan suggère que des « lois de diminution d'entropie » devraient venir le limiter. A l'intérieur même du vivant, il est possible d'imaginer des modèles où l'environnement oriente les mutations et la complexification des espèces, suivant un processus qui ne serait pas simplement dégénérescence (perte de capacités de régulation).

L'auto-organisation n'est également pas absente du langage. Malgré des résistances d'origines diverses, comme l'organicisme chomskyen (cf Colloque Chomsky-Piaget) et les tabous sur l'étude de l'origine des langues (v. Langues, Origine des —), on voit se dessiner, en linguistique, un courant auto-organisateur, essentiellement parmi les spécialistes de l'évolution et les énonciativistes. Des phénomènes connus depuis longtemps, comme la phonologisation[11], la sémantisation des variantes combinatoires, la grammaticalisation d'une alternance contextuelle, obéissent au processus décrit par Atlan. Par exemple, quand un phonème influence son contexte (nasalisation de voyelle devant consonne nasale), il s'agit de copies, qui assurent une redondance initiale. Mais si l'original disparaît (chute de consonnes

[9] Nous verrons dans le chapitre dédié à notre logiciel adaptatif Phoenix que les compétences associées aux agents sont redondantes sur un réseau.

[10] En physique, l'entropie est une grandeur thermodynamique, fonction d'état d'un système, qui caractérise l'état de désordre de celui-ci.

[11] La phonologie est une branche de la linguistique qui s'attache à décrire les systèmes de phonèmes des langues en termes de différences et de ressemblances fonctionnelles.

finales), c'est une des copies qui va assurer la fonction initiale de l'original, engendrant un nouveau phonème ou génotype (voyelle nasale). A nouveau il va y avoir redondance, et ainsi de suite. Le même type de phénomène s'observe en syntaxe, ainsi, le déplacement de la fonction négative du *ne* latin au *pas* français passe par la locution doublement redondante il n'avance pas, signifiant il n'avance pas d'un pas... (G. Bernard)

Dans un registre différent et moins philosophique, le Web lui-même présente des caractéristiques d'auto-organisation [Flake & al., 2002], en dépit de sa nature décentralisée, non organisée et hétérogène. Des millions d'individus, de cultures, de natures et de buts différents s'y rejoignent et créent des communautés (pages Web, newsgroups…). La structure du Web est basée sur des liens hypertextes qui permettent notamment d'identifier des communautés. Une communauté Web est un ensemble de pages Web qui ont des hyperliens avec d'autres pages dans et en dehors de cette communauté. La nature très dynamique des changements de liens hypertextes provoque des réorganisations en continu des communautés. Cela s'apparente parfaitement à la dynamique d'un système multi-agents.

I.6.1. Vers des systèmes autonomes et clos

L'autopoièse est une théorie systémique de l'autonomie développée par le neurobiologiste chilien Fransisco Varela [Varela, 1989b]. Le système considéré cherche à conserver sa cohésion interne, i.e. son unité. Le système possède un nombre quasi-infini d'états stables internes (valeurs propres) mais c'est au niveau de ses points de couplages avec son environnement que vont se produirent les perturbations qui influenceront le système. Une perturbation fait passer le système d'un état stable à un autre état stable. La *commande* est l'envers conceptuel de l'autonomie.

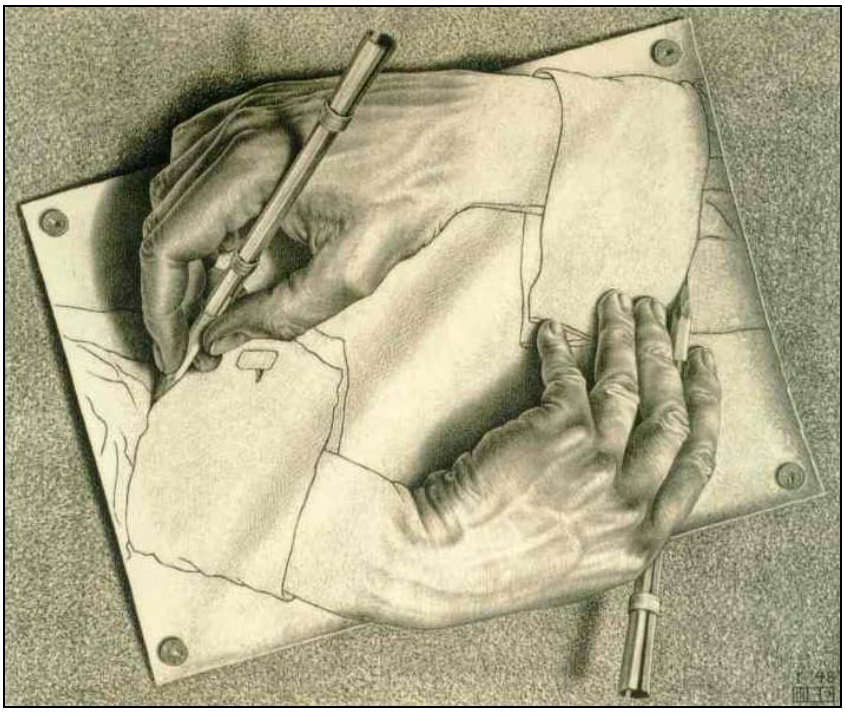


Figure 1 - M.C Escher - Drawing Hands - Lithograph, 1948

La circularité (dessin de M.C. Escher) est un concept un peu semblable à la récursivité mais impliquant plusieurs intervenants. Il est un des principes clés de notre système adaptatif. Il va de soi que comprendre un système n'est pas seulement énumérer ses parties ou définir sa fonction en tant qu'artefact humain. On doit se livrer d'une part à une analyse plus systémique et surtout à considérer non pas les composantes du système mais les interrelations entres ses

composantes. « Les composants peuvent être quelconques aussi longtemps que leurs inter-relations satisfont à un ensemble de conditions souhaitées » [Varela, 1989a]. Tout cela aura bien entendu une influence sur la mise au point de méthodologies d'analyse des SMA, telles que celles présentées dans le prochain chapitre.

I.6.2. Les systèmes autopoiétiques

L'approche de Francisco Varela retient comme principe général que le vivant élémentaire a un caractère mécaniste. Un système autopoiétique de première espèce est basé sur trois caractères. Il s'agit tout d'abord d'un système qui opère mécaniquement. Le système produit et reproduit essentiellement ses propres composants. Il est donc clos et autonome. Toute action sur lui est le fait d'une perturbation interprétée et non d'une entrée propagée. Le système est ensuite considéré par tout observateur comme une entité identifiable et non sécable de vie. Que ce soit le vivant élémentaire ou le vivant organisé, le comportement d'un système autopoiétique ne s'explique pas de façon satisfaisante. Il s'agit enfin d'un système définitivement complexe opérant par couplages innombrables sur lequel il est vain de tenter de réduire le processus pour avancer une explication de comportement.

Le vivant organisé est conçu à partir du vivant élémentaire comme un ensemble de couplages particuliers entre des systèmes autopoiétiques disjoints, appelés couplages par clôture opérationnelle [Varela, 1989b]. La notion de système autopoiétique de seconde espèce résulte du couplage de systèmes autopoiétiques de première espèce.

On peut considérer qu'à un niveau META, le comportement d'un SMA, qui génère mécaniquement les propres agents le constituant pourrait s'apparenter à de l'autopoièse.

I.6.3. Notion d'émergence

« Le tout est plus que la somme des parties » - C'est ainsi que l'on pourrait résumer simplement le concept d'émergence. Contrairement à l'approche classique de résolution de problème pour laquelle la tâche globale est décomposée en sous-tâches (principe Cartésien « décomposer le problème en autant de parties qu'il se peut pour les mieux résoudre »), l'approche émergentiste ne fait que coder les agents, l'environnement et les interactions. C'est lors de l'exécution que le processus construit un cheminement vers une éventuelle solution. Ce processus n'est en général pas optimum sur le facteur temps et sur le facteur quantité d'information échangée. Cette approche est néanmoins particulièrement intéressante. Le terme d'émergence était employé à l'origine par les philosophes, les biologistes et les physiciens pour caractériser le fait qu'une chose « sorte » d'une autre sans que celle-ci la produise à la manière dont une cause produit nécessairement un effet et suffise à en faire comprendre l'apparition. Cette propriété est reprise depuis quelques années dans le domaine de l'IAD et particulièrement dans la branche des SMA. Il est souvent postulé (à tort ou à raison) que l'activité collective massive et complexe observée au niveau du système global est la résultante des multiples interactions d'activités plus simples au niveau des agents constituant le système.

A noter que l'émergence est différente d'un épiphénomène puisqu'un épiphénomène est causé par le processus mais n'interagit pas avec le processus. L'ombre de la main est un épiphénomène puisque sa présence ne change pas ce que l'on fait par ailleurs. Pour être remarqué et identifié, un phénomène, un comportement ou un processus qui émerge doit être suffisamment stable.

Le terme d'émergence peut désigner le résultat même ou le processus qui a conduit à ce résultat. Il est difficile d'expliquer ce qu'est l'émergence, le concept permet de couvrir certains phénomènes souvent inexplicables ou relevant du domaine de la « magie »… Les

interactions des agents vont faire converger le système vers un état stable ou non, et si l'état est relativement stable, on considère avoir une solution au problème. Ce phénomène n'est pas nouveau et n'est en tous cas, pas différent des automates cellulaires qui se stabilisaient parfois en des formes étranges ou au contraire se mettaient à osciller entre deux états (voir la section sur le chaos). Se pose alors le problème de l'imprédictibilité des phénomènes émergents. On parle de non-réductibilité du Tout aux parties. Le concept d'émergence en philosophie s'oppose à celui de réduction qui consiste à considérer qu'un phénomène peut toujours être expliqué par des processus sous-jacents. Mais est-ce réellement rationnel ? Toutes ces tentatives d'explication, de formalisation, de théorisation ne prennent pas en compte le facteur temps. En outre la notion d'émergence est aussi caractérisée par les idées de continuité et d'irréversibilité. Il y a, dans un système dynamique, une itération entre, d'une part, l'interprétation que font les agents de leur environnement local, et d'autre part, les actions des agents sur l'environnement. Les actions modifiant l'environnement, il y a ensuite à nouveau interprétation, etc. On parle d'émergence lorsqu'une telle dynamique se stabilise. Cela peut être l'émergence d'une structure ou d'une fonctionnalité globale. Mais il s'agit toujours là d'un postulat des émergentistes. On retrouve cette notion de stabilisation dans les réseaux de neurones.

En tout état de cause il faut se placer à un niveau n+1 pour observer une émergence d'interactions à un niveau n. On ne parle donc pas d'émergence si on n'a pas d'observateur au niveau du phénomène global. Si on trouve une cause entre le niveau n et n+1 alors il n'y a pas émergence, l'un est réductible à l'autre. Cela constitue en soit un gros problème car il est bien connu que le fait même d'observer une expérience peut influencer l'expérience (?!). L'observateur peut rester extérieur au système, auquel cas on lui demande un effort d'interprétation dans une direction qui n'est pas intrinsèquement contenue dans la dynamique du système ou l'observateur peut lui-même être partie prenante de la dynamique d'ensemble de par ses possibilités d'interaction avec le système auquel cas il est moteur même de l'émergence [M.R. Jean, 1997] et [Capera & al., 2002].

Il existe plusieurs types d'émergences. On note l'émergence comportementale qui est typiquement fonctionnelle et caractérisée par la façon dont les agents interagissent dans le but de remplir une tâche. L'autre type d'émergence est l'émergence organisationnelle qui est axée sur l'apparition de caractères d'organisation.

Il est difficile d'intellectualiser cette organisation émergente. On ne sait sur quel critère, régularité ou disparité, juger pour avoir une idée de la "valeur" d'une organisation. Plusieurs niveaux d'organisations récurrents sont alors envisageables et la nécessité d'agents observateurs de la forme du SMA peut être utile et intéressante à implémenter.

I.6.4. Exemples d'émergence

L'émergence est un concept observé dans beaucoup de domaines. Nous proposons ici deux domaines caractéristiques.

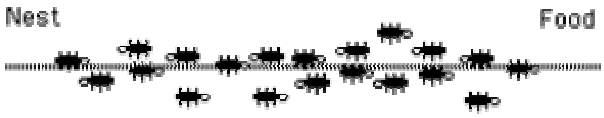
I.6.4.1. Emergence dans les phénomènes naturels

On observe un phénomène d'émergence dans beaucoup de processus naturels comme les phénomènes climatiques. L'auto-organisation est en effet présente dans le monde et en particulier celui des insectes sociaux. Par exemple, le regroupement géographique d'objets dans des colonies d'insectes. Des observations d'éthologistes ont rapporté que des insectes très simples parviennent à regrouper des objets comme des brindilles sur un ou plusieurs endroits d'un environnement. C'est un exemple de communication indirecte entre agents (voir

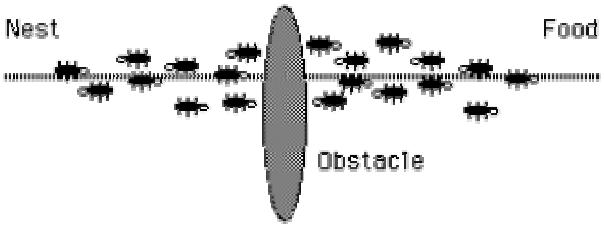
annexes). Des informaticiens ont représenté ces phénomènes dans le cadre d'une simulation : des agents sont présents dans un environnement dans lequel sont disposées aléatoirement des brindilles. Chaque agent a pour unique règle de fonctionnement « si je trouve une brindille, je la prends avec moi et la dépose si je rencontre une autre brindille ». Il n'y a aucun autre algorithme sophistiqué codé dans l'agent. Le résultat d'une telle simulation, après un certain nombre d'itérations, est que les brindilles sont regroupées en un ou plusieurs petits tas dans l'environnement [Bonabeau & al., 1999]. L'informaticien n'a pas codé de fonction particulière issue de ses expériences vaudoues pour faire « émerger » cette organisation. C'est de cette façon que les termites construisent les termitières.

Il est bien entendu possible de tirer d'autres exemples d'auto-organisation dans la nature comme l'émergence d'un chemin minimum par communication indirecte (voir annexes) entre deux points géographiques. C'est typiquement le cas des fourmis qui cherchent et rapportent de la nourriture vers leur nid. Elles sont capables de trouver le plus court chemin de l'un à l'autre [Deneubourg & al., 1990b] sans utiliser le sens de la vue. De même, elles sont capables de s'adapter au changement de l'environnement dans le cas par exemple où un chemin n'est plus accessible du à un obstacle nouveau.

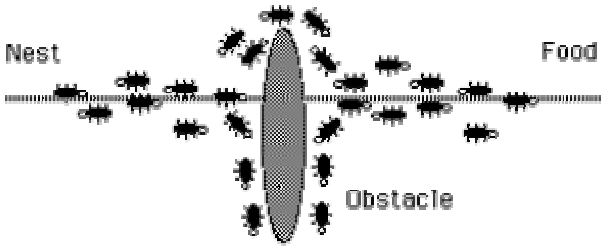
Sur la figure suivante, il n'y a pas d'obstacle entre la nourriture (Food) et le nid (Nest) : les fourmis se déplacent linéairement en suivant le chemin le plus court, la ligne droite.



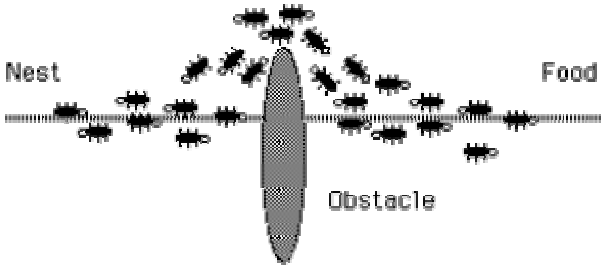
Les fourmis déposent une trace chimique (des phéromones) lorsqu'elles se déplacent et préfèrent suivre un chemin riche en phéromones plutôt qu'un autre dont la concentration est moins riche. Cette règle de fonctionnement simple suffit à expliquer comment elles peuvent trouver le chemin le plus court pour rejoindre leur nid, dans le cas par exemple où un obstacle les empêcherait de suivre une ligne directe.



Lorsqu'elles sont confrontées à un obstacle inattendu, elles ne peuvent utiliser leur chemin habituel et elles doivent choisir entre contourner l'obstacle par un moyen ou par un autre, dans l'exemple ci-dessous, par la droite ou par la gauche. On notera que pour l'expérience, le barycentre de l'obstacle n'est pas placé exactement sur la ligne médiane constituant le chemin initial des fourmis. On peut donc s'attendre initialement à ce que la moitié des fourmis passent par la gauche et l'autre moitié par la droite (figure suivante).



On observe que les fourmis qui par chance ont choisi le chemin le plus court, parviennent à reconstituer plus rapidement la concentration en phéromones sur ce nouveau chemin que celles qui ont choisi le chemin le plus long. Plus la concentration va augmenter, plus d'autres fourmis vont avoir tendance à choisir le chemin riche en phéromones. Du à ce processus auto-catalytique, les fourmis vont finalement choisir le chemin le plus court (figure suivante).



L'aspect le plus intéressant de cette expérience est incontestablement ce processus auto-catalytique qui se met en œuvre. Trouver le chemin le plus court est le résultat de l'émergence de l'interaction de différents agents autonomes. Les fourmis se déplacent à peu près à la même vitesse et déposent globalement une même quantité de phéromone sur leur passage. Il est clair que la trace de phéromone a tendance à s'accumuler plus rapidement sur le chemin le plus court que sur le chemin le plus long et comme les fourmis ont une tendance à se diriger naturellement vers le chemin contenant le plus de phéromone, il en résulte ce fameux processus auto-catalytique. Dans le cas d'un chemin strictement identique en terme de longueur, une simple différence de concentration à un instant donné peut décider du contournement par la droite ou par la gauche de l'obstacle. On s'approche ici du domaine du chaos, le battement des ailes d'un papillon en Amérique peut provoquer une tempête sur l'autre hémisphère. Il se peut que le processus mette un temps infini et que les fourmis aient tendance à utiliser aussi bien un chemin plutôt que l'autre. Peut être que d'autres paramètres interviennent aussi dans une moindre mesure comme la facilité physique à exécuter une action plutôt qu'une autre (il est plus facile de descendre un pente que de la monter). Une des règles de la nature étant de tendre vers la réalisation de l'action nécessitant la moindre dépense énergétique[12]. Cela fonctionne globalement par des systèmes d'attracteurs que nous étudierons un peu plus loin dans cette partie. Nous verrons aussi que dans certains cas, ces attracteurs peuvent avoir des effets pervers et empêcher l'agent de sortir d'un espace d'états.

I.6.4.2. Emergence dans l'économie

Je crois qu'il n'existe pas de plus bel exemple d'adaptation que l'économie de marché. « L'ordre économique est une émergence, c'est la conséquence non intentionnelle et non voulue des actions d'un grand nombre de personnes mues par leurs seuls intérêts [Friedman, 1992]. Le résultat global est l'auto-organisation du marché.

Lorsqu'une voie est porteuse, beaucoup d'entreprises se créent. S'il y a suffisamment de ressources c'est à dire de capital, elles y dégagent du bénéfice. Lorsque trop d'entreprises sont créées et se partagent les ressources, ces dernières arrivent à manquer et donc les entreprises s'affaiblissent. Certaines meurent. Celles qui survivent le peuvent grâce aux ressources qui sont suffisantes pour que l'entreprise se maintienne dans son domaine de viabilité (seuil critique de rentabilité). Il y a un équilibre dynamique qui se crée. Une boulangerie possède par exemple un certain nombre de clients et fait des bénéfices. Voyant cela, un autre agent boulanger décide d'ouvrir sa propre boulangerie dans la même rue à

[12] Nous verrons que notre modèle YAMAM reprend ce principe.

quelques mètres du concurrent déjà installé. Puis un autre fait de même. La clientèle observe donc l'ouverture des nouveaux magasins. Les clients se répartissent à présent dans les multiples boulangeries qui, par voie de conséquence, ne font plus chacune assez de bénéfice et mettent la clé sous la porte. Face à la demande des agents clients et donc face à une bonne perspective de chiffre d'affaire, une nouvelle boulangerie va ouvrir ses portes. Il y a eu une stabilisation du système mais celle-ci reste fragile puisqu'il n'est pas impossible qu'un nouvel agent boulanger vienne à nouveau perturber cet état d'équilibre… Il s'agit bien entendu d'un schéma simplifié car de nombreux autres facteurs peuvent complexifier le modèle. La réalité inclut notamment un facteur difficilement simulable qui est l'aspect psychologique des clients. C'est notamment vrai lorsque une information est ébruitée à la bourse. Un agent croit avoir une information et décide de vendre ses actions. Les autres, voyant avec quelle motivation il souhaite céder ses actions, font de même. C'est l'effet boule de neige, exemple de phénomène chaotique. Une petite déstabilisation du système engendre finalement un profond déséquilibre. Nous nous sommes intéressés de très près à cette notion d'émergence d'auto-organisation dans l'économie et cet exemple sera repris ultérieurement dans la conception de notre système multi-agents générique.

I.6.5. Critique de l'émergence

A toute théorie, même scientifique, s'opposent des irréductibles qui critiquent, et bien souvent dénigrent, la thèse. L'émergence est un phénomène qui peut difficilement être mis en équation. Il n'y a pas eu pour le moment suffisamment de recherches sur ce phénomène étrange. Certains critiquent son apport et d'autres réfutent son existence même. En effet, L'émergence ne se pilote pas et ne se contrôle pas. On peut parfois prédire une émergence, prédire vers quel espace d'état va tendre le système émergent mais jamais on ne pourra contrôler totalement l'émergence. C'est en effet la problématique de l'émergence. Le système n'est plus totalement déterministe et l'informatique passe des sciences dures aux sciences molles. Minsky, égal à lui même, pense que « les phénomènes émergents peuvent tout à fait être expliqués dès lors que l'on prend également en compte les interactions entre les parties ainsi que les particularités et lacunes des perceptions et des attentes de l'observateur » [Minsky, 1988]. Bertalanffy corrobore cette position « la connaissance de l'ensemble des parties contenues dans un système et celle des relations qui les lient permettra de déduire du comportement des parties celui du système » - [Bertalanffy 1973].

I.6.6. Intérêt de cette approche

L'intérêt principal est bien entendu des propriétés nouvelles non programmées a priori et permettant au système de résoudre globalement un problème. Un autre intérêt est l'adaptation des phénomènes et des structures émergentes aux aléas et changements de l'environnement (dans la mesure où ces changements ne sont pas trop radicaux). Dans le cadre des SMA, les agents sont indissociables de leur environnement. Les agents évoluent en même temps ou presque que leur environnement. Il ne peut y avoir émergence sans environnement. Il est raisonnable de penser que l'environnement sert uniquement à instancier des règles comportementales et structurales de base permettant une éventuelle émergence d'un phénomène global. La conscience serait un exemple typique d'émergence.

Comment une activité massivement parallèle peut-elle faire naître l'émergence de sens par auto-organisation ? - Dans bien des cas, une coopération multi-agents se réduit à un échange de données symboliques. Est-il intrinsèquement illusoire d'attendre d'un accroissement quantitatif des agents, un raffinement qualitatif ? Sans doute, la qualité n'est pas dans le nombre mais bien dans l'organisation, dans un équilibre dynamique entre

redondance et spécialisation. La redondance des agents consistant le système est nécessaire à sa robustesse et le caractère de spécialisation engendre son efficacité.

La difficulté de l'exercice tient du fait qu'on ne sait « faire émerger » mais on commence à savoir comment préparer les conditions nécessaires à l'émergence. Dans le contexte des insectes sociaux [Deneubourg & al., 1990a] et [Camazine & al., 2001], des simulations sont effectuées informatiquement en codant des règles simples de fonctionnement, grâce auxquelles le système manifeste un comportement émergeant. Mais la cause véritable de l'émergence se situe probablement dans la création de ces règles simples de fonctionnement et non pas après l'application de ces règles. Les travaux sur les Animats [Meyer & Wilson, 1991] se rapprochent davantage de ce champ d'investigation. De nombreux ouvrages sont consacrés aux études des insectes sociaux et des phénomènes d'émergence avec des approches mathématiques et algorithmiques. [Bonabeau & al., 1999] est l'un des plus complet, il propose aussi des applications de ces modèles d'insectes sociaux dans la conception de systèmes complexes plus généraux.

I.6.7. Une tentative de formalisation de l'émergence

Il faut tenter de comprendre les circonstances qui font qu'un phénomène d'émergence peut se produire et donc établir une corrélation entre l'état initial du système et son état final, si état final il y a, puisqu'il arrive fréquemment que le système oscille autour d'un attracteur. Vince Darley [Darley, 1994] essaie de définir plus rigoureusement le concept de phénomène émergent dans un système complexe. "Emergent phenomena are those for which the amount of computation necessary for prediction *from an optimal set of rules, classifications and analysis, even derived from an idealised perfect understanding, can never improve upon the amount of computation necessary to simulate the system directly* from our knowledge of the rules of its interactions" – Vincent Darley 1994. Il précise que la meilleure façon de prédire un phénomène émergent reste la construction d'une simulation.

Il y a différents niveaux de difficulté et de complexité selon le regard porté au système. C'est un niveau de granularité d'étude : globalement on observe que le système passe d'un état A à un état B, mais il est possible de décomposer davantage pour comprendre les raisons de la transition. Il est probable que la simulation soit le seul moyen aujourd'hui de prédire la tendance comportementale d'un système (le terme prédire est volontairement vague) - si toutefois il est possible d'entrer dans la simulation tous les facteurs susceptibles d'influencer l'évolution du système (ce qui est peu probable). Si on omet des paramètres mêmes jugés d'influence minimum, la simulation risque de ne pas rendre compte du comportement du système réel. La raison principale est la présence du chaos, c'est-à-dire le fait qu'une micro perturbation puisse engendrer finalement une tendance complètement différente du système. La météo est un bon exemple. Si on reprend l'exemple du papillon, on dit qu'un battement d'ailes à un endroit du monde peut déclencher une tornade à des milliers de kilomètres. La difficulté de prévision d'un phénomène dépend de sa taille et de la complexité des interactions d'un membre donné de la famille des systèmes complexes. Vince Darley conclut par "*Emergence is purely the result of a phase change in the amount of computation necessary for optimal prediction of certain phenomena*".

Pour formaliser (n représentant la taille du phénomène), il définit s(n) comme la quantité de calcul théoriquement nécessaire pour simuler un système réel et arriver à la prédiction du phénomène et u(n) comme la quantité de calcul nécessaire pour parvenir au résultat avec la méthode de « deeper level of understanding » qui consiste à réaliser une analyse plus sensible et de déduire le prochain état du système.

Si u(n) < s(n) alors le système est non-émergent

Si u(n) ≥ s(n) alors le système est émergent.

Pour la plupart des systèmes simples, u(n) << s(n), mais pour beaucoup de systèmes dits complexes, alors que la taille et la complexité augmentent, il va y a avoir une phase de changement où les courbes u(n) et s(n) vont se croiser. Cette intersection des courbes montre qu'il n'y a pas de discontinuité séparant les systèmes émergents des systèmes non-émergents. Toute la connaissance utile pour prévoir la tendance est contenue dans l'accumulation des interactions. La quantité de calcul c est le nombre d'étapes par la complexité de chaque étape. c est donc donné par la formule :

$c = \sum_{s \in S} (\text{complexity of } s)$ où S est l'ensemble des étapes nécessaires au calcul.

On suppose que toutes les étapes de calcul sont identiques. La complexité d'une étape est mesurée en terme de complexité de Kolmogorov [Li & Vitanyi, 1997]. Il propose une justification de cette définition sur un automate cellulaire, une cellule ayant une dépendance de ses cellules voisines pouvant recevoir plusieurs états possibles. Il postule enfin que cela est valable pour tout système complexe. Cela lui permet de proposer une définition formelle d'un ratio d'émergence et d'un coefficient d'émergence. Il spécifie que si la notion d'émergence existe bien dans les systèmes complexes, elle implique néanmoins une irréductibilité computationnelle.

Le fonctionnement d'un système multi-agents provoque souvent des phénomènes émergents. Ces phénomènes peuvent être souhaitables ou non dans la mesure où ils sont de nature non-prévisible. Seule une simulation peut permettre de connaître les tendances d'évolution d'un système complexe. *"I argue that in a certain fundamental sense, emergent systems are those in which even perfect knowledge and understanding may give us no predictive information" – Vince Darley.* Le caractère apparemment imprévisible des systèmes complexes est dû à un phénomène chaotique que nous étudierons dans les prochains paragraphes.

I.6.8. Dynamique et Chaos

Parmi les réactions possibles d'un système face à des perturbations extérieures et aléatoires, il semble que la dynamique du chaos (issue de la théorie des systèmes dynamiques et chaotiques du mathématicien Henri Poincare) soit une solution robuste et puissante. Le chaos [Gleick, 1999] ressemble à du hasard, aussi pourrait-on croire que de tels systèmes réagissent à du hasard par du hasard. En fait, les systèmes chaotiques sont totalement déterministes[13] mais ils sont d'une diversité telle que l'observation de leur comportement nous fait croire à un comportement aléatoire. La découverte des systèmes chaotiques réconcilie en effet les notions apparemment antinomiques de chaos et de déterminisme. Des systèmes " très simples " obéissent à des lois parfaitement déterministes et pourtant leur comportement est totalement imprévisible. Cette imprévisibilité n'est pas le fruit du hasard (ces systèmes n'obéissent pas à la loi des grands nombres comme le tirage du loto), mais de la sensibilité aux conditions initiales. Déterministe, parce que des effets objectifs et précisément mesurables et repérables déterminent la suite des événements. Chaos, parce qu'on ne sait pas du tout ce qui va se passer, malgré la connaissance que nous avons de toutes les données qui déterminent les événements.

De même que l'excès d'ordre engendre désordre et cacophonie, la théorie du chaos nous enseigne qu'il contient en lui-même ses propres facteurs d'équilibre et d'ordre. On peut

[13] Le Petit Larousse définit le déterminisme comme la conception philosophique selon laquelle il existe des rapports de cause à effet entre les phénomènes physiques, les actes humains etc...

utiliser le hasard pour auto-organiser un système. Ceci a donné de bons résultats pour la technique du recuit simulé (pour « palier » la complexité des problèmes NP complets). Utiliser le chaos pour auto-organiser un système pourrait peut-être conduire à des résultats meilleurs encore. En effet les méthodes probabilistes peuvent difficilement faire émerger de l'auto-organisation loin de l'équilibre, tel que le phénomène des tourbillons de Benard. Au contraire, des modèles chaotiques semblent pouvoir se stabiliser sur ces états organisés loin de l'équilibre.

En utilisant certaines ce ces théories (parfois plus philosophiques que mathématiques), on pourra se demander comment et pourquoi, à partir de diverses unités actives telles que l'individu dans une population, une intelligence globale de groupe se dégage liée à des formes d'organisation, alors que l'individu lui-même ne possède pas suffisamment d'intelligence pour comprendre le processus général de fonctionnement du groupe. C'est le cas de la fourmilière qui est un système intelligent dans son ensemble tout en réunissant des êtres dénués d'intelligence (voir les travaux d'A. Drogoul). Les organisations qui se forment dans une population d'individus sont parfois le résultat d'attracteurs dans un système chaotique.

Outre les mathématiques et la physique, la notion de chaos est aussi présente dans le domaine des sciences sociales et de l'économie [Jemam, 1993] et [Laszlo & Laugel, 1998]. A l'instar des phénomènes physiques complexes, les sociétés humaines peuvent se révéler, elles aussi, complexes et/ou chaotiques. Mark Michaels [Michaels, 1995] a déterminé sept lois fondamentales relatives aux comportements organisationnels et aux systèmes sociaux. Ces sept lois établissent que des systèmes simples peuvent devenir complexes, que ces systèmes sont dynamiques, sont bâtis sur un feed-back positif, sont sensibles aux conditions initiales et sont auto-organisationnels. Ces systèmes, toujours selon Michaels, peuvent être contrôlés par le chaos. Ilya Prigogine, Erwin Schrödinger, S. Kauffmann et H. Atlan ont démontré certains aspects des lois de Michaels et établi des conditions liées à l'établissement de l'auto-organisation dans les systèmes complexes telles l'existence d'un état loin de l'équilibre, la nécessité de non-linéarité, de redondance et de bruit, une certaine résistance au changement ainsi qu'une cohérence du système.

En ce qui concerne l'économie, [Baumol & Benhabib, 1989] se sont intéressés à la signification de la théorie du chaos et ses applications à l'économie. Ils mettent en évidence que l'étude de phénomènes économiques, d'apparence complexes et chaotiques, peut être facilitée par l'application de certains développements de la théorie du chaos. Ils ajoutent même que dans certains cas, l'utilisation d'outils tels que le diagramme de phase permet justement de démontrer que ces phénomènes chaotiques ne le sont pas en réalité et sont le fruit d'un système non-linéaire déterministe. Blake LeBaron [Lebaron, 1999], toujours en économie, s'est intéressé au problème de la prévision à partir de recherches effectuées sur la dynamique chaotique dans les séries temporelles économiques. Ces recherches ont évidemment un attrait dans le domaine des mouvements commerciaux, de la bourse et des marchés financiers ainsi que dans l'étude des cycles économiques. Les résultats observés sont très variables. Dans certains cas, l'analyse des séries temporelles par les outils dérivés de la théorie du chaos permet de mettre en évidence une non-linéarité et/ou la présence d'un attracteur. Cependant, dans le domaine de la prévision, diverses études menées sur des séries temporelles n'ont pas apporté de preuves réellement convaincantes quant à la suprématie des outils du chaos sur les outils statistiques traditionnels. Les applications de la théorie du chaos en économie ne s'arrêtent pas là. De nombreuses recherches ont aussi été effectuées dans l'étude des cycles économiques de Kondratief, de l'économie spatiale et de la rationalité économique…

[Nous insistons sur cet aspect non linéaire du fonctionnement de l'économie puisque nous allons utiliser dans notre modèle (chap 4) des concepts similaires à l'économie de

marché et une explication rapide sur le comportement parfois chaotique de l'économie peut s'avérer nécessaire].

Bien que le fonctionnement des marchés financiers soit d'une extrême complexité, il existe des mécanismes autorégulateurs basés sur un mélange subtil de psychologie humaine, de comportement social et de pensée rationnelle. Par exemple, si le prix d'un produit est trop élevé, la demande va baisser ce qui aura pour conséquence de faire baisser son prix. L'existence de ces mécanismes auto-régulateurs a des implications importantes et surprenantes sur le comportement des marchés, des prix et des économies, et peut entraîner le chaos. Ainsi, le comportement du prix de l'or est semblable à l'évolution des populations. Supposons que les marchands d'or, pour maximiser leurs profits, augmentent le prix de l'or chaque semaine tout comme une population à la Malthus croît chaque année. Quand les prix deviennent trop hauts, il y a moins d'acheteurs et le prix baisse, comme la croissance trop forte d'une population est limitée par le manque de nourriture, la maladie ou la guerre.

En fait, dans ce modèle simplifié, les équations qui décrivent l'évolution du prix de l'or sont très semblables à celles qui décrivent l'évolution d'une population donnée. Tout comme dans le cas d'une population, le prix de l'or peut atteindre un certain équilibre si son taux d'augmentation d'année en année n'est point trop élevé. Si le taux est un peu plus grand, le prix de l'or commence à osciller entre 2, 4, 8, 16, … valeurs possibles, un dédoublement successif qui rappelle celui des populations. Si le taux est encore plus haut, le prix de l'or devient chaotique. Nous sommes loin de pouvoir prédire les hauts et les bas du cours de l'or, mais les comportements décrits ici suggèrent que la théorie du chaos a tout pour nous aider à mieux comprendre le comportement des marchés et des économies.

Le chaos a ainsi investit la physiologie, la biologie, l'écologie et l'économie. Mais son étude quantitative dans ces sciences dites « molles » est loin d'atteindre au degré de sophistication qu'elle connaît dans des sciences « dures » comme l'astrophysique du système solaire, l'hydrodynamique ou même la météorologie. Cette disparité tient à deux raisons principales. D'abord les systèmes biologiques et économiques sont si complexes qu'il est difficile de trouver les équations qui décrivent avec précision leur évolution temporelle. Les approximations sont ici encore trop simplistes. Ensuite, il s'agit de ce qu'on appelle des « systèmes complexes adaptatifs », c'est-à-dire des systèmes qui apprennent, se souviennent et s'adaptent, changeant par-là même la nature du système initial. Ainsi, même si on arrivait à écrire les équations d'évolution temporelle des systèmes biologiques et économiques, ces équations devraient changer lentement dans le temps.

Malgré ces difficultés, la théorie du chaos nous apprend que certaines situations dynamiques en biologie, économie ou politique, au lieu de conduire à un équilibre, peuvent mener à une évolution temporelle chaotique, impossible à maîtriser. Les politiciens, législateurs et autres responsables officiels doivent donc considérer la possibilité que leurs décisions, loin de produire le meilleur équilibre voulu, puissent induire des oscillations violentes et imprévisibles, assorties de conséquences peut-être désastreuses. La théorie du chaos peut nous aider à avoir une meilleure compréhension de ces oscillations afin de mieux les maîtriser.

I.6.8.1. L'ordre à l'équilibre

Lorsque le milieu est à l'équilibre, au niveau macroscopique, bien que très désordonné au niveau microscopique, les effets locaux vont s'inhiber entre eux, on peut alors écrire des relations causales au niveau global. Ces relations semblent suivre une évolution relativement linéaire, stable et prévisible en termes probabilistes. On opère comme si le niveau local et microscopique n'existait pas malgré sa grande agitation. Il apparaît alors des relations

causales au niveau global. Ce genre d'ordre est observable dans certaines simulations multi-agents.

I.6.8.2. L'ordre par fluctuations

Lorsqu'un système est loin de l'équilibre, une faible fluctuation peut dans certaines conditions favorables, s'amplifier jusqu'à unifier et auto-organiser la totalité du système. Comme si chaque micro-élément constituant du système était informé de l'état de l'ensemble du système. Dans l'ordre par équilibre, les systèmes évoluent « linéairement » pour de petites variations des contraintes. Ici, au contraire, le système va même ignorer ces contraintes, il va s'auto-organiser, en réaction à ces contraintes, il atteindra alors une autonomie telle vis à vis de ces contraintes, qu'il en deviendra quasi-indépendant. Il n'y aura plus de relation causale liée aux conditions aux limites. Le pionnier dans l'ordre par fluctuations fut certainement Prigogine, prix Nobel de chimie 1977 pour son travail sur la thermodynamique du non-équilibre. La thèse de Prigogine tend à montrer qu'il existe à coté des sciences déterministes, une autre science non linéaire et quasi-imprévisible ; la sciences des phénomènes chaotiques. Elle semble gouverner des disciplines aussi diverses que certains mécanismes neuro-biologiques ou l'évolution du climat. La théorie des structures dissipatives de Prigogine est à rapprocher de la théorie des catastrophes et du chaos fractal.

Une des illustrations de l'ordre par fluctuations de Prigogine est le processus d'élaboration des termitières (étudié par le professeur GRASSE), formes très désorganisées ressemblant à des arcades : une sorte de jonction entre 2 piliers [Bonabeau & al., 1999].

Observation de la construction d'un pilier :

Chaque termite semble avoir un comportement désordonné, il transporte et abandonne aléatoirement (?) de petites boulettes de terre. Ces boulettes sont imprégnées d'une substance hormonale ayant la propriété d'attirer les termites. Il y aura donc une amplification du phénomène à un endroit déterminé. (N.B : On peut imaginer que cet endroit est déterminé par le principe de la distribution déséquilibrée des cumuls de phénomènes aléatoires. En terme de probabilité, un endroit deviendra peut être une zone d'attraction pour un pilier de la termitière).

Jonction des 2 piliers :

Le fait que les deux piliers voisins se rejoignent pour former une sorte d'arcade, peut s'expliquer par une attraction d'ordre secondaire par la substance du pilier voisin. Il y aurait attraction globale des deux piliers par déplacement progressif des centres de gravité de chacun des piliers. On retrouve une explication détaillée de ce fonctionnement dans [Bonabeau & al., 1999].

I.6.9. Notion d'attracteur et de système chaotique

Le chaos est un comportement apparemment imprédictible dans un système déterministe en raison d'une grande sensibilité à des conditions initiales. Le chaos apparaît dans un système dynamique si deux points de départs proches divergent exponentiellement. Allgood et Yorke définissent le chaos comme une trajectoire qui est exponentiellement instable et qui est ni périodique ou périodique asymptotiquement, c'est-à-dire qui va osciller irrégulièrement sans se fixer.

Il semble que les systèmes qui présentent un nombre de paramètres gigantesque appartiennent à la catégorie des systèmes non prévisibles. Ce n'est cependant pas toujours vrai. Un système ne possédant qu'une variable peut se révéler imprévisible ; la raison réside dans la notion mathématique du chaos qui peut s'infiltrer dans le plus simple des systèmes

dynamiques. Les systèmes déterministes peuvent donc se comporter de façon apparemment imprévisible ou aléatoire. Cela est souvent dû à la présence ou l'absence d'attracteurs. Un attracteur simple est un système attiré vers un régime stable, et qui est totalement déterminé par ses conditions initiales et insensibles aux perturbations. Ce type d'attracteur n'a qu'un petit nombre de points fixes (un ou deux) et est insensible au bruit. Un attracteur fractal, quant à lui, fait qu'un système va errer parmi des milliards de points fixes différents, ce système est sensible aux conditions initiales mais aussi hyper-sensible aux perturbations. On est dans le cadre du chaos fractal. Le chaos permet de comprendre en quoi les SMA sont concernés par ces phénomènes et plus particulièrement encore dans le cadre des simulations par SMA, très sensibles aux conditions de départ.

I.7. Conclusion

Nous avons présenté dans cette partie les grandes tendances de l'IA et de l'IAD. Nous avons exposé également la problématique des systèmes complexes qui font nécessairement référence à des notions de structure et d'organisation avec des effets de bords liés aux concepts de dynamique, d'émergence d'auto-organisation, de chaos etc. Cela s'inscrit dans un cadre global concernant la systémique. Cette démarche permet effectivement une approche transversale et pluridisciplinaire des phénomènes, en définissant un certain nombre d'invariants de structure, autant de points d'équilibre.

Les principes de la « systémique » sont présentés dans [Rosnay, 2000]. L'auteur propose une décomposition des phénomènes en processus et une identification de leurs interrelations[14]. Il intègre à cette démarche générale des outils comme la théorie du Chaos, qui est une théorie de l'équilibre comme son nom ne l'indique pas, la géométrie fractale, l'approche énergétique du chimiste afin de réaliser un bilan des perspectives technologiques, politiques et géopolitiques, économiques et sociales de la planète.

Nous avons posé quelques définitions sur lesquelles nous allons nous appuyer dans la prochaine partie pour parler plus en détail d'agents et de systèmes multi-agents qui sont des outils bien adaptés pour répondre à la simulation de ces systèmes dynamiques complexes. L'objectif de cette partie était de poser et de cadrer notre domaine d'étude. L'intelligence artificielle est un vaste sujet de recherche et la question même d'une intelligence conçue par l'homme continue d'alimenter les débats philosophiques et scientifiques. Nous postulons que l'intelligence peut émerger de l'interaction entre de nombreuses entités et que le chaos constitue une première voie de recherche pour tenter de maîtriser le comportement émergeant des systèmes ayant une dynamique forte.

[14] De manière plus opérationnelle, il présente la métaphore du Cibionte, espèce de superorganisme dont le cerveau est constitué de couples d'humains et d'ordinateurs, reliés par les réseaux de communication, et dont le corps est composé de la planète terre et des superstructures humaines. Au sein de cette vision, l'humain serait un élément neuronal qui, en symbiose avec l'ordinateur, formerait l'élément de base du Cibionte. Cette fiction très intéressante reprend chaque concept scientifique exposé dans cette partie de manière plus romancée.

Chapitre II. Agents et SMA

« Les batailles qui mettent en concurrence des paradigmes ne sont pas de celles qu'on peut arbitrer à l'aide de preuves » – T.S. Kuhn

Résumé : Ce chapitre présente plus en détail les notions d'agents et de systèmes multi-agents. Ces systèmes possèdent des propriétés remarquables qu'il est utile de présenter dès à présent afin de mieux appréhender les prochains chapitres. Nous présenterons notamment leur typologie, leur comportement et leur caractère adaptatif lors de situations d'interaction et de coopération.

II.1. Introduction

Dans la première section de ce chapitre nous avons choisi de développer le concept d'agent en proposant différentes définitions. La deuxième section a pour objectif de donner une vision générale du paradigme multi-agents et d'introduire quelques concepts de base communément employés dans les systèmes multi-agents. Nous validons ensuite notre approche en montrant ce que l'IAD et notamment les SMA peuvent apporter au vaste problème de l'intelligence artificielle et peut-être à l'émergence d'une première forme d'intelligence.

Les systèmes multi-agents et l'IAD, en général, permettent d'appréhender des problèmes de taille et surtout de complexité plus importante que ceux abordés par les approches classiques de l'Intelligence Artificielle. L'IAD s'appuie sur la distribution de l'intelligence dans des agents formant une société dans laquelle chacun a une certaine autonomie. Les systèmes d'IAD ne sont plus fermés sur leurs propres connaissances, mais sont de véritables sociétés d'individus qui doivent se mouvoir, planifier, agir et réagir dans un milieu dans lequel ils entrent parfois en conflit avec d'autres agents. Un agent "vit" au sein d'une communauté d'agents appelée système ou univers multi-agents. Les agents peuvent être nombreux ou non. Ils peuvent avoir des objectifs individuels ou un objectif global à atteindre. Pour réaliser leurs objectifs, les agents peuvent coopérer, entrer en conflit, négocier, etc. Un système multi-agents est alors un espace virtuel ou non, composé d'un environnement, d'objets passifs et d'agents communiquant directement ou indirectement et pouvant opérer sur les objets de l'environnement.

De ces organisations[15] se dégagent certains caractères évoquant une certaine forme d'intelligence. Le domaine de recherche des SMA a émergé il y a une vingtaine d'années et reste en constante évolution. En effet, les premières recherches en Intelligence Artificielle, présentées brièvement dans le premier chapitre, s'étaient engagées dans des voies très réductionnistes en occultant notamment la notion de contexte de cas. De nombreux chercheurs en I.A. et de nombreux psychologues s'accordent à dire aujourd'hui que l'intelligence n'est

[15] Dans le cadre des SMA, l'organisation est l'ensemble des interrelations entre les composants actifs ou passifs d'un univers ou « l'ensemble des relations qui définissent une machine comme une unité » [Varela, 1989].

pas une caractéristique isolée indissociable du contexte social dans lequel elle s'exprime et dans lequel elle est interprétée [Cardon, 1996b].

Dans les programmes informatiques, la tendance est, de nos jours, à distribuer la connaissance et le savoir-faire. On parle d'informatique distribuée. De même, « l'intelligence » doit être distribuée pour profiter d'une « émergence » de sens rendue possible par les différentes interactions des nombreuses entités composant un système. Les Systèmes Multi-Agents se positionnent au carrefour de l'Intelligence Artificielle Distribuée et de la Vie Artificielle. Une analyse systémique contribue à mettre en exergue la problématique de l'interaction d'entités fonctionnelles et autonomes à laquelle peuvent répondrent les SMA.

Dans les systèmes d'IAD, un agent peut avoir plusieurs niveaux d'intelligence. Une première approche consiste à considérer des agents ayant un gros niveau de connaissance. Ces agents cognitifs sont apparentés à des experts ayant déjà des possibilités, à eux seuls, de réaliser des opérations complexes et pouvant être amenés à coopérer. On dit aussi que ces agents sont à gros grain. Une deuxième approche consiste à considérer des agents moins intelligents mais plus nombreux, de faible granularité, qu'on appellera agents réactifs. Dans ce type de système, l'intelligence du système émerge de l'interaction entre les différents agents.

Nous nous positionnons dans une perspective plus néosystémique qu'holistique[16] puisqu'on s'intéressera au fruit des interactions au sein d'un système. Un agent est une entité active et même proactive, relativement autonome, et la plus indépendante possible du reste du système. D'un point de vue global, le système présente alors une notion de cohérence et de stabilité grâce aux différentes interactions entre les entités le constituant.

Le terme d'agent au sens général est utilisé pour désigner une personne physique, une substance ou un organisme vivant [Mingers, 1994]. Il s'agit d'une entité agissant dans un environnement où existent d'autres entités similaires mais pas nécessairement identiques. [Shoham 1993]. Le terme d'agent fut repris dans le sens informatique pour désigner une entité artificielle pouvant représenter un organisme vivant [Drogoul, 1993]. Des organismes vivant de plus en plus complexes comme des insectes ou des animaux plus évolués encore ont servi de modèles aux informaticiens.

Le concept de base du paradigme multi-agents est bien sûr l'agent. Les puristes considèrent qu'un agent n'existe pas en dehors d'un système multi-agents. Il correspond à une entité active évoluant dans un entour, le tout composant le Système Multi-Agents (SMA). Nous traiterons l'étude des systèmes multi-agents et de leurs particularités dans la deuxième partie de ce chapitre. L'agent intervient donc dans la résolution de problèmes grâce à ses connaissances, son comportement[17] et son savoir-faire représenté par exemple sous forme de service. L'agent participe à la résolution collective d'un problème par le biais d'interactions avec les autres agents de ce système sans nécessairement posséder de vision globale de ces propres actions.

Le terme d'entité active est souvent utilisé pour désigner un agent car il permet de rendre compte de la notion d'objet concurrent et autonome ayant le propre contrôle de son activité. Il est employé pour signaler le fait qu'un agent ne corresponde pas à la notion de module statique [Hewitt, 1977] mais bien à une notion dynamique, c'est à dire que cette entité possède une identité et une certaine autonomie dans son environnement.

[16] Holisme : selon le dictionnaire Hachette, il s'agit d'une théorie selon laquelle les phénomènes sont des totalités irréductibles à la somme ou même à l'association structurelle de leurs composantes.
[17] Le comportement caractérise l'ensemble des propriétés émergentes que l'agent manifeste par le truchement de ses actions dans son environnement.

Il faut bien distinguer le concept d'agent du concept d'objet ou d'acteur. Les objets n'ont ni but ni recherche de satisfaction. Un objet est défini par un ensemble de services qu'il offre (ses méthodes) et qu'il ne peut refuser spontanément d'exécuter si un autre objet lui demande [Ferber, 1995]. Les agents disposent d'objectifs qui leur donnent une certaine autonomie décisionnelle vis-à-vis des messages qu'ils reçoivent. C'est pour cette raison qu'on utilise le terme enrichi d'entité active et pas seulement d'entité qui pourrait désigner effectivement un simple objet ou acteur. Les objets n'utilisent qu'un mécanisme d'envoi de message simple qui se résume à un appel de procédure qui peut être distant ou non. Il n'y a pas de langage de communication à proprement parler entre les objets. Pour les agents, les interactions sont nettement plus complexes et font intervenir des communications de plus haut niveau.

La différence entre un acteur et un agent provient de la notion de réflexivité organisationnelle que n'a pas a priori un acteur contrairement à un agent. Un agent doit disposer d'un comportement autonome, d'un but[18] et d'une représentation, même partielle, de son environnement. Un acteur est une entité qui exécute une tâche comme un calcul. Un agent possède une plus grande latitude comportementale, une motivation et une autonomie. A.Cardon avance la distinction suivante : « l'agent est pro-actif et pas l'acteur. La notion de pro-activité est génératrice de celle d'autonomie, qui est plus cognitive et encore plus générale. ». La proactivité signifie que l'agent ne reste pas inerte, sans activité mais cherche à accomplir un objectif. Il doit générer et réaliser ses propres buts. C'est en quelque sorte une prise d'initiative.

Un agent ne doit pas être contrôlé par une règle de production développée par le concepteur du système ; au contraire il est mû par un ensemble de tendances qui peuvent de surcroît évoluer dans le temps. Une entité intelligente (agent plus cognitif) doit être capable de se voir elle-même dans un système et surtout de percevoir l'impact de ses actions sur le système.

Il peut même y avoir une notion de récursivité dans le concept d'agent. On peut considérer qu'un agent soit l'entité atomique composant un système. Le système n'a d'existence propre que par les interactions des agents le composant. Ce système peut être vu à un plus haut niveau de granularité comme un agent - ce principe sera repris dans la conception de notre modèle d'organisation au chapitre IV. Il est par exemple possible de modéliser le fonctionnement d'une cellule humaine par des agents (modélisation biologique) [Tisseau, 1998] et pourtant à une autre niveau on va considérer que l'être humain entier est un agent (modélisation sociologique) ; le système multi-agents est alors composé d'agents cellules.

Dans ce chapitre, nous présentons les caractéristiques générales du paradigme agent en définissant la terminologie et une typologie possible des agents, avant de nous intéresser plus précisément au comportement et au caractère autonome attribué aux agents. Nous expliquons précisément ce qu'est un agent par rapport aux autres agents de son univers, c'est à dire quel rôle il joue et quelles organisations il peut former ou intégrer. Nous verrons que plusieurs formes d'agents sont catégorisées dans le grand « cosme » du paradigme multi-agents. Nous terminons en présentant les caractéristiques essentielles permettant de définir le comportement et « l'état mental » d'un agent.

[18] Dans le sens qui nous concerne, le Petit Larousse défini le but comme la fin que l'on se propose d'atteindre. Le but semble synonyme de objectif.

II.2. Les agents : définitions

Il y a bien sûr plusieurs définitions du terme agent. Certaines proposent de ne pas dissocier l'agent du Système Multi-Agents et de l'environnement. D'autres réduisent l'agent à un simple processus communicant. Des définitions générales communément admises sont ici proposées. Nous nous appuierons à l'avenir sur les définitions utilisées dans la communauté des chercheurs en SMA qui considèrent toutes, d'une commune mesure, que des agents sont des entités actives plus ou moins autonomes et composant un système multi-agents (SMA).

II.2.1. Définition de Erceau

"Les agents sont des entités physiques (capteurs, processeurs,...) ou abstraites (tâches à réaliser, déplacements,...), qui sont capables d'agir sur leur environnement et sur elles-mêmes, c'est-à-dire de modifier leur propre comportement. Elles disposent, pour ce faire, d'une représentation partielle de cet environnement et de moyens de perception et de communication." [Erceau, 1993]

II.2.2. Définition de Ferber

Jacques Ferber, propose cette définition [Ferber, 1995] et dans [Ferber, 1999]. *« On appelle agent une entité physique ou virtuelle :*

- *qui est capable d'agir dans un environnement,*
- *qui peut communiquer directement avec d'autres agents,*
- *qui est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser),*
- *qui possède des ressources propres,*
- *qui est capable de percevoir (mais de manière limitée) son environnement,*
- *qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),*
- *qui possède des compétences et offre des services,*
- *qui peut éventuellement se reproduire, mourir et changer d'état*
- dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit ».

II.2.3. Définition de Maes

Pattie Maes [Maes, 1995] du groupe de recherche SMA au MIT propose une bonne définition générique du terme agent: "*An agent is a computational system that inhabits a complex, dynamic environment. The agent can sense, and act on, its environment, and has a set of goals or motivations that it tries to achieve through these actions*."

Il sort de ses définitions trois notions fondamentales permettant de caractériser un agent : percevoir, décider et agir :

Percevoir : Le fait de percevoir implique le fait que l'agent soit plongé dans un univers et qu'il communique dans cet univers. Il peut ou non percevoir les actions des autres agents, percevoir des mouvements d'objets ou tout type de transformation de son environnement. L'agent possède donc un domaine de perception qui s'étend entre 0 et l'ensemble des événements ayant eu un effet sur l'environnement dans un temps donné.

Décider : cela induit le fait qu'un agent ait des connaissances qui lui sont propres et un comportement autonome lui permettant d'exploiter ces connaissances et de « raisonner »… Il décide en fonction de ses perceptions.

Agir : l'agent peut agir sur son environnement. Il peut manipuler des objets, les déformer, communiquer avec d'autres agents, engager une coopération ou un conflit. Toute action en tant que telle modifie d'une certaine façon une partie de l'environnement.

II.3. D'autres types d'agents

Les chercheurs en SMA proposent chacun leur définition de l'agent. D'une manière générale, chaque agent doit être plongé dans un environnement et doit communiquer avec d'autres agents de cet environnement. On trouve néanmoins dans l'industrie logicielle, d'autres agents qui ne répondent pas aux définitions des chercheurs du domaine. Ces agents peuvent être réduits à de simples programmes (processus) qui accomplissent des tâches. Ils adoptent tous une forme de mobilité. Il est possible de distinguer plusieurs types d'agents sur Internet : les agents pour le commerce et les achats, les agents assistants, les agents pour la recherche d'information et la veille.

Les agents pour le commerce se classent en deux catégories. Les premiers sont contrôlés par les clients. Souvent appelés shopping agents, ils ont pour but de faciliter le processus d'achat. Bargain Finder, conçu par Andersen Consulting, a été un des prototypes les plus connus sur Internet. Peu à peu, des agents plus sophistiqués ont vu le jour. Les seconds sont les agents contrôlés par les producteurs, que l'on appelle parfois « interfaces intelligentes ». Bien qu'ils suivent la même logique, ils sont dotés d'un système de profiling pour s'adapter à leurs utilisateurs. BroadVision, SelectCast et, Agentware constituent des « interfaces intelligentes » assez évoluées.

Les Agents assistants permettent de vous assister pendant vos activités de recherche sur le réseau internet par exemple. Les "agents guide" essayent de suggérer des sites susceptibles de vous intéresser, en observant votre comportement sur Internet. Les "agents de bureau" (desktop agents), vous assistent au sein du bureau de travail de votre ordinateur. Les tâches sont très diversifiées et sont parfois directement intégrées au sein d'un logiciel existant comme le Pack Office de Microsoft.

Les agents de recherche servent à optimiser la recherche d'informations sur Internet. Les meta-moteurs "on-line" interrogent simultanément plusieurs outils de recherche et vous évitent ainsi de le faire vous-même. Les meta-moteurs "off-line" sont le prolongement logique des méta-moteurs "on-line". Vous définissez préalablement avant la connexion votre stratégie de recherche avant la connexion. On peut citer Copernic, Inforia Quest, Matahari, WebFerret… Enfin les agents de recherche avancés présentent des fonctionnalités parfois très intéressantes comme la personnalisation par secteur d'activité, la veille, filtrage et archivage des résultats, l'analyse linguistique des requêtes ou encore l'analyse des résultats.

Il existe aussi des agents mobiles sur Internet qui sont capables de naviguer sur différents sites équipés d'une plate-forme commune permettant l'hébergement de ces agents. Telescript de General Magic est connu pour être la première implémentation commerciale des agents mobiles. Il s'agit d'un langage interprété que GM a remplacé ensuite par Odyssey basé sur JAVA. On y trouve la notion de « places » (lieux virtuels), de « travel » (pour se déplacer), de « meetings » (quand deux agents se rencontrent sur une place), de « connections » (lorsque les agents communiquent), « d'authorities » (pour définir le niveau d'autorisation et de « permits » (qui permet d'accéder à des ressources et d'exécuter des

instructions). Un autre projet célèbre d'agents mobiles fut les « aglets » d'IBM Japon, contraction de Agent et Applet. Le projet ne fut malheureusement pas véritablement suivi par IBM. Nous reviendrons sur les aglets dans le chapitre sur l'état de l'art des outils SMA.

II.4. Cas particulier des animats

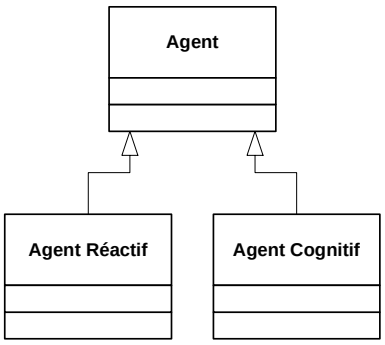
Les animats sont des animaux simulés sur ordinateur, ou des robots réels, dont les lois de fonctionnement sont inspirées de celles des animaux, et qui s'avèrent capables de « survivre » ou d'assurer leur mission dans un environnement éventuellement imprévisible ou menaçant [Meyer & al., 1993], [Cliff & al., 1994], [Maes & al., 1996]. Pour ce faire, les animats doivent maintenir dans leur zone de viabilité les variables essentielles à cette survie ou à cette mission [Ashby, 1952]. Ils sont donc être équipés de senseurs et d'effecteurs adaptés aux problèmes que leur pose l'environnement, ainsi que d'une structure de contrôle qui leur permette de choisir quelle action accomplir à quel moment en fonction des buts immédiats et lointains qu'ils essaient d'atteindre.

Un animat est "situé" dans un environnement réel ou simulé. Il a des senseurs qui captent des informations de l'environnement externe et/ou interne, des effecteurs qui lui permettent d'agir dans l'environnement, et une structure de contrôle qui coordonne ses perceptions et ses actions. Le comportement d'un animat peut être qualifié d'adaptatif tant que sa structure de contrôle permet de maintenir ses variables essentielles dans leur zone de viabilité. Dans la mesure où cette structure de contrôle sert aussi à choisir les buts successifs que l'animat cherche à atteindre ou à arbitrer entre des buts conflictuels, elle joue le rôle d'un système motivationnel. L'organisation de la structure de contrôle peut être modifiée par des processus de développement, d'apprentissage ou d'évolution.

L'approche animat a pour objectif à court terme de contribuer à l'avancement des diverses disciplines qui, au sein de la biologie, sont concernées par l'étude des comportements adaptatifs des animaux. Elle est également intéressée par toute possibilité de mise à l'épreuve du savoir correspondant, sous la forme de logiciels ou de matériels adaptatifs applicables à l'informatique ou à la robotique. A plus long terme, l'objectif de l'approche animat est de contribuer à l'avancement des sciences cognitives en recherchant en quoi l'intelligence de l'homme s'explique à partir des simples comportements adaptatifs hérités de l'animal.

Si une telle vocation paraît être la même que celle de l'*Intelligence Artificielle* (IA) traditionnelle, du moins lorsque cette dernière vise à comprendre les ressorts de l'intelligence humaine [Barr & Feigenbaum, 1981] l'approche animat vise à modéliser des organismes simples, mais capables d'interagir avec un environnement aussi proche du réel que possible, et dans lequel ces organismes doivent poursuivre plusieurs buts parfois contradictoires, comme se nourrir, se reproduire, échapper à leurs prédateurs, etc. Pour survivre, ces organismes ne peuvent utiliser que les informations que leur fournissent leurs senseurs, informations qu'ils doivent parfois activement rechercher et qu'ils doivent toujours interpréter en fonction de l'environnement rencontré et du but poursuivi. Ainsi, leurs structures de contrôle mettent à l'épreuve les conceptions de James [James, 1890], selon lesquelles "le cerveau n'est pas construit pour penser de façon abstraite, il est construit pour assurer notre survie dans le monde".

II.5. Typologie des agents



Les agents peuvent être classés en deux catégories principales selon leur comportement et leur granularité. Cette notion de granularité est bien sûr très subjective, elle exprime la complexité de « raisonnement » d'un agent afin de séparer les agents dits "intelligents" des agents moins "intelligents". On parle d'agents cognitifs et d'agents réactifs. Il s'agit là de deux écoles de pensée et d'un réel clivage. La première, celle des agents cognitifs, considère qu'un agrégat de ce type d'agents peut être quasiment assimilé à des systèmes experts distribués. La connaissance des systèmes experts en question étant utilisée pour la représentation du monde que se fait l'agent modélisé, ainsi que pour exprimer ses intentions[19] d'actions. La seconde école, celle des agents réactifs, ne suit pas le paradigme cognitiviste et considère qu'un agent de ce type, ne possède pas de vision globale du système mais répond par des actions à des stimuli. C'est typiquement le cas de l'agent « fourmi » dans son environnement « fourmilière ». La communication entre agents réactifs s'effectue à l'aide de propagations de signaux ayant une signification intrinsèque mais dénuée de sémantique profonde.

Il faut cependant convenablement pondérer cette dichotomie entre agents. Un agent cognitif peut être considéré comme un système complet, lui-même régit par des interactions d'agents réactifs. Le fonctionnement d'un agent cognitif résulte lui-même d'interaction entre entités de base qui peuvent être assimilées à des agents réactifs. Au niveau atomique, le fonctionnement du cerveau humain n'est que la résultante d'équations chimiques. L'être humain est réduit à un tas de cellules auto-organisées subissant l'influence de certains attracteurs chaotiques. Sa connaissance est réduite à des influx de potentiels électrochimiques. En descendant à un niveau de granularité aussi basique, tout devient représentable par des agents réactifs. La transmission des influx nerveux le long d'un neurone se fait par une conduction saltatoire transitant par des nœuds de Ranvier. Les neurones communiquent avec d'autres grâce à un signal, qui ayant passé ou non la fonction de seuil, se propage aux cellules nerveuses voisines. Un agent cognitif, considéré ainsi, n'est qu'une encapsulation d'une multitude d'agents réactifs.

L'émergence d'un comportement jugé intelligent, c'est à dire ayant un sens pour nous, est le fruit d'interactions multiples dans des sous-ensembles conceptuels d'entités basiques et réactives. Ces unités de base ne sont que très peu autonomes les unes envers les autres mais, dû à leur grand nombre et leur grande diversité et dû à des influences multiples de leur environnement, peuvent se révéler relativement autonomes lorsque considérées par groupe. C'est à dire qu'une zone d'activité qu'on pourra nommer cluster peut tendre vers un espace d'états particulier sans agir sur d'autres clusters. Cependant la frontière séparant les clusters n'est pas distincte. Elle évolue autant qu'évoluent les interactions locales des agents réactifs. Les fonctions de seuil définissent dynamiquement en fonction du nombre d'éléments actifs impliqués, l'étendue du cluster. C'est sur l'évolution massivement parallèle des agents réactifs que repose l'équilibre dynamique du système.

[19] Le Petit Larousse définit l'intention comme le dessein délibéré d'accomplir tel ou tel acte, c'est-à-dire la volonté. Dans le monde SMA, l'intention est le processus qui consiste à sélectionner un ensemble de buts à atteindre.

Les définitions classiques de ces types d'agents sont nombreuses, en voici une synthèse :

II.5.1. Agents réactifs

Ce sont des agents qui réagissent uniquement à leur perception de l'environnement et qui agissent en fonction de cette perception. C'est le niveau de complexité le plus bas des agents. Ce type d'agents ne peut pas vraiment être qualifié d'intelligent, étant donné que leur fonctionnement est principalement basé sur le principe du stimulus/action. Ce principe permet aux agents d'agir grâce à des réflexes totalement conditionnés. A la perception d'un stimulus particulier, l'agent fournit une réponse stéréotypée. La communication qu'il a avec d'autres agents et avec son environnement est très rudimentaire. Néanmoins, ils présentent des caractéristiques intéressantes [Drogoul, 1993] : simplicité de la description du comportement local et émergence de comportements globaux au-delà de la simplicité de l'agent. Les principaux travaux utilisant des agents réactifs concernent les robots [Brooks, 1989], [Steels, 1994], la planification [Gallone & al., 1994] ou la résolution collective de problèmes [Deneubourg & al., 1990b]. On utilise parfois le terme d'animats.

Un agent est réactif s'il n'est pas cognitif. Il ne possède pas de représentation explicite du monde, ou alors on peut dire que sa représentation du monde se situe à un niveau sub-symbolique, c'est-à-dire intégré dans ses capacités sensori-motrices. En particulier, un agent réactif peut être pulsionnel ou tropique.

Selon Ferber, un agent réactif est tropique s'il n'est pas pulsionnel, c'est-à-dire qu'il ne répond qu'à des stimuli de l'environnement, son comportement étant guidé intégralement par l'état local du monde dans lequel il se trouve plongé. Il agit de manière totalement réflexe aux états du monde, il n'a aucun but ni aucun état interne. Un agent réactif est pulsionnel si son comportement est téléonomique [Ferber, 1995], c'est-à-dire dirigé par des buts.

II.5.2. Agents cognitifs

C'est un type d'agents plus complexe [Erceau, 1991]. De tels agents sont non seulement capables de percevoir et d'agir sur leur environnement, mais en plus ils ont des capacités de cognition leur permettant de raisonner sur les autres ou sur l'avancement de la résolution. Ils font souvent appel à des modes de communication plus complexes qu'une simple perception. Ils communiquent généralement grâce à des structures de données partagées ou par des communications directes. L'action d'un tel agent n'est plus seulement une réaction à un stimulus externe mais une réelle production de son comportement interne. La différence entre l'agent cognitif et réactif se situe d'une part dans la complexité des représentations au sein des agents, mais aussi dans la complexité des informations échangées.

Pouvant utiliser leurs capacités cognitives, les agents cognitifs échangent des données plus riches au niveau sémantique que les agents réactifs. En effet, ils sont souvent pourvus de connaissances et de savoir-faire leur permettant des raisonnements plus élaborés (inférences, filtrage des informations, accointances[20], représentation neuronale...). La gestion de leurs interactions s'en trouve donc différente, [Chevrier, 1993]. Bien souvent, les agents cognitifs disposent d'une représentation explicite des autres agents et de la manière d'interagir avec eux. Cette représentation est désignée par le terme de réseau d'accointance.

Un agent est cognitif s'il est "intelligent" : il dispose d'une base de connaissance comprenant l'ensemble des informations et des savoir-faire nécessaires à la réalisation de sa

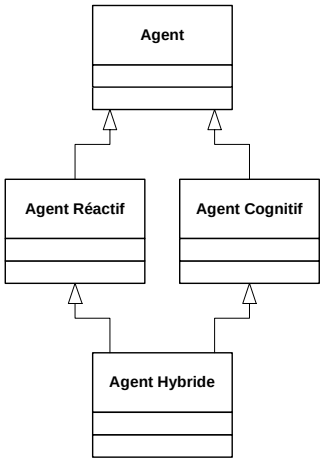
[20] Dans certains cas où les agents sont essentiellement communicants, chaque agent est relié directement à un sous-ensemble d'agents appelé son réseau d'accointances. Il s'agit en quelque sorte de son carnet d'adresses.

tâche et à la gestion des interactions avec les autres agents et avec son environnement. Il dispose d'une représentation symbolique et explicite du monde à partir de laquelle il peut raisonner. Un agent cognitif peut être intentionnel si son comportement est téléonomique[21]. Dans ce cas, l'agent possède des plans explicites lui permettant d'accomplir ces buts.

Un agent est dit rationnel [Newell, 1990] s'il utilise au mieux ses connaissances et ses capacités de réflexion. Il choisit d'agir au mieux pour satisfaire un but. Par extension, un agent est dit irrationnel, lorsque son développement arrive au point de manifester des états mentaux. L'agent peut alors avoir des croyances et exprimer un doute ou au contraire une certaine conviction. Il répond alors davantage à des impulsions internes plutôt qu'à une analyse rationnelle de la situation. L'agent cognitif devra apprendre de ses observations, des ses réussites et de ses échecs. Il modifiera en conséquence sa perception du monde.

II.5.3. Agents hybrides

Certains définissent un type d'agent supplémentaire communément appelé agent hybride. Il reprend les caractéristiques des agents réactifs et des agents cognitifs pour former une entité capable de se comporter encore différemment. La représentation en notation UML correspond au schéma ci-contre. Cette forme d'agent, très peu courante, est représentée en modélisation par objets par un héritage multiple qui n'est plus en vigueur dans les langages récents. Elle fut utilisée il y a quelques années dans la conception des premiers SMA. Nous évoquons juste son existence dans un souci d'exhaustivité mais nous n'en parlerons donc plus dans les prochains chapitres de ce document.



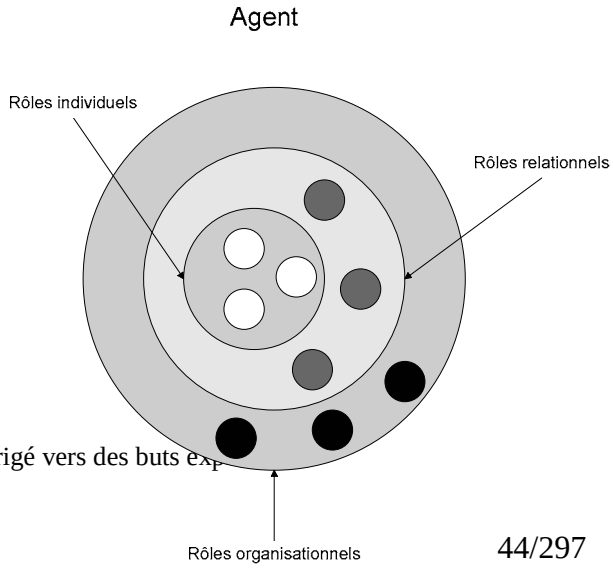
Nous verrons dans le chapitre IV quelle représentation nous avons choisi pour nos agents. Celle-ci se distingue particulièrement des représentations classiques présentées ici.

II.6. Le rôle

L'encyclopédie Hachette définit le rôle comme :

1) *une fonction ou un emploi*
2) *l'ensemble des conduites qui constituent l'apparence sociale de quelqu'un ou l'image qu'une personne veut donner d'elle-même et qui ne correspond pas à sa véritable personnalité « jouer un rôle ».*

Dans le cadre de notre étude, nous ne retiendrons que la première définition. Plus précisément, le rôle est une représentation abstraite d'une fonction ou d'un service proposé par un agent. La deuxième définition est néanmoins intéressante puisqu'elle intègre la notion de communauté, c'est-à-dire que l'agent tient un rôle parce qu'il est présent dans une société ou dans une organisation. Le rôle que va jouer un personnage ou un agent



[21] Le comportement d'un agent est téléonomique s'il est dirigé vers des buts exp

va donc être une interface avec son entour, c'est-à-dire qu'un individu va se faire une représentation d'un autre individu par le truchement du rôle joué.

Un rôle peut être attribué dynamiquement à un agent. Il peut par exemple jouer un rôle de client ou de fournisseur ou même les deux simultanément. D'autres rôles peuvent être définis comme le rôle de médiateur qui se charge de la mise en relation des clients et des fournisseurs.

Dans les premières architectures décentralisées, un agent présentait directement ses fonctionnalités comme un objet possède des méthodes. Pour des raisons de découplage fonctionnel propre au génie logiciel et pour des raisons conceptuelles liées à la notion de sociétés d'agents, on préfère aujourd'hui présenter les fonctions d'un agent via une interface appelée le rôle.

Nous verrons les méthodologies de conception des SMA dans un prochain chapitre – chaque méthodologie peut appréhender le rôle de différentes façons. Certaines proposent d'associer potentiellement plusieurs rôles à un agent. D'autres spécifient qu'un rôle est au contraire tenu par plusieurs agents. Dans une méthodologie comme Aalaadin [Ferber & Gutknecht, 1997], chaque agent peut prendre en charge plusieurs rôles et un rôle identique peut être tenu par plusieurs agents mais le rôle sera toujours local à l'agent. C'est à dire que si un rôle nécessite lui même une organisation d'agent, on associera tout de même le rôle à un agent et l'agent pourra acquérir une compétence qui pourrait elle même être composée d'agents.

Dans [Drogoul, 1998], un agent est vu comme un ensemble de rôles, parmi lesquels on peut distinguer trois niveaux :

- Les rôles individuels qui sont les différents comportements que les agents sont capables de tenir sans se soucier de la stratégie choisie pour les tenir.
- Les rôles relationnels qui concernent comment ils choisissent d'interagir avec un autre (en activant ou en désactivant les rôles individuels) avec le respect des dépendances mutuelles de leurs rôles individuels.
- Les rôles organisationnels ou comment les agents peuvent gérer leurs interactions pour devenir ou rester organisés (en activant ou en désactivant des rôles relationnels).

II.7. Le comportement

Le comportement caractérise l'ensemble des propriétés apparentes que l'agent manifeste dans son environnement. Un comportement est une réponse à un événement ou une situation. Dans le contexte des agents, un événement est une chose qui se produit et qui change l'environnement ou l'état de l'agent. Un agent défini son comportement en fonction des événements qui lui arrivent. Un événement peut être par exemple l'arrivée d'un courrier électronique. Ca peut être aussi un événement produit à une date ou heure particulière. D'une manière générale il s'agit de l'introduction d'un nouvel élément dans un système. Lorsqu'un événement se produit, l'agent doit l'analyser et l'évaluer pour produire une réponse adaptée.

La décision d'un agent va donner lieu à une action. Ce processus constitue le comportement de l'agent. Une action va modifier l'état du monde. Par extension, l'action n'est pas seulement envisagée comme le résultat de ce que font les agents mais comme le résultat des réactions du monde aux influences des agents. Il s'agit donc d'une distinction entre le geste et les conséquences du geste.

Le comportement d'un agent peut être considéré d'un point de vue externe ou interne à l'agent. Le comportement externe d'un agent correspond à l'observation d'une suite d'actions entreprises par celui-ci, alors que son comportement interne correspond à l'expression de ses capacités de perception, de décision, et d'action.

Pour nous, le comportement interne d'un agent exprime quand et comment un agent va utiliser ses connaissances, ses savoir-faire et ses facultés de perception de l'environnement, ou de communication pour décider de ses actions. Pour un concepteur, la définition du comportement peut alors se résumer ainsi : *"comment assembler les différentes parties d'un agent de manière qu'il accomplisse les actions que l'on attend de lui ? "*.

II.7.1. Modélisation et représentation du comportement

Il ne s'agit donc pas de décrire l'agent et son comportement uniquement par les buts, les plans, et les actions dont il peut disposer et les décisions qu'il peut prendre, mais aussi de décrire comment et quand un agent va communiquer et échanger des informations avec les autres. Il existe plusieurs formalismes pour décrire les comportements d'agents basés sur des actions et leurs conséquences :

- L'action comme transformation d'un état global : le modèle fonctionnel par transformation d'états est le plus classique. Une action *op* est alors définie comme une transition d'état, c'est-à-dire comme un opérateur dont l'exécution produit un nouvel état. Il est notamment utilisé dans le cadre de la planification en I.A.
- L'action comme réponse à des influences : on peut utiliser une extension du modèle précédent. Il s'agit d'ajouter les conséquences des actions simultanées des agents.
- L'action comme processus informatique : une action peut être vue comme un ensemble d'événements produits et consommés par des processus informatiques. On peut alors décrire un comportement en ne s'intéressant qu'aux états initiaux et finaux d'actions considérées comme atomiques. On utilise dans ce cas des automates à états finis ou des réseaux de Petri pour formaliser les comportements des agents et de l'environnement.
- L'action comme modification locale : cette action se propage le long d'un réseau d'automates. On utilise les modèles locaux, en particulier les automates cellulaires qui sont bien adaptés notamment pour l'environnement mais moins adaptés pour la description des agents.
- L'action comme déplacement physique : dans des modèles physiques et géométriques, les actions sont considérées comme des déplacements dans un espace géométrique. On utilise des champs de vecteurs et des champs scalaires dans un espace euclidien par exemple. Ce modèle est bien adapté pour les agents mobiles.
- L'action comme commande : dans un modèle cybernétique, l'action est une commande. On s'intéresse à ce que l'agent perçoit de l'environnement et aux commandes qu'il envoie à ses organes effecteurs pour satisfaire ses buts. On fait varier un certain nombre de paramètres d'entrées appelés variables de commmande, d'un système physique pour obtenir des valeurs particulières des variables de sortie. Ce modèle est bien adapté pour les agents réactifs.

[Ferber, 1995] utilise le formalisme BRIC (Basic Representation of Interactive Components) qu'il a mis au point pour décrire les comportements des agents et leurs interactions. Chaque composant BRIC peut être considéré comme un module, i.e. une entité logicielle définie par l'ensemble des opérations qu'elle peut accomplir, et qui sont figurées sous la forme de bornes d'entrées/sorties. Les composants interagissent par le biais de liens de

communication établis entre leurs bornes, une borne de sortie étant liée à une ou plusieurs bornes d'entrées d'un autre composant. Les composants BRIC peuvent être utilisés pour modéliser des agents et des systèmes multi-agents, ce qui présente l'intérêt d'être récursif ; un agent est un composant BRIC défini lui-même en terme de composants plus élémentaires. Un SMA peut alors être décrit comme un ensemble de composants interconnectés. Ce formalisme est intéressant mais assez difficile à utiliser lorsqu'on souhaite modéliser des SMA avec de nombreux agents et de nombreuses interactions avec le monde.

II.7.2. Modélisation en fonction du type d'agent

Différents travaux ont proposé des modèles intégrant toutes les capacités de l'agent : perception, communication, décision et action [Gaussier & al., 1998], [Gleizes & al., 1994]. Ces travaux proposent des techniques de raisonnement et de prise de décision (introspection, croyances, intentions, engagements, etc.). Ces techniques considèrent généralement qu'un agent possède des connaissances et un savoir et qu'il interagit ou dépend d'autres agents du système pour exploiter ce savoir. Le fonctionnement d'un agent est alors centré sur ses facultés de décision et de raisonnement qui prennent place dans un contexte social d'interactions et de dépendances.

Il y a plusieurs façons d'évaluer un événement en fonction de la nature de l'agent. Si l'agent est de type réactif, l'événement sera analysé formellement et l'agent déclenchera une procédure très particulière en réaction immédiate à cet événement. Il n'y a pas dans ce cas de réelle prise de décision. Un événement implique une action. On parle d'action réflexe. En revanche, lorsqu'il s'agit d'un agent cognitif, l'événement est analysé, puis l'agent va devoir décider une action à mener. Cette prise de décision peut prendre un certain temps et peut être le résultat de différents algorithmes. Il est possible que deux agents cognitifs, pourtant du même type, ne parviennent pas à la même décision. Chaque agent possède un « état mental » qui va influer sur sa prise de décision. L'état mental en question repose sur des tendances comportementales. Un agent va posséder certaines croyance ou centaines informations. Il peut désirer réaliser un but plutôt qu'un autre et avoir des intentions propres [Hubert, 1999]. Cela se traduit par une configuration différente de l'agent, ce qui induit des attitudes différentes. Par conséquent en fonction de son vécu et de son espace d'états internes, il prendra une décision peut être différente d'un autre agent. Un agent dit cognitif est plagié sur un être humain. Vous placez quelques humains dans une situation similaire, aucun ne se comportera exactement de la même façon qu'un autre car chacun a un vécu différent et une perception différente des choses donc cela engendre des prises de décisions ou des comportements différents. Autant il est simple de contrôler un agent réactif pour lui faire faire exactement ce qu'on veut, autant il peut être singulièrement complexe de vouloir donner une tendance à un agent cognitif et de lui imposer un comportement. Le problème se rapproche un peu de celui des réseaux neuronaux. On ne sait jamais s'il faut ajouter une couche cachée ou augmenter les interconnexions ou encore modifier tel ou tel paramètre pour obtenir un comportement clairement identifiable [Dreyfus & al., 2002].

II.7.3. Prévision d'un comportement

Le comportement d'un agent réactif est déterministe, celui d'un agent cognitif ne l'est pas véritablement ou plutôt il nous échappe puisque son état mental est une sorte de boite noire dont nous ignorons quasiment tout (sauf peut-être pour le concepteur). En outre, contrairement à un agent cognitif, un agent réactif ne peut pas apprendre de nouvelles choses. Il peut néanmoins changer de rôle si son entour se déstabilise. C'est le cas avec les fourmis. La très grande majorité de la communauté SMA considère une fourmi comme un agent réactif. Cependant les fourmis ont un rôle particulier à tenir dans la fourmilière. Elles peuvent

être des guerrières ou s'occuper des larves [Drogoul, 1993]. Des expériences ont montré que si on supprime une grande partie des fourmis jouant un rôle identique, celles jouant un autre rôle vont s'adapter et prendre un nouveau rôle pour rééquilibrer le système. C'est l'exemple même de l'auto-organisation [Bonabeau & al., 1999].

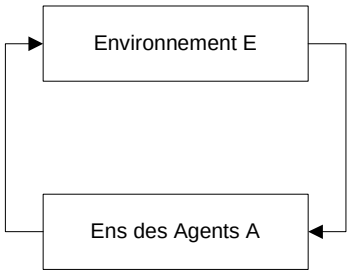
Si le comportement individuel d'un agent réactif est considéré comme déterminisme il n'en va pas de même pour un système composé d'agents réactifs. On est ici dans le cadre d'un comportement global.

Un concepteur ne contrôle pas toujours précisément les évolutions de l'univers des agents car les SMA obéissent aux principes des phénomènes chaotiques [Gleick, 1999]. Nous avons vu dans le chapitre précédent qu'une modification minimale des conditions initiales ou l'effet de valeurs aléatoires est amplifiée par les nombreuses interactions entre les agents et il devient, par voie de conséquence, impossible de prévoir finement l'évolution d'une situation. En d'autres termes, un observateur du système ne peut prévoir avec précision vers quel espace d'états convergera le SMA. L'expérience des insectes qui regroupent les copeaux de bois [Quinqueton & Hamadi, 1999] est un bon exemple. On ne sait pas à l'avance à quel endroit vont se situer les tas de copeaux à la fin des itérations, même si on peut tout de même deviner le résultat global de la simulation. Faut il y voir là une situation réellement non déterministe ou doit on imputer cela à une limite intrinsèque de notre perception humaine ?

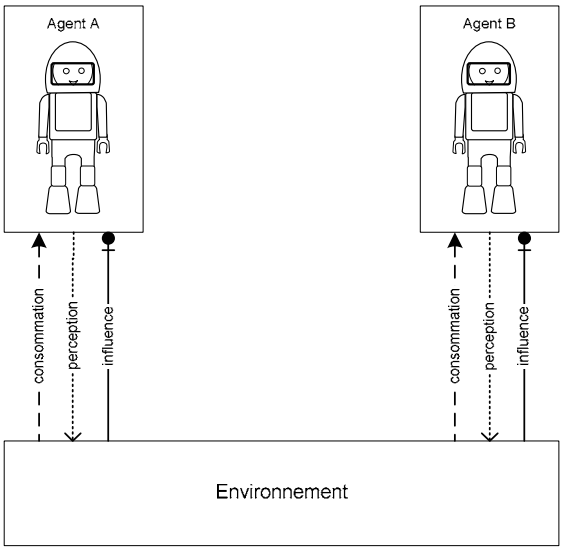
Malgré ce qui vient d'être dit, il est cependant possible dans certains cas de construire des systèmes qui se stabilisent comme on le souhaite en adoptant des comportements correctement définis et paramétrés. Le SMA peut alors s'auto-adapter à une situation inattendue ou inconnue et de ce fait, peut se montrer quasiment insensible à des perturbations provoquées par son environnement. Le SMA s'adapte ainsi à une modification de structure plus ou moins profonde de son environnement. On peut trouver des applications de ces phénomènes adaptatifs dans les problèmes d'optimisation (par exemple l'optimisation du routage de paquets IP sur un réseau en fonction de la charge).

II.8. L'environnement

Un agent ne peut exister sans environnement, de même qu'on considère qu'un agent n'existe pas sans système multi-agents. L'environnement est une structure dans laquelle l'agent évolue. Un agent va agir sur son environnement et l'environnement va agir sur l'agent. On retrouve cette notion de boucle de rétro-action dans la « commande » de Francisco Varela [Varela, 1989].



Cette structure peut être centralisée et on la représente comme un bloc monolithique. Elle peut aussi être formée de cellules réunies en réseau et on la représente sous forme d'un automate cellulaire. On parle dans ce dernier cas d'environnement distribué. Chaque cellule se comporte comme un environnement centralisé en miniature.



La première catégorie est la plus répandue aujourd'hui. Un environnement peut être un espace physique où se trouvent des agents et des objets.

Dans un environnement centralisé, les agents ont accès à la même structure. Ils produisent des influences sur l'environnement et celui-ci répond en renvoyant les éléments consommés ou perçus. Les problèmes de poursuite comme les proies/prédateurs sont un exemple d'environnement centralisé. Les systèmes comme MACE [Gasser & al., 1987] ou MICE [Durfee & Montgomery, 1989] utilisent de tels environnements.

Ce qui distingue un environnement distribué d'un environnement centralisé tient en quatre points :

- L'état de la cellule dépend des autres cellules qui l'environnent et donc aussi des influences produites par les agents dans les cellules voisines.
- La perception des agents s'étend généralement au-delà d'une cellule, ce qui fait qu'il n'est pas possible d'envoyer à l'agent uniquement les informations de l'état de la cellule sur laquelle il se trouve.
- Les agents se déplacent de cellule en cellule, ce qui fait qu'il faut gérer les liens que les agents entretiennent avec une cellule particulière.
- Des signaux peuvent se propager de cellule en cellule. Cette propagation prend un certain temps et il faut alors synchroniser le déplacement des signaux avec les mouvements des agents.

L'environnement est en général très spécifique à l'application dans laquelle les différents composants agents sont réifiés.

Le problème majeur du SMA est assurément la représentation du monde. Il faut avant toute chose décrire de façon précise le monde dans lequel les agents vont évoluer. La description des lois physiques pourra être représentée sous forme de règles. Mais une certaine connaissance doit préalablement exister pour accueillir l'agent dans le monde dans la mesure ou l'agent peut se définir comme l'image duale de son environnement. Il faut qu'il y ait une adéquation correcte entre le système concret, c'est à dire réel et le système représenté, c'est à dire le modèle. Une organisation est la résultante d'une interaction entre agents. En outre, l'organisation régule d'elle-même globalement le comportement des agents. Puisqu'un agent produit des actions sur l'environnement dans lequel il évolue, il peut agir dans le sens de rendre cet environnement moins hostile par exemple. Il modifie sa structure interne selon les actions qu'il va entreprendre pour agir sur son environnement. Il peut ainsi devenir moins

sensible à un environnement difficile et défavorable ou au contraire augmenter sa sensibilité dans un environnement moins hostile. Il filtre donc différemment les influences externes selon leur nocivité.

II.9. L'autonomie

Un agent possède des buts qu'il peut faire évoluer tout en restant maître des décisions qu'il prend. C'est donc une entité active possédant un ensemble de motivations et un mécanisme de régulation [Tzafestas, 1995]. Un agent est donc qualifié d'autonome par rapport à ses caractéristiques physiques ou conceptuelles. Ces deux aspects de l'autonomie correspondent aux définitions suivantes :

Un agent est défini comme autonome physiquement s'il s'agit d'un processus intelligent avec une mémoire privée (savoir et données) et une aptitude à [Gleizes & al., 1994]. Il est donc une entité active différente et indépendante des autres entités du système pour son exécution. La notion d'autonomie, d'un point de vue conceptuel, fait référence à l'autonomie de l'agent au sens où il possède ses propres buts, ses motivations ou des capacités pour résoudre seul une (ou plusieurs) tâche(s) précise(s).

Dans le cadre d'un système multi-agents, l'agent existe par son caractère d'autonomie mais reste socialement attaché aux autres agents. Il est possible de trouver des agents intelligents rendant des services à l'utilisateur comme des agents de recherches sur le web mais cela ne constitue pas intrinsèquement un système multi-agents puisque l'agent est chargé d'une tâche et est libre de communiquer ou non avec d'autres agents pour satisfaire son objectif..

II.9.1. L'autonomie par rapport à ses propres buts

Un agent autonome est un agent qui peut atteindre seul ses buts. Il n'est pas nécessaire qu'il coopère avec d'autres agents pour y parvenir. Il peut néanmoins décider de coopérer dans l'objectif d'atteindre son but avec une meilleure performance. Toute la difficulté consiste à comprendre comment un agent choisit son but ou plus simplement quel but un agent recherche t-il spontanément. Nous tenterons d'y répondre dans le chapitre 4 avec la notion de circulation d'énergie.

Cette notion d'autonomie, de même que la notion de localité[22] de la tâche à exécuter, a été utilisée pour classer les comportements possibles d'un agent (cohabitation, collaboration, coopération et distribution) au sein d'une société [Demazeau, 1995].

II.9.2. L'autonomie par rapport à ses motivations

Un agent autonome est un agent qui a la liberté de ses choix pour interagir [Gasser, 1992]. C'est à partir de son état interne, de ses buts, de sa représentation du contexte de la résolution qu'il décide de coopérer ou non [Castelfranchi, 1995]. Un agent n'est donc pas considéré ici comme une entité forcément bienveillante et asservie par ses relations avec les autres entités du système. Il peut décider par lui-même des motifs le poussant à interagir et avoir ses propres références.

[22] Une tâche est locale si un seul agent est concerné par celle-ci. Elle est globale si plusieurs agents sont concernés.

II.9.3. La vie artificielle

Afin de doter des agents d'un comportement plus « réel », on tente de développer leur autonomie au maximum. Pour cela ont fait appel à des techniques issues de la vie artificielle. Les agents évoluent vers de véritables animats qui constituent le pilier des études sur la vie artificielle.

Etymologiquement, un système est dit autonome (auto-nomos) s'il est régi par des lois qu'il a lui-même édictées. Dans [Varela & Maturana, 1994], l'autonomie a un sens précis : un système est autonome s'il est autopoïétique. Cela signifie que le système est constitué d'un ensemble de composants dont les transformations et les interactions ne cessent de reproduire l'organisation dont ils sont les composants. Francisco Varela parle de clôture opérationnelle entre le système et l'environnement, c'est-à-dire entre ce qui reste invariant (le système) et ce qui se transforme (l'environnement).

Ce principe fort intéressant dans le cadre de la vie artificielle, est parfois critiqué dans le domaine du SMA puisque l'individualité des agents est donnée par le concepteur. Le concept de viabilité fut introduit dans [Varela & Bourgine, 1992]. Un système est dit viable si sa dynamique reste à l'intérieur d'un domaine de tolérances. Le comportement d'un agent doit donc le contraindre à rester dans son domaine de viabilité s'il veut survivre. Il doit donc déterminer une trajectoire dans l'espace des configurations possibles qui lui permette de rester dans sa zone de viabilité. Ainsi, l'agent doit se nourrir, fuir les prédateurs, éviter les accidents etc. s'il veut survivre. Tout être vivant possède au moins l'objectif de rester en vie, c'est la raison pour laquelle, il s'alimente, il cherche à fuir ses prédateurs etc.

Pour qu'une entité biologique conserve son vivant, son état E doit rester à l'intérieur de son domaine de viabilité V au cours de son évolution (le temps t). Ceci est une fonction d'évolution f, implémentable dans un automate cellulaire.

$$E(t+1) = f\big(E(t)\big) \text{ avec } \forall E, f(E) \subseteq V$$

Mises à part quelques exceptions, l'utilisation massive de mathématiques dans la biologie théorique a montré sa non adéquation pendant les 30 ou 35 dernières années. L'IAD, contrairement à la branche plus classique le l'I.A., a les moyens de s'intéresser aux processus du vivant. La modélisation en IAD semble être une bonne alternative susceptible de caractériser le domaine étonnamment complexe du vivant. Plus particulièrement, l'intersection entre les SMA et la vie artificielle constitue une voie intéressante et prometteuse.

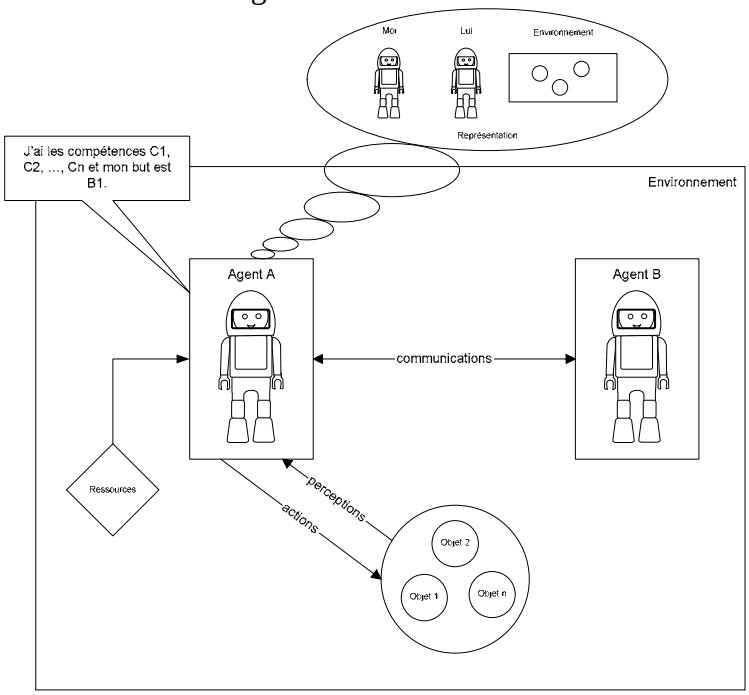
II.10. L'agent comme entité d'un système

II.10.1. Définition d'un SMA

Nous avons vu qu'un agent est une entité relativement autonome présente dans un système qui lui permet d'évoluer, de communiquer etc. Ce système est appelé un système multi-agents. [Ferber, 1995] défini un système multi-agents comme un système composé des éléments suivants :

- un environnement E identifié et muni d'un système de repérage dans l'espace (souvent Euclidien).
- un ensemble d'objets O passifs pouvant être perçus, créés, modifiés ou détruits par des agents.

- un ensemble d'agents A actifs ($A \subseteq O$)
- un ensemble de relations R qui unissent des objets entre eux
- un ensemble d'opérations Op offrant la possibilité aux agents de A de percevoir, produire, consommer, transformer et manipuler des objets de O
- un ensemble de lois universelles qui sont des opérateurs chargés de représenter l'application des actions des agents sur le monde et la réaction du monde à ces actions.



Un SMA peut-être ouvert (cas d'un magasin où des clients entrent et sortent librement) ou fermé (l'ensemble d'agents reste le même comme dans un match de football).

Un SMA peut être homogène, c'est-à-dire que tous les agents sont construits sur le même modèle (ex: une réunion de travail, une colonie de fourmis) ou hétérogène, c'est-à-dire que les agents sont de modèles différents et/ou de granularité différentes (ex: un éco-système).

Un agent a une représentation partielle de son environnement, il peut utiliser des objets de son environnement et communiquer avec d'autres agents. Lorsqu'il se fixe le but de réaliser une action, il a besoin de compétences et de ressources.

L'intérêt du SMA réside dans la faculté innée des agents le constituant, d'élaborer des substrats communicationnels communs offrant la possibilité d'allouer des tâches de façon dynamique et de coordonner des actions. Un agent agit sur son environnement et l'environnement agit sur ce même agent. C'est cette boucle de feed-back qui définit la notion d'autopoiétisme [Varela, 1989].

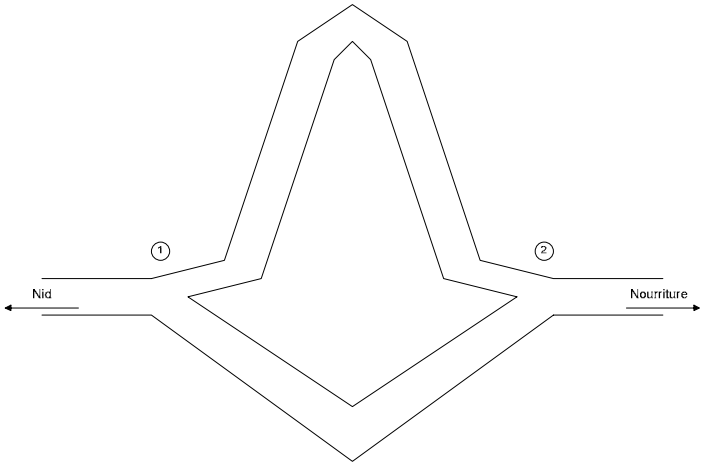
II.10.2. Vers des systèmes multi-agents adaptatifs

Un système multi-agents adaptatif présente des propriétés émergentes et une faculté à se réorganiser dynamiquement.

Les éthologistes utilisent la modélisation et la simulation pour comprendre le comportement animal. Des travaux assez récents dans le domaine du comportement collectif des insectes sociaux suggèrent que des modèles d'auto-organisation permettent d'expliquer comment des comportements complexes émergent au niveau de la colonie à partir d'interactions entre insectes très simples. Les algorithmes de contrôle et d'optimisation inspirés de modèles de fourragement collectifs chez les fourmis ont connu un succès inattendu

et sont connus sous le nom de Ant Colony Optimisation (ACO) ou Ant Colony Routing (ACR). On trouve notamment des applications pratiques de l'ACR dans l'optimisation des réseaux de télécommunication.

La meilleure illustration de la capacité des fourmis à optimiser est l'expérience du double pont. Dans cette expérience, les fourmis sont capables de sélectionner le plus court chemin pour aller du nid à une source de nourriture grâce au dépôt et au suivi de pistes chimiques. La substance chimique déposée sur l'environnement est appelée phéromone. En réalité, les fourmis sélectionnent le plus court chemin de manière statistique. Il se peut que la colonie se retrouve piégée sur la branche la plus longue si par hasard celle-ci a été la première à être marquée. Les chercheurs ont amélioré ce principe naturel en ajoutant un paramètre d'évaporation de la phéromone qui change les propriétés de convergence de l'algorithme. Dans la figure suivante, la branche la plus longue est r fois plus longue que la branche courte. Si r est suffisamment grand, la branche courte est sélectionnée par la colonie car les premières fourmis à revenir de la source de nourriture vers le nid sont celles qui ont choisi la branche courte. En revanche, si la branche courte est présentée à la colonie après la branche longue, cette dernière sera marquée en phéromone et la branche courte ne sera pas sélectionnée.



Autre application des SMA, celle du voyageur de commerce (TSP). Ce problème classique, typiquement NP-complet, consiste à trouver le chemin le plus court qui relie n villes en visitant chaque ville une et une seule fois. On peut appliquer l'algorithme des phéromones pour résoudre ce problème d'optimisation. On définit d_{ij} comme la distance entre les villes i et j et τ_{ij} comme la concentration en phéromone sur l'arête reliant i à j. τ_{ij} prend initialement une petite valeur τ_0, la même pour toutes les arrêtes (i,j). Un groupe de m fourmis est déployé sur des villes aléatoires. Chaque fourmi, indexée par k, construit un tour complet qui passe une et une seule fois par chacune des villes. A tout instant, la fourmi possède une liste J_k des villes qu'il lui faut encore visiter.

Une fourmi située à la ville i se déplace vers la ville j, choisie parmi les villes qui n'ont pas encore été visitées suivant la probabilité :

$$p_{ij}^{k} = \frac{\left([\tau_{ij}]^{\alpha} . [d_{ij}]^{-\beta} \right)}{\sum_{l \in J_k} [\tau_{il}]^{\alpha} . [d_{il}]^{-\beta}}$$

Où α est un paramètre qui détermine les influences de la phéromone et β un paramètre qui détermine les influences de la distance. Lorsque toutes les fourmis ont réalisé un tour, les pistes chimiques sont mises à jour selon :

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \Delta\tau_{ij}$$

où ρ est le taux d'évaporation et $\Delta\tau_{ij}$ le renforcement reçu par l'arête (i,j). $\Delta\tau_{ij}$ est proportionnel à la qualité des solution dans (i,j) qui a été utilisé par plusieurs fourmis. Si L_k est la longueur de tour T_k construit par la fourmi k, alors :

$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$ avec $\Delta\tau_{ij}^{k} = \frac{Q}{L_k}$ si (i,j) $\in$ T_k et $\Delta\tau_{ij}^{k} = 0$ sinon. Q est un paramètre positif.

Ce mécanisme de renforcement reflète l'idée que la densité de phéromone doit être plus faible sur un chemin long que sur le chemin court parce qu'un chemin long est plus difficile à maintenir. Le mécanisme d'évaporation des phéromones assure que le système ne converge pas prématurément vers une solution de qualité médiocre. Il faut alors jouer sur les paramètres α et β. Lorsque α=0, l'algorithme fait une recherche « gloutonne », dans laquelle la prochaine ville à visiter est sélectionnée sur la seule base de sa distance avec la ville actuelle. Lorsque β=0, la phéromone est seule utilisée pour guider la recherche. Il semble, d'après certaines simulations réalisées, que l'utilisation de la distance comme critère de sélection de la prochaine ville est une nécessité.

La version simple de l'ACO qui vient d'être présentée a des performances moyennes. Des versions plus sophistiquées incluent une procédure de recherche locale qui cherche à améliorer les solutions en évaluant les effets de petits changements ce qui fait un subtil équilibre entre exploitation et exploration. Avec ces évolutions, l'algorithme donne des résultats qui le place en tête des autres algorithmes comme la programmation évolutionnaire, le recuit simulé ou les algorithmes génétiques. Plus de détails peuvent être trouvés dans [Dorigo & Gambardella, 1997].

Ces expériences d'optimisation sont réalisées directement sans modèle d'organisation. La réorganisation dynamique et le concept d'émergence posent néanmoins de sérieux problèmes pour la phase de conception du système. En effet de nombreuses applications nécessitent la réalisation de logiciels caractérisés par une phase de spécification incomplète pour les raisons suivantes :

- le système doit évoluer dans un environnement dynamique et il est impossible de spécifier totalement toutes les situations auxquelles il aura à faire face,
- le système est ouvert,
- le système est complexe,
- il n'existe pas de solution algorithmique connue pour résoudre le problème,
- l'organisation interne du système n'est pas connue, a priori.

L'imprévu étant inhérent à la vie de ces systèmes, l'auto-organisation, qui correspond à un changement décidé de manière autonome, devient un moyen pour parvenir à surmonter les perturbations éventuelles de l'environnement [Marcia, 1996] et pour réaliser des systèmes adaptatifs. Cette auto-organisation apporte des caractéristiques non-algorithmiques et donc non programmées. Il s'agit d'un phénomène d'émergence de fonctionnalités qui apporte, entre autres choses, la robustesse du système. Un système multi-agents adaptif ou AMAS présente quelques caractéristiques :

- le système est plongé dans un environnement,
- le système réalise une fonction,
- le système est composé d'agents autonomes en interaction,
- chaque agent du système réalise une fonction partielle,
- l'organisation du système détermine le résultat du système.

Apprendre pour le système consiste donc à transformer sa fonction actuelle de manière autonome afin de s'adapter à l'environnement c'est-à-dire à changer l'organisation entre ses agents. Cet apprentissage doit lui permettre d'avoir une activité "correcte" dans l'environnement dans lequel il est immergé : ceci définit l'adéquation fonctionnelle d'un système. La question est de savoir quand et comment un système peut se transformer pour

tendre vers l'adéquation fonctionnelle. La théorie des AMAS [Gleizes & al., 1999] indique que pour tout système fonctionnellement adéquat, il existe au moins un système dont les agents sont en interactions coopératives. Ce résultat est très important car il permet de guider la conception d'un système multi-agents adaptatif. La première étape consiste à identifier les agents puis à faire en sorte qu'un agent de manière locale soit ou tende à être en interactions coopératives avec les autres ce qui, de par la théorie, assure l'adéquation fonctionnelle. La conception est ascendante et les agents du système vont localement s'organiser pour que le comportement collectif réalise la fonction globale.

De manière générale, un agent, dans la théorie des AMAS, est doté de compétences, de possibilités de communication ou d'interaction avec les autres agents et/ou avec son environnement, de croyances sur certains autres agents du système, d'aptitudes qui lui permettent de raisonner et d'une attitude sociale basée sur la coopération. Le comportement de chaque agent est spécifié de manière à ce qu'il essaye d'atteindre son objectif local tout en gardant des interactions coopératives avec les autres. Avant chaque action, l'agent analyse localement s'il est ou non en interactions coopératives. Plus précisément, il recherche s'il existe des situations non coopératives et, si tel est le cas, tente de les éliminer pour revenir à un état coopératif. La coopération est l'attitude sociale qui guide le comportement de chaque agent par la prise en compte de critères locaux. Un agent a donc deux rôles essentiels : le premier est de réaliser la fonction pour laquelle il a été spécifié et le second est d'agir sur l'organisation interne du système s'il rencontre des situations non coopératives dans le but de retourner à des situations coopératives.

II.10.3. Inconvénients des SMA

Les systèmes multi-agents sont de plus en plus utilisés dans les sciences de la vie et de l'environnement pour leur possibilité de représenter directement les différents acteurs biologiques, sociaux et économiques, leurs comportements et leurs interactions. Malgré leur intérêt, ils soulèvent plusieurs problèmes.

Un problème important porte sur la définition d'un modèle "sans biais", c'est-à-dire d'un modèle qui assure au modélisateur que l'outil informatique implémente bien le modèle que le modélisateur a décrit (on ne s'intéresse pas ici au problème de l'adéquation entre le modèle et la réalité qui est un problème de modélisation), et que les conséquences observées lors de la simulation sont bien directement des conséquences du modèle et non le produit d'un comportement non désiré du système informatique. L'un des biais les plus courants correspond au problème de la simultanéité des actions : les simulateurs existants ne savent pas bien prendre en compte des actions simultanées. Ainsi, deux agents qui, dans le modèle, devraient agir en même temps, vont être animés l'un après l'autre dans la simulation. Le problème n'est pas seulement technique, il est aussi théorique : il a nécessité le développement d'une théorie de l'action, le modèle "influence/réaction", une extension du calcul situationnel, qui permet de décrire des comportements simultanés d'actions. Une architecture de simulation de systèmes multi-agents fondée sur ce principe a été réalisée dans le cadre du DEA de F. Michel au LIRMM.

Un autre grand problème tient dans l'explosion du nombre de communication entre agents pour l'organisation du SMA. Les agents sont, on l'a dit, des entités autonomes plongées dans un environnement et très communicantes. Si les SMA peuvent apporter des solutions à des problèmes de nature intrinsèquement distribuée [Parunak, 1998], ils apportent avec leur solution une autre problématique qui est l'augmentation non négligeable des communications. Il arrive fréquemment qu'une application SMA soit moins rapide à l'exécution qu'une application non répartie. La simple répartition du calcul est loin de garantir automatiquement des temps de simulations plus courts. Une conception inadaptée de la

simulation pourra conduire à un nombre important de message de partage d'information entre agents simulés ou environnement transitant par le réseau, causant dans certains cas une exécution plus lente qu'une simulation non distribuée.

II.11. Interaction et coopération

Une interaction est une mise en relation dynamique de deux ou plusieurs agents par le biais d'un ensemble d'actions réciproques. Les interactions sont un élément indispensable à la constitution d'organisations sociales [Morin, 1977]. C'est grâce à la coopération que les agents peuvent accomplir des actions et dégager plus que la somme de leurs actions. L'interaction est donc le composant essentiel de toute organisation. C'est à la fois la source et le produit de la permanence de cette organisation. Un agent sans interaction avec d'autres agents n'est plus qu'un corps isolé dépourvu de caractéristiques adaptatives.

A contrario, c'est aussi en raison de la multitude d'agents dans une organisation qu'ils doivent coordonner leurs actions et résoudre des conflits. Résoudre des conflits implique la mise au point d'un protocole de négociation et donc implique un certain nombre de communications. C'est bien là que se situe le principal point faible des SMA : faire face à une quantité d'information considérable.

Traiter des interactions, cela consiste à décrire des mécanismes élémentaires permettant aux agents d'interagir, mais également d'analyser et de concevoir les différentes formes d'interactions que les agents peuvent avoir pour accomplir leurs buts. Il y a en réalité plusieurs facteurs qui influencent l'interaction des agents comme les relations de dépendance, la confiance et la capacité de persuasion [Esmahi, 2001].

La notion d'interaction suppose :

- Un ensemble d'agents capables d'agir et/ou de communiquer
- Des situations susceptibles de servir de point de rencontre entre agents : collaboration, utilisation de ressources limitées, régulation de la cohésion du groupe[23].
- Des éléments dynamiques permettant des relations locales et temporaires entre agents : communication, choc, champ attractif ou répulsif…
- Une certaines flexibilité dans les relations entre agents leur permettant à la fois de créer, de maintenir ou de briser les relations, c'est-à-dire de disposer d'une certaine autonomie.

Interagir implique une ou des communications. Communiquer est une forme d'action particulière qui tend à modifier l'état mental de l'agent destinataire et éventuellement de l'agent émetteur. Lorsqu'un agent sous-traite une tâche à un autre agent, cela provoque chez l'agent destinataire une intention d'accomplir cette tâche et donc modifie sont état mental. L'agent émetteur de l'ordre satisfait tout de même son objectif sans accomplir la tâche lui-même, ce qui contribue à faire aussi évoluer son propre état mental.

Les différentes formes d'interactions sont la collaboration et la coordination d'actions. La première consiste à répartir le travail entre plusieurs agents. La seconde s'intéresse à la manière dont les actions des agents sont organisées dans le temps et dans l'espace pour

[23] Un groupe se défini comme la notion primitive de regroupement d'agents [Ferber, 1995]. Dans sa forme la plus simple, un groupe peut être vu comme un moyen d'identifier par regroupement un ensemble d'agents. D'une manière plus évoluée, le groupe peut être vu comme un SMA usuel. Un point majeur de cette définition est que les différents groupes peuvent se recouper librement.

accomplir les buts. Il y a dans le cadre de l'éthologie bien des exemples d'interactions entre les animaux d'une même espèce [McFarlard, 1990]. D'une manière générale, les interactions entre individus jouent un rôle essentiel dans le développement d'un être vivant, que ce soit au niveau réactif ou cognitif. [Ferber, 1995] définit une situation d'interaction comme « un ensemble de comportements résultant du regroupement d'agents qui doivent agir pour satisfaire leurs objectifs en tenant compte des contraintes provenant des ressources plus ou moins limitées dont ils disposent et de leurs compétences individuelles ».

Dans un SMA, les principales situations d'interaction peuvent être classées par rapport à trois critères : les objectifs (buts) ou intentions des agents, les relations que les agents entretiennent envers les ressources qu'ils possèdent ainsi que les moyens (ou compétences) dont ils disposent pour parvenir à leurs fins.

1) Des agents seront dans une situation de collaboration ou d'indifférence si leurs buts sont compatibles. Si leurs buts ne sont pas compatibles, ils seront dans une situation d'antagonisme.

2) Les ressources dont disposent les agents sont une composante très importante dans une situation d'interaction. Une ressource est un élément environnemental ou matériel utile à la réalisation d'une action. Par conséquent, tout agent requiert des ressources pour accomplir ses tâches. La quantité de ressources limitée est souvent une source de conflit entre agents.

3) Si un agent a besoin de ressource pour accomplir une tâche, il a également besoin d'une ou de plusieurs compétences. S'il dispose de toutes les compétences requises pour accomplir sa tâche, il peut l'exécuter seul. Au contraire, il se peut que l'agent ne possède pas toutes les compétences nécessaire, auquel cas, il est contraint de travailler en collaboration avec un autre agent ayant les compétences manquantes. Il peut ainsi sous-traiter une tâche à un autre agent. Les interactions sont donc dans ce cas très bénéfiques puisque les actions de chacun des agents contribuent à satisfaire un objectif commun. Le système résultant de ces interactions dispose alors de propriétés nouvelles qui s'expriment parfois comme une fonctionnalité émergente.

La coopération est une forme d'interaction. Elle consiste à établir qui fait quoi, avec quel moyens, de quelle manière et avec qui. Cela sous-entend qu'il faut trouver des solutions aux différents sous-problèmes que constitue la collaboration par répartition de tâches, la coordination d'actions et la résolution de conflits. La coopération se résume donc par la formule :

Coopération = collaboration + coordination d'actions + résolution de conflits

Ces concepts sont expliqués plus en détails dans les paragraphes suivants. La résolution de conflit supposant une négociation pour lever ce conflit. Plusieurs agents coopèrent ou sont dans une situation de coopération si l'une des conditions suivantes est vérifiée :

- L'ajout d'un nouvel agent permet d'accroître différentiellement les performances du groupe
- L'action des agents sert à éviter ou à résoudre des conflits potentiels ou actuels.

Les situations d'interactions sont classées dans le tableau suivant :

Buts	Ressources	Compétences	Type de situation	Catégorie
compatibles	suffisantes	suffisantes	indépendances	indifférence
compatibles	suffisantes	insuffisantes	collaboration simple	coopération
compatibles	insuffisantes	suffisantes	encombrement	

compatibles	insuffisantes	insuffisantes	collaboration coordonnée	
incompatibles	suffisantes	suffisantes	compétition individuelle pure	antagonisme
incompatibles	suffisantes	insuffisantes	compétition collective pure	
incompatibles	insuffisantes	suffisantes	conflits individuels pour des ressources	
incompatibles	insuffisantes	insuffisantes	conflits collectifs pour des ressources	

II.12. Coordination d'actions

Des agents qui travaillent dans un SMA accomplissent un certain nombre de tâches. Il existe également des tâches supplémentaires liées à la mise en relation des autres tâches pour améliorer globalement le déroulement. Il s'agit d'une articulation des actions individuelles de chacun des agents de manière à ce que l'ensemble du système se comporte de façon cohérente et performante.

Ces tâches supplémentaires sont appelées des tâches de coordination. [Malone, 1988] décrit la coordination d'actions comme « l'ensemble des activités supplémentaires qu'il est nécessaire d'accomplir dans un environnement multi-agents et qu'un seul agent poursuivant les mêmes buts n'accomplirait pas ».

La coordination des actions est donc l'une des principales méthodes pour assurer la coopération entre agents autonomes. Il y a plusieurs raisons pour utiliser une coordination : dans un SMA, les ressources sont limitées et il faut bien souvent éliminer les actions inutiles qui consomment des ressources. On recherche donc une réduction des coûts. En outre, les agents ont besoin d'informations et de résultats que seuls d'autres agents peuvent fournir. Cela est vrai dans tout système informatique lorsque des processus doivent être synchronisés. (nous verrons par la suite que dans l'implémentation de notre middleware, nous cherchons davantage à coordonner les tâches que le agents eux-mêmes.)

Coordonner des agents ou des tâches est très difficile car s'il existe une action coordonnée, toute action d'un agent Ai dépend des actions de tous les autres agents $\{A_1, \ldots, A_n\}$, sachant que leurs actions dépendent elles aussi des actions de tous les autres agents. Coordonner les actions de ces agents revient à résoudre un système à n-équations et n-inconnues, ce qui est impossible à résoudre aujourd'hui. On s'en remet donc à des techniques heuristiques. On pourra néanmoins se reporter à l'approche de [Zeghal, 1993] pour des cas particuliers où les opérations que l'agent Ai exécute au temps t, peuvent se traduire sous la forme de vecteurs (vitesse, accélération) dans un espace géométrique. A noter qu'il existe des modèles de coordination pour les agents tels que ECM avec son langage associé STL++ [Schumacher & al., 1988].

On note plusieurs formes de coordination d'action :

- La coordination par synchronisation : c'est la forme la plus basique pour laquelle les actions sont décrites précisément au niveau de leur enchaînement. Très utilisée en automatisme, cette forme de coordination se formalise souvent par des réseaux de Petri.
- La coordination par planification : elle s'exécute en deux phases. La première consiste à créer un ensemble de plans d'actions qui décrivent précisément les actions à effectuer pour atteindre un but. La seconde est l'exécution de l'un de ces plans. Du fait de la dynamique des SMA, l'environnement dans lequel évoluent les agents change en permanence et de ce fait les agents sont contraints de revoir leur planification en temps

réel, ce qui est très difficile à implémenter. Amal El Fallah a travaillé depuis quelques années sur cet épineux sujet [ElFallah & Haddad, 1996].

- La coordination réactive : c'est en quelque sorte le contraire de la planification. Des agents réactifs gèrent leur boucle de perception-action en temps réel. En fonction d'un stimulus de l'environnement, l'agent déclenche spontanément une action réflexe. Il n'y a donc pas du tout de planification. Bien souvent cette approche n'assure pas une optimisation du système sur le plan de l'efficacité mais assure une robustesse du système et une grande adaptabilité. Ce type de coordination est celui qu'utilisent les bancs de poissons, les oiseaux migrateurs ou les meutes de loups. Des simulateurs comme ICHTYUS [Mesle, 1994] ont montré la validité de cette approche.
- La coordination par réglementation : cela consiste à édicter des règles de comportement qui visent à éliminer les conflits potentiels entre agents. On pourrait appeler ces règles des heuristiques. C'est une extension d'une coordination réactive avec des règles d'un niveau plus haut. Le meilleur exemple est le code de la route.

Lorsqu'un conflit apparaît entre deux ou plusieurs agents, il y a deux méthodes pour le résoudre : l'arbitrage et la négociation.

L'arbitrage consiste à définir des règles de comportement qui agissent comme des contraintes sur l'ensemble des agents. Le résultat global a pour effet de limiter les conflits et donc empêcher la déstabilisation du système. C'est ni plus ni moins le système que nous utilisons dans notre société pour définir le comportement adéquat des individus ; les textes de lois constituent une forme de coordination par réglementation et la justice a pour objectif de faire respecter ces lois. Cela implique que les agents soient capables d'un libre-arbitre et soient donc très cognitifs. L'implémentation de ce genre d'agents dépassant nos capacités actuelles, nous n'en parlerons pas plus longtemps. Il est néanmoins possible de donner à ces agents cognitifs des règles de comportements et lorsqu'une règle n'a pas été édictée pour résoudre une situation de conflit, un agent arbitre décide d'une action à accomplir pour régler la situation. Dans des cas très restreints, l'agent arbitre prend une décision, même si ce n'est pas la meilleure, et débloque la situation. Il peut donc s'agir d'un arbitre réactif qui choisi aléatoirement entre deux alternatives dans le seul but de débloquer le système.

Une façon de régler un conflit est l'utilisation d'un protocole de négociation. Deux agents en conflit vont entrer en communication pour trouver un accord bilatéral. Cette forme de résolution a été particulièrement étudiée par les américains avec Lesser [Conry & al., 1988], Durfee [Durfee & al., 1989] et Sycara [Sycara, 1989]. Cette négociation ne fonctionne évidemment qu'avec des agents cognitifs.

Dans le cadre d'agents purement réactifs, incapable de manipuler de la sémantique, les conflits sont souvent rendus impossibles par la description précise que l'on peut faire de l'environnement et par des règles de fonctionnement intégrées aux agents. Nous proposons dans les prochains chapitres une autre approche basée sur le concept de viabilité des animats qui permettra à des agents réactifs de ne pas rester sur une situation d'inter-blocage (deadlock). Cette approche est basée sur un système d'échange d'énergie, chaque agent désirant maximiser sa fonction énergie.

Philippe Mathieu au Laboratoire d'informatique de Lille travaille aussi sur le problème de la négociation. Il propose quatre exemples de situations permettant d'initier des processus de négociation entre les agents: la vente aux enchères, le troc, la conquête de territoires (RISK) et la prise de rendez-vous. Un modèle de négociation est appliqué ensuite à chacun de ces exemples. La gradation de la complexité de ces exemples permet de décrire la

négociation d'un agent vers un autre, d'un agent vers n autres agents, les processus simultanés de négociation et les interdépendances entre les négociations terminées et celles en cours.

II.12.1. Une coordination sans communication

Il existe une autre forme de coordination, qui utilise une forme de communication dite indirecte, c'est-à-dire en relayant l'émetteur et le récepteur via l'environnement. On parle aussi de stigmergie. Elle fut étudiée par les chercheurs en biologie [Deneubourg & al., 1990a], en SMA réactifs et par quelques autres chercheurs tels que Stan Franklin au département des mathématiques de l'Université de Memphis [Franklin, 1997]. Elle suggère que certains exemples dans la nature nous invitent à considérer une coordination sans communication basée probablement sur un principe de synergie. Des observations de scènes de chasse chez certains animaux sauvages sont assez étonnantes. Donald Griffin dans [Griffin, 1984] rapporte une scène de « stratégie » de chasse impliquant une coordination apparente de cinq lionnes.

Franklin cite aussi l'exemple des jumeaux de Hensel, des siamois utilisant un seul corps mais ayant des systèmes nerveux disjoints. Il s'avère qu'ils parviennent à accomplir toutes les actions classiques de la vie quotidienne de façon coordonnée et sans communiquer.

De même, le robot Genghis, construit par Rodney Brooks au MIT [Brooks, 1989] possède six pattes. Chaque articulation et contrôlée par un servo-moteur. Il y a douze comportements de balanciers des pattes. Chaque comportement est un agent muni de capteurs et d'effecteurs qui agissent de façon autonome [Franklin & Graesser, 1996]. Chaque agent-comportement cherche à maximiser sa fonction « plaisir » et à minimiser sa fonction « douleur ». Correctement programmé, le robot avance en coordonnant ses six pattes. Un an plus tard, R. Brooks et P. Maes [Maes & Brooks, 1990] ont déprogrammé Genghis et ajouté quelques capteurs. Ils l'ont équipé d'un capteur de touche sous l'abdomen et sur la tête qui envoie un message de douleur lorsqu'il touche le sol. Une petite roulette envoie un message de plaisir lorsque Genghis avance correctement. Le robot a appris à marcher seul par émergence, sans programmation et sans aucune communication entre les pattes.

II.12.2. Le comportement de grégarisme

Il s'agit d'une forme particulièrement de coordination sans communication. C'est typiquement un comportement qu'adoptent, on ne sait pourquoi encore précisément, les bancs de sardines ou les vacanciers sur les plages au mois d'août. Ce comportement consiste simplement à se grouper en nombre. En guise d'explication de ce phénomène, certains chercheurs avancent que, l'union faisant la force, ce comportement est utile pour se défendre contre la prédation. A cela d'autres répondent qu'au contraire, le grégarisme facilite la tâche aux prédateurs. Le grégarisme est cependant utile pour la reproduction puisque facilite le choix d'un partenaire sexuel ; et cela est notamment vérifié le mois d'août à la plage. Mais les seuls en réalité à bénéficier de ce regroupement spontané sont les parasites, les bactéries ou les virus qui peuvent ainsi décimer une population entière très facilement. Les évolutionnistes considérant que « ce qui n'est pas favorable à la vie n'est pas sélectionné et est donc amené à disparaître ». Donc puisque le grégarisme est monnaie courante du règne des bactéries jusqu'à celui de l'homme, cela laisse entendre qu'il est bon de rester groupés. Maintenant appliquons ce raisonnement aux SMA. Le grégarisme doit être considéré comme un sous-produit d'un processus sélectif et donc une conséquence d'autres comportements adaptatifs tel que le choix du meilleur habitat ou du meilleur partenaire sexuel.

Un exemple de comportement courant : si vous hésitez entre deux restaurants inconnus côte à côte, vous allez devoir choisir. Observation initiale du phénomène : les deux restaurants sont vides. Les premiers clients arrivent, hésitent un moment entre ces deux très bons

restaurants et finissent par orienter leur choix en fonction d'un critère subjectif et purement irrationnel ; ce qui est toujours préférable au comportement de l'âne de Buridan. D'autres clients arrivant ensuite vont hésiter un moment et prendre la décision terriblement intelligente d'entrer dans celui contenant le plus de monde en supposant que celui est forcément le meilleur. On observe donc l'émergence d'un processus auto-catalytique.

Les moules ont le même comportement primitif. Elles ont tendance à se fixer sur le même rocher même si d'autres rochers aussi confortables sont présents aux alentours. Il y a donc une règle de production qui consiste à dire « Si cet endroit est bon pour mes congénères, il est bon pour moi ». Il est cependant possible qu'un esprit rebelle et anarchiste décide de s'éloigner du groupe pour diverses raisons louables. Il va donc créer un déséquilibre du système en instaurant un nouveau pôle d'attraction. De nouveaux agents anarchistes « anti-moutons » observant avec quelle audace ce révolutionnaire s'écarte du groupe, vont également venir se grouper auprès de lui, favorisant ainsi le développement de cet attracteur. Peu de temps après les deux groupes arrivent à une mutuelle stabilité créant même deux groupes distincts et bientôt rivaux. Car lorsqu'un individu s'approche de ce nouveau pôle d'attraction, il va en quelque sorte adhérer à un groupe puisque naturellement un groupe est un amas d'individus ayant une particularité en commun. Dans le cas présent, des individus qui ont fait le même choix. Et les sociologues vous le diront, la somme des comportements individuels des entités constituant le groupe n'est pas égal au comportement global du groupe. Il y a donc une notion de comportement individuel et une notion de comportement de groupe. Si par définition un individu a tendance à se rapprocher d'autres individus qui lui ressemblent, ce même individu va aussi avoir tendance à s'éloigner de ceux qui lui sont différents instaurant donc un schisme volontairement ou non. Si on aime ce qui est comme nous, par voie de conséquence on a de l'aversion pour ce qui est différent. Il y a donc une certaine rivalité qui va émerger entre deux groupes différents qui deviennent même concurrents. On constate donc que le résultat de l'association de plusieurs agents crée un groupe. Si d'autres groupes se créent de la même façon sur plusieurs attracteurs, ces groupes vont automatiquement entrer dans une forme particulière concurrence. Ce sont les groupes eux-mêmes qui entrent en conflit et pas les agents constituant ces groupes. Les agents jouent un rôle dans ce groupe. Une fois encore, le comportement de groupe est différent de la somme des comportements individuels.

Il également est probable qu'au sein d'un même groupe se forment d'autres sous-groupes fonctionnant selon la même logique de regroupement. C'est typiquement le comportement d'une société. Une société vit donc de ses coopérations et de ses conflits et trouve dynamiquement un équilibre en fonction de l'interaction de ses sous-ensembles. Un conflit est une forme de désordre mais ce désordre est nécessaire au fonctionnement même de la société. C'est grâce à cet équilibre instable et dynamique que peux exister une société. Sans le désordre, le système devient totalement équilibré et va donc passer de l'état dynamique à l'état statique et perdre ainsi de son autonomie. Il va donc mourir. Car toute forme de vie est la résultante d'interactions spontanées. Il n'y a pas de vie statique. La vie se réduit donc à maintenir cet équilibre fluctuant entre ordre et désordre.

Et l'émergence, d'où vient-elle ? Lorsqu'on construit un système multi-agents, on va imposer quelques règles comportementales simples à un agent et la mise en présence de plusieurs agents va enclencher un processus d'interactions. Mais nous devons encore engrammer ces règles dans l'agent. Ces règles sont le résultat d'une observation d'un comportement sur des individus existants. On observe ainsi le comportement d'une fourmi dans sa fourmilière et on dit « voilà lorsque telle condition est présente avec telle autre, la fourmi se comporte de la sorte ». On en a déduit une règle d'observation. Mais il est intéressant de se demander comment ces règles ont-elles pu apparaître de façon à ce que la

fourmi se comporte de la sorte. L'émergence intervient donc à plusieurs niveaux, mais le véritable phénomène intéressant d'émergence, à mon sens, se situe précisément au niveau le plus bas, sur la formation des règles comportementale elles-mêmes. La seule règle originelle doit être « je dois rester en vie ». Ensuite en fonction des ressources présentent dans son environnement la fourmi érige d'autres règles comportementales qui seront sans doute ensuite codées génétiquement et transmises selon les lois de la génétique à ses descendants.

II.13. Conclusion

Nous avons vu qu'un agent est une entité autonome ayant des facultés de perception, de communication, de décision et d'action. Un des aspects importants définissant le concept d'agent repose notamment sur son autonomie décisionnelle. Cette autonomie dépend du type d'agent et du modèle interne sur lequel repose le fonctionnement des agents, c'est-à-dire du raisonnement et du comportement. Elle peut être une autonomie pour la réalisation des buts ou une autonomie de décision vis-à-vis des motivations se préoccupant de permettre aux agents de choisir par eux-mêmes quand et comment interagir avec les autres.

Un agent est plongé dans un environnement et interagit avec celui-ci. Nous ne parlons pas d'agent sans environnement ni sans une autre collection d'agents avec laquelle l'agent susdit interagit. Les interactions entre les agents et la résolution distribuée qu'elles mettent en place induisent alors une contradiction avec ce principe d'autonomie de décision propre au concept d'agent. Les agents peuvent avoir conscience de cette résolution collective (agents cognitifs) ou non (agents réactifs), mais de toute façon, ils utiliseront ou subiront à un moment ou à un autre des contraintes issues de leurs interactions avec les autres agents du système.

Toute la difficulté de concevoir des systèmes multi-agents consiste alors à gérer cette contradiction entre le principe d'autonomie des agents et la résolution collective par le système qu'ils composent. Tous les agents doivent s'intégrer au même système pour former un tout cohérent résolvant le problème à traiter, même si les formalismes qui modélisent les différents agents du système sont hétérogènes. Cette nécessité d'intégration et d'interaction avec les autres fait intervenir des mécanismes et des notions permettant la coordination de la résolution distribuée du problème pour obtenir un comportement global cohérent et efficace du système. Les notions d'interactions et d'organisation permettent d'appréhender cette nécessité. Ce qui va former le tout est justement cet ordre à l'équilibre et cette contradiction qui n'est pas sans rappeler le concept du Yin et du Yang chinois [Granet, 1988].

L'équilibre dynamique est la clé de voûte de l'auto-organisation d'un système. Un système va rester en vie s'il maintient la dynamique de son organisation et non pas uniquement sa structure figée. Il peut en effet connaître des reconformations structurelles qui ne sont pas nuisibles à sa cohérence interne. La perturbation devient au contraire un élément clé du système. Ce système présente des propriétés de souplesse et de robustesse qui sont assurées par un équilibre dynamique entre la spécialisation et la redondance des agents le constituant.

Le système n'est pas optimum au niveau de son fonctionnement car la présence d'agents proposant les mêmes services est consommatrice de ressources et donc de temps pour l'exécution d'une opération informatique. En revanche la redondance des agents sera très bénéfique pour assurer son intégrité en cas d'agression par un système étranger. Un bon système multi-agents est justement non optimisé s'il veut conserver sa robustesse. Cela est tout à fait contraire à ce qu'on attend d'un programme informatique actuel dans lequel on cherche à réutiliser et à optimiser ses modules fonctionnels.

Chapitre III. Méthodologies et modèles SMA

« La machine d'arithmétique fait des effets qui approchent plus de la pensée que tout ce que font les animaux mais elle ne fait rien qui puisse faire dire qu'elle a de la volonté, comme les animaux » - Blaise Pascal.

Résumé : Les méthodologies d'analyse classiques échoppent à prendre en considération les caractéristiques particulières des SMA. Beaucoup de recherches ont donc porté sur de nouvelles méthodes plus adaptées à notre domaine d'étude. Ce chapitre présente, avec un état de l'art des méthodologies d'analyse des SMA et des Systèmes Multi-Agents adaptatifs, celles qui semblent les plus appropriées pour notre modèle d'organisation YAMAM.

III.1. Avant-Propos

Les logiciels informatiques sont conçus aujourd'hui pour des applications de plus en plus complexes, souvent distribuées, et ayant une très forte dynamique endogène[24] ou exogène. Il devient donc de plus en plus difficile de contrôler l'activité de ces logiciels situés dans des environnements de plus en plus dynamiques. Pour faire face à ces difficultés, une solution consiste à accorder davantage d'autonomie aux logiciels afin qu'ils s'adaptent au mieux aux imprévus et offrent plus de robustesse. L'utilisation des techniques SMA présente donc un intérêt particulier pour faire face à cette complexité grandissante des logiciels complexes.

Il existe bien entendu des méthodes pour l'analyse objet, aujourd'hui utilisées dans l'industrie mais le passage d'une analyse orientée objet à une analyse orientée agent soulève beaucoup de problèmes liés en grand partie à la dynamique inhérente aux SMA. Par définition, un agent est autonome dans le sens où il est capable de décider seul de l'action à réaliser en fonction de ses perceptions et de ses connaissances (croyances, compétences…). Cette autonomie lui permet de prendre en compte des événements imprévus. Contrairement à un agent, un objet ne réagit qu'à un appel de méthode et toutes les situations auxquelles il va être confronté doivent être prises en compte par le concepteur. De plus, les agents ont une composante sociale très importante qui influence leur comportement, chose que ne possèdent pas les objets. Les agents et les objets se différencient donc essentiellement au niveau du comportement et des interactions. Ces différences conceptuelles orientent fortement les travaux sur les méthodologies de conception.

D'une manière générale, les spécialistes du domaine s'accordent à dire qu'il existe à ce jour peu de méthodes génériques d'analyse et de conception de SMA. "*At this time, there are two major impediments to the widespread adoption of agent technology: (i) the lack of systematic methodology enabling designers to clearly specify and structure their application as multi-agent systems; and (ii) the lack of widely available industrial-strength multi-agent system toolkits.*" [Jennings & al., 1998].

[24] Endogène : Se dit d'un élément qui se forme à l'intérieur de l'organe qui l'engendre. Ant. Exogène.

De plus, les systèmes multi-agents « classiques » prennent en compte la complexité des interactions mais ne traitent pas directement la dynamique du système [Casteran & al. 2000]. C'est la raison pour laquelle nous nous intéressons aujourd'hui aux systèmes multi-agents adaptatifs qui apportent de réelles solutions techniques adéquates pour maîtriser cette complexité logicielle. Ces systèmes adaptatifs présentent des propriétés d'émergence [Gleizes & al., 1999], [Cardon & Guessoum, 2000], [Parunak & Brueckner, 2001]. Les méthodologies de conception de systèmes multi-agents adaptatifs sont donc encore différentes et encore plus complexes que les méthodologies applicables aux SMA classiques.

Nous décrivons dans ce chapitre, certainement de façon non exhaustive, les méthodes les plus connues aujourd'hui qui permettent d'approcher une modélisation multi-agents puis nous insisterons sur celles qui permettent l'analyse des SMA dynamiques applicables notamment à notre problématique.

III.2. Intérêt d'une méthodologie

Une méthodologie est une "recette" qui permet à un ingénieur-concepteur de trouver une solution à un ensemble de problèmes spécifiques. Elle doit être suffisamment précise pour permettre à quelqu'un de non spécialiste d'appliquer cette « recette » à un problème donné. Une méthodologie consiste en :

- une définition de l'espace du problème auquel la méthodologie est applicable
- un ensemble de modèles qui représente différents aspects du domaine du problème ou une solution aux différentes étapes
- un ensemble de méthodes qui transforment les instances d'un modèle en un autre modèle
- un ensemble de directives procédurales qui définissent un ordre pour l'application systématique des étapes de la méthodologie.

L'application d'une méthodologie commence avec l'énoncé du problème et termine avec une solution à ce problème. Les méthodes et les directives guident le concepteur pas à pas vers la solution. Nous allons présenter dans ce chapitre quelques méthodologies qui visent à guider le concepteur d'un SMA vers une solution. Ces méthodologies sont actuellement des propositions et aucune ne s'impose encore comme un standard tel que MERISE [Tardieu & al., 2000] pour les bases de données par exemple.

Des méthodologies existent dans de nombreux domaines qui ne touchent pas directement aux SMA. On trouve par exemple des méthodes comme ATOME de Laasri ou Hearsay II, pour la conception des BlackBoards. DVMT (Distributed Vehicle Monitoring Testbed) fut créée par Lesser en 1984 pour le contrôle hiérarchique décentralisé dans le cadre de la résolution distribuée de problèmes [Lesser & Corkill, 1983]. RETSINA de Sycara est une autre infrastructure agent. Il existe aussi ContractNet de Smith, en 1982, qui concernait le contrôle hiérarchique décentralisé.

Les concepteurs de SMA recherchent également des méthodologies applicables à leur domaine d'activité. Nous allons en présenter quelques unes.

III.3. Les méthodologies d'analyse des SMA

Il existe à ce jour un certain nombre d'outils pour passer d'une programmation orientée objet à une programmation orientée agent. La programmation orientée agent

nécessite cependant en amont une analyse et une modélisation orientées agent qui diffèrent assez radicalement d'une méthode d'analyse objet ; les agents et les objets se différentiant essentiellement au niveau du comportement et des interactions. Ces différences conceptuelles orientent fortement les travaux sur les méthodologies de conception.

Une analyse est une phase préalable à la conception, mettant en œuvre un processus de recherche de sens dans l'étude du système. On cherche à comprendre les tenants et les aboutissants du système étudié. L'analyse d'un SMA consiste à décrire son organisation.

Le comportement d'un système multi-agents est le résultat des interactions et du comportement des agents qui le constituent. Pour le concepteur, la tâche est simplifiée par le fait qu'il peut se focaliser uniquement sur le développement les agents et des mécanismes d'interaction entre ces derniers. L'approche de [Durand, 1996] est intéressante et propose de modéliser les schémas d'organisation en présence, leurs interactions et leur instanciation dans des institutions existantes. Il propose de décrire les SMA grâce à plusieurs schémas mais il ne s'agit pas à proprement parler d'une réelle méthodologie d'analyse. On peut néanmoins distinguer dans cette approche trois niveaux de description qui sont les schémas d'organisation, les types d'agents et les agents concrets.

Depuis environ cinq ans, de nombreuses méthodologies de conception de systèmes multi-agents ont été développées ou sont en cours de développement. Aucune d'entre elles n'est pour le moment universellement appliquée dans la mesure où les SMA demeurent à l'état de recherche et que certaines de leurs propriétés sont encore non maîtrisées.

Ces méthodologies constituent soit une extension des méthodologies orientées objet, comme par exemple Agent Modeling for System of BDI agents [Kinny & al., 1996] ou MESSAGE [Message, 2001], soit une extension des méthodologies à base de connaissances, comme CoMoMAS de Glaser [Glaser, 1997]. Cependant ces différentes méthodologies ne fournissent pas de techniques adéquates pour modéliser les caractéristiques spécifiques aux agents telles que leur état mental et leur comportement social dans un SMA. D'autres méthodologies comme GAIA [Wooldridge & al., 2000] ont vu le jour, mais restent néanmoins toujours incomplètes. Elles ne prennent véritablement pas en compte, par exemple, la vérification des SMA ou l'aspect adaptatif des SMA.

Les différentes méthodologies s'expliquent aussi par le fait qu'il y a différentes façons de considérer un SMA. Ocello et Demazeau dans [Ocello & Demazeau, 1997] proposent une décomposition et une encapsulation fonctionnelle pour la modélisation d'un SMA. Dans l'analyse fonctionnelle descendante d'une application, il est possible d'identifier les briques (sous-systèmes) qui composent le système. Un agent est défini au niveau *n* peut être vu comme un SMA au niveau *n-1*, lorsqu'il est décrit à un niveau de granularité plus bas. Au contraire, dans une analyse fonctionnelle ascendante, il faut trouver les fonctions d'un niveau *n-1* qui peuvent être encapsulées dans une fonction d'un niveau *n*. Brazier [Brazier & al., 1995] présente un mécanisme de décomposition du système en modules primitifs ou composés. Baeijs dans [Baeijs, 1998] propose une grammaire pour la définition récursive d'organisation, laquelle permet de considérer différents niveaux de décomposition : l'organisation, les groupes (liens) et les agents (membres). Une organisation à un niveau n peut être considérée comme un groupe à un niveau n-1. Par contre, un groupe à un niveau n peut être considéré comme un membre à un niveau n+1. [VanAeken & Demazeau, 1998] ont développé un modèle d'organisation dynamique - les Systèmes Multi-Agents Minimaux (SMAM) - dans lequel les agents sont organisés en couples d'agents atomiques. Le SMAM est défini récursivement comme étant soit un agent atomique, soit un couple composé de deux SMAM. Ces travaux démontrent la possibilité d'organiser le système en groupes de manière dynamique.

Nous avons observé par ailleurs que certaines méthodologies de SMA font partiellement référence à d'autres méthodologies connues dans d'autres domaine comme les Systèmes à Base de Connaissance pour lesquels il existe KADS, KOD, MKSM, DOLMEN, REX, DIADEME, ... De même pour l'ingénierie logicielle, il existe MERISE, SADT, SA, SA-RT. Les spécialistes de la programmation orientée objet utilisent les méthodes Booch et OMT alors que les professionnels des systèmes de Mémoire d'Entreprise utilisent SAGACE, ACTIONIS, HLL ou GRAI… Les méthodologies agents sont donc encore à l'état de recherche et sont souvent comparées dans des groupes de travail comme COLLINE [Colline, 2002] dirigé par Jean-Pierre Müller.

D'une manière très générale, deux démarches sont possibles pour la méthodologie : La démarche *Ascendante qui va v*ers les Design Patterns [Buschmann & al., 1996]; c'est la construction du point de vue de l'utilisateur ; et la démarche *Descendante qui traite des* concepts et des méthodes. La plupart de ces méthodologies se basent sur deux notions pour décrire ces systèmes : les rôles que l'agent peut adopter (ce qui peut mener, dans certains cas, à une bijection entre l'agent et le rôle) et parfois les groupes (ou organisations) dans lesquels il joue ces rôles. C'est le cas, par exemple, dans Aalaadin qui propose un modèle d'organisation sur un triplet Agent-Groupe-Rôle [Gutknecht & Ferber, 1998b], qui permet néanmoins à un agent d'appartenir à plusieurs groupes et donc de pouvoir adopter plusieurs rôles au sein de ces différents groupes. On retrouve également la notion de groupe dans le modèle MOISE [Hannoun & al., 1999] qui s'articule autour de trois niveaux : individu, groupe et société. L'organisation des agents y est vue comme un système de règles qui contraint le comportement des agents.

III.4. MAS et AMAS

Les SMA classiques sont différents des SMA adaptatifs. Ces derniers présentent une plus grande dynamique car les agents ont un comportement dicté par la coopération. Un AMAS (ou Adaptative Multi-Agent System) a donc une propriété de reconformation en continue du système. Cela a nécessairement une incidence sur la phase d'analyse et de conception qui diffère donc d'une analyse classique de SMA. Par exemple, en programmation objet, la communication entre objets consiste uniquement à invoquer une méthode. Les interactions entre les objets sont définies de manière rigide et n'évoluent pas dans le temps. Au contraire, dans les SMA, l'interaction entre les agents est beaucoup plus développée et complexe à mettre en œuvre (fondée sur des protocoles) et évolue dans le temps. De plus, dans une conception d'un système adaptatif, l'organisation entre les agents est incomplètement définie et se met en place au cours du fonctionnement du système. Les relations (telles que : *est-un* et *est-composé-de*) sont insuffisantes pour modéliser les relations organisationnelles d'un système complexe [Jennings & Wooldridge, 2000]. Cela nécessite donc des méthodes d'analyse et de conception encore différentes des méthodes applicables aux SMA standards.

De manière générale, les méthodologies orientées agent présentées dans la littérature, ne sont pas adaptées à la conception de systèmes complexes adaptatifs car la plupart des méthodologies de conception de SMA partent des agents et d'une description de leur organisation. Le concepteur n'a pas toujours la connaissance de l'organisation finale qui permet d'aboutir à la réalisation de la bonne fonction ou activité.

Il existe un second problème commun à ces méthodologies qui dépend du point précédent : le fait que le système soit décomposé en organisations et en rôles pour l'agent oblige le concepteur à avoir une idée précise sur l'identification de ses agents. Ces méthodologies n'expliquent pas toujours comment décomposer un système afin d'identifier

des agents, mais à quoi le système devra ressembler une fois que les différentes structures seront mises en place. Le fait que dans certaines applications, le concepteur ne puisse pas donner de rôles aux agents (par ignorance ou par trop grande complexité) rend ces méthodologies inadaptées à un grand nombre de cas, notamment celui des systèmes adaptatifs complexes.

Enfin, les agents adaptatifs représentent un cas où leur création ne doit pas s'articuler autour des fonctionnalités du système, mais par rapport à leurs capacités comportementales et organisationnelles. Ce dernier point met d'ailleurs en avant l'idée que les agents adaptatifs sont cohérents pour le développement de systèmes à fonctionnalité émergente. La conception de systèmes adaptatifs pose un certain nombre de problèmes. Il est par exemple délicat d'identifier les agents dans un système dont il est impossible de connaître une organisation à un moment donné. Faut-il ensuite décomposer un agent en un système multi-agents ou l'agent peut-il être considéré comme atomique (en fonction de sa complexité) ? Un autre problème est l'identification des connaissances nécessaires à chaque agent afin de réaliser cette adaptation. Nous n'avons pour le moment pas de réponses à ces questions, en tout cas pas de façon générique.

La plupart des descriptions de travaux portant sur les méthodologies proposent, dans la phase d'analyse, de décrire les agents impliqués dans la description du problème ; seuls quelques travaux tels que DESIRE [Brazier & al., 1997] décrivent le lien entre les scénarios d'expression des besoins et les agents. Le projet RNTL « ADELFE » [Bernon & al., 2002] (Atelier pour le DEveloppement de Logiciels à Fonctionnalité Emergente) a pour but de concevoir un atelier de développement de systèmes adaptatifs, en indiquant les critères essentiels pris en considération dans trois phases : adéquation des AMAS, analyse et conception. L'approche VOYELLES de [Tavares & Demazeau, 2002] est plus adaptée aux SMA adaptatifs. Elle considère les Agents, les Environnements, les Interactions et les Organisations (AEIO) de même importance paradigmatique. Cette méthode est liée au toolkit MASK [Occello & al., 2002].

III.5. Phases de construction des SMA

En ce qui concerne les phases de construction des systèmes multi-agents, de plus en plus de méthodologies préconisent le découpage en quatre grandes phases : l'analyse, la conception, le développement et le déploiement. En effet, les systèmes multi-agents sont des systèmes logiciels complexes, nécessitant des méthodes d'ingénierie adéquates. Traditionnellement, on ne distingue en génie logiciel que trois phases de construction de systèmes : l'analyse, la conception et le développement. C'est cependant de moins en moins vrai aujourd'hui puisque les logiciels deviennent de plus en plus complexes. Dans le cadre des SMA qui sont en tout état de cause des logiciels complexes et distribués par nature, cette quatrième phase semble incontournable. Nous proposons d'utiliser par la suite les définitions suivantes des quatre étapes :

- ***Analyse*** : la détermination, séparation et description du type de problème et du domaine environnant. En pratique, il s'agit d'identifier le domaine d'application et le cœur du problème à résoudre.
- ***Conception*** : la définition de l'architecture de la solution au problème, de manière déclarative. En pratique, il s'agit de spécifier un principe de solution, par exemple en utilisant UML.

- ***Développement*** : la construction d'une solution fonctionnelle au problème. En pratique, il s'agit de coder la solution dans un langage de programmation, qui peut être général (par exemple Java) ou plus orienté multi-agents (par exemple AOP).
- ***Déploiement*** : l'application de la solution au problème réel dans le domaine. En pratique, il s'agit de lancer le logiciel sur un réseau d'ordinateurs, et de maintenir (modifier ou remplacer des agents) ou d'étendre ses fonctionnalités (migration d'agents).

Ces différentes étapes seront souvent reprises dans les différentes méthodologies existantes aujourd'hui.

III.6. Comparaisons des méthodes

En ce qui concerne les méthodes applicables aux SMA, nous nous appuierons partiellement sur les critiques de [Iglesias, 1998], de [Wooldridge & Ciancarini, 1999] et [Wooldridge & al., 2001], de [Bernon & al., 2001] et de [Sabas & al., 2001]. Ils proposent de comparer les 9 méthodologies les plus connues (GAIA, MaSE, MMTS, HLIM, CoMoMAS, MASB, MAS-CommonKADS, AOMEM et Cassiopée) sous les dimensions (méthodologie, représentation, organisation, technologie, coopération, agent). Chaque dimension est définie avec un certain nombre de critères. Cette comparaison est une première étape vers une tentative d'unification des méthodologies SMA. Les méthodes GAIA, MaSE, HLIM, MMTS et AOMEN utilisent comme toile de fond des techniques orientées-objet. L'avantage de ces méthodes est que l'expérience et le succès liés à l'utilisation des méthodes orientées-objet peuvent faciliter l'intégration de la technologie agent. Elles appréhendent mieux que les méthodes CoMoMAS et MAS-CommonKADS la phase d'analyse. Les méthodes CoMoMAS et MAS-CommonKADS constituent une extension de la méthode à base de connaissances CommonKADS. Ces méthodes ainsi que la méthode MMTS fournissent des modèles (cependant complexes) qui prennent mieux en compte l'état interne des agents que ceux des méthodes GAIA, MaSE, HLIM, et AOMEN. La méthode Cassiopée a été développée pour un contexte particulier, celui des robots footballeurs. Aucune des méthodes n'a fourni des modèles qui recouvrent totalement tous les concepts des SMA. Le support logiciel (UCMs) utilisé par HLIM pour la représentation est très intéressant pour la compréhension, l'interprétation et la communication entre les différents acteurs du développement des SMA. Le formalisme (AgML, AgDL) utilisé par MaSE rend générique ses composantes et facilite la vérification. Les dimensions coopération, organisation et technologie restent encore, à notre avis, à approfondir dans ces différentes méthodologies.

Il existe aussi des méthodes très spécifiques à un domaine particulier comme CAROSSE (Conception, Analyse et Réalisation de l'Organisation de Sous-Systèmes Electroniques Embarqués) de Rémy FOISEL (LIP6) et d'Olivier CAYROL (PSA).

III.7. Méthodes retenues

Il existe un certain nombre de méthodologies offrant un intérêt certain pour l'étude des SMA d'un point de vue organisationnel. Une étude des méthodologies d'analyse plus précise peut être trouvée dans le rapport interne [Savall, 2002]. Parmi celles-ci, certaines présentent un intérêt pour notre problématique de recherche mais ne sont pas directement applicables dans notre contexte.

III.7.1. GAIA

La méthodologie GAIA de Wooldridge [Wooldridge & al., 2000] est fondée sur la vue d'un système multi-agents comme une organisation de rôles interagissant en relation les uns avec les autres. Cette méthodologie est assez bien adaptée pour concevoir des SMA mais moins pour les SMA Adaptatifs. Néanmoins, afin d'optimiser le processus de stabilisation, il peut être intéressant d'appliquer cette méthode pour démarrer l'exécution du SMA sur une organisation déjà cohérente. Dans GAIA, les caractéristiques du domaine d'application sont les suivantes:

- Les agents sont des systèmes informatique à faible granularité, chacun faisant usage de ressources système significatives (comme par exemple un processus unix).
- Le but est d'obtenir un système qui maximise une mesure de qualité globale, mais qui peut être sous-optimale du point de vue des composants du système.
- Les agents sont hétérogènes, c'est-à-dire que différents agents peuvent être implémentés en utilisant différents langages de programmation, architectures et techniques.
- La structure organisationnelle du système est statique : les relations entre agents ne changent pas pendant l'exécution
- Les capacités des agents et les services qu'ils fournissent sont statiques : ils ne changent pas pendant l'exécution
- Le système dans son ensemble contient un nombre comparativement faible de types d'agents différents (moins de 100).

GAIA s'occupe du niveau macro et du niveau micro de la conception. Cette méthode propose au développeur de penser ses systèmes à base d'agents comme un process de conception d'organisations. L'analyse et la conception sont vues comme un process de développement de modèles de plus en plus détaillé. Les concepts principaux de GAIA sont divisés en deux parties : la partie abstraite et la partie concrète. L'idée de la méthode est de passer des concepts abstraits à des concepts de plus en plus concrets. On est ainsi à chaque étape de plus en plus proche de l'implémentation. Les entités sont celles utilisées pendant la phase d'analyse pour conceptualiser le système. Les entités concrètes sont utilisées dans le processus de conception.

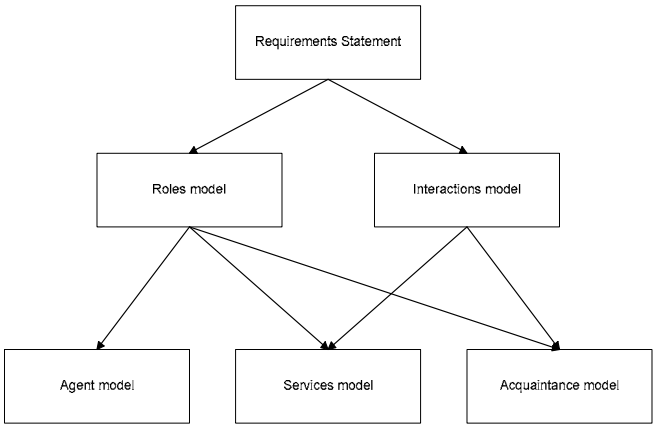


Figure 2 - Relations entre les modèles de GAIA

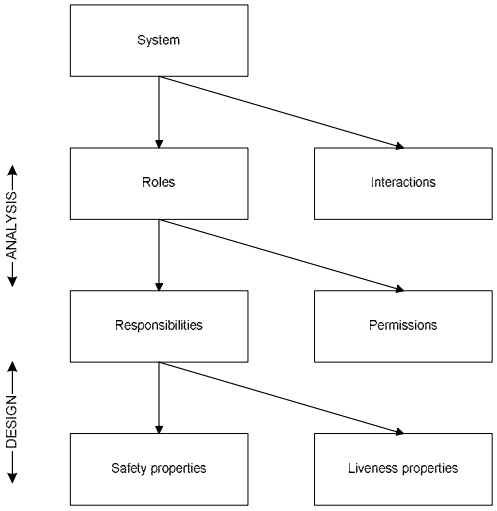


Figure 3 - Concepts d'analyse dans GAIA

L'objectif de la phase d'analyse est de développer une compréhension du système et de sa structure sans référence à aucun détail d'implémentation. L'organisation du système est vue comme un ensemble de rôles en relations les uns les autres qui forment des schémas d'interactions. L'entité la plus abstraite dans la hiérarchie est le système. Un rôle est défini par quatre attributs : la responsabilité, la permission, l'activité et le protocole.

La responsabilité détermine la fonctionnalité. C'est certainement le plus important associé au rôle. Les responsabilités sont divisées en deux types : les propriétés de « vivacité » (liveness) et les propriétés de sûreté (safety) [Pnueli, 1986].

> *Toute propriété est l'intersection d'une propriété de sûreté et d'une propriété vivacité. Une propriété de vivacité ou "liveness" exprime intuitivement "qu'une bonne chose arrivera nécessairement". Par exemple le programme réagira normalement à une stimulation ou réagira normalement après avoir atteint une certaine situation. La terminaison, l'absence de famine ou la garantie de service sont des propriétés de vivacité. Les propriétés de sûreté ou "safety" expriment pour leur part "que de mauvaises choses ne se produiront pas" durant l'exécution du système. Par exemple, après avoir atteint telle situation, aucune situation incorrecte ne sera atteinte. L'exclusion mutuelle ou l'absence de blocage (deadlock) sont typiquement des propriétés de sûreté – On trouve ces notions dans le cadre de la théorie des graphes et des automates (systèmes réactifs synchrones par ex) - NDLR.*

Afin de réaliser les responsabilités, un rôle a un ensemble de ***permissions*** qui sont, ni plus ni moins, des « droits » associés à un rôle. Les permissions d'un rôle identifient les ressources disponibles pour ce rôle pour réaliser les responsabilités. Les ***activités*** d'un rôle sont des calculs associés au rôle qui peuvent être fait par un agent sans interaction avec d'autres agents. Ce sont en quelque sorte des actions privées. Un rôle est aussi identifié par un nombre de ***protocoles*** qui définissent de quelle façon ce rôle va interagir avec d'autres rôles. Le fameux « Contrat Net Protocol » est associé aux rôles de « manager » et de « contractor ».

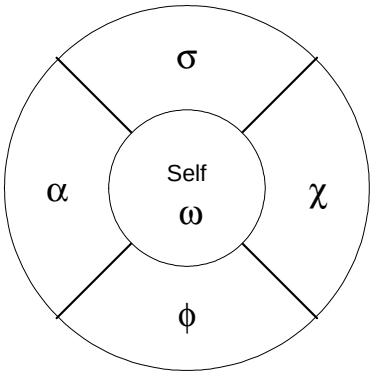
GAIA propose donc deux modèles dans la phase d'analyse : le modèle des rôles, qui identifie les rôles clés du système, et le modèle des interaction qui est vu comme un ensemble de définitions de protocoles. Le processus de conception (Design) implique le développement de 3 modèles :

- modèle d'agents (identifie les types et les nombres d'instances d'agents)
- modèle de services (identifie les services requis pour réaliser le rôle d'un agent)
- modèle d'accointance (documente les lignes de communications entre les différents agents)

La méthodologie GAIA est intéressante mais présente quelques limitations, notamment dans son aspect statique. Nous nous intéressons au contraire aux systèmes multi-agents adaptifs, c'est-à-dire que nous cherchons un modèle dans lequel les relations entre les agents puissent changer dans le temps.

III.7.2. AALAADIN

Cette méthode est particulière et propre à Jacques Ferber du LIRMM, son concepteur. Globalement, l'analyse du SMA se compose de cinq dimensions :



- La dimension physique traite de l'implémentation, de l'architecture de l'organisation et de ses ressources personnelles.
- La dimension sociale concerne la place et donc le rôle d'une organisation dans une organisation de

plus haut niveau. Cette dimension s'occupe aussi de l'aspect social des activités d'une organisation, c'est à dire de ses capacités à répondre à des besoins ou fonctions générales nécessaires au bon fonctionnement de l'organisation de plus haut niveau dans laquelle elle se trouve.

- La dimension relationnelle s'occupe des interactions, des échanges qu'une organisation peut avoir avec d'autres organisations de même niveau, des ses communications et de leur coordination.
- La dimension environnementale est liée aux capacités d'une organisation à agir, percevoir et raisonner sur le monde environnant. C'est à ce niveau que sont définies ses capacités d'interactions avec le monde, ses compétences proprement instrumentales.
- La dimension personnelle représente tout ce qui a trait à l'organisation en tant que telle et en particulier à ses aspects « méta » : la représentation de soi, l'acquisition des connaissances, la préservation de sa structure et l'activité proprement autopoïétique d'une organisation.

La méthode Aalaadin a donné lieu à un très bon toolkit JAVA, développé par Olivier Gutknecht, qui porte le nom de Madkit. Il sera étudié dans un prochain chapitre. Aalaadin [Ferber & Gutknecht, 1998a] est un projet qui porte sur l'analyse, la conception, la formalisation et la réalisation de systèmes multi-agents dans une perspective organisationnelle. Le modèle conceptuel sous-jacent (AGR) est fondé sur les notions d'agents, de groupes et de rôles, un SMA étant vu comme un ensemble de groupes d'agents interconnectés par l'intermédiaire d'agents appartenant à plusieurs groupes. Ce projet s'exprime selon quatre niveaux :

- Le niveau méthodologique : Ferber a développé une méthodologie à partir des concepts AGR qui s'exprime de manière complémentaire aux méthodologies classiques (UML par exemple) ou centrées sur l'agent. Il s'agit d'un "framework organisationnel", un cadre méthodologique volontairement partiel et complété par des modèles externes.
- Le niveau formel : le problème de la dynamique des groupes dans AGR a été entièrement formalisé en termes de pi-calcul [Milner, 1999], un calcul algébrique très utilisé pour définir la sémantique des processus concurrents. Un cadre formel a ainsi pu être donné aux notions d'agents, de groupes et de rôles.
- Le niveau architectural : un environnement de développement, Madkit, écrit en Java, implémente les concepts AGR et permet d'écrire des applications directement avec ce modèle.
- Le niveau applicatif : le modèle AGR et Madkit sont utilisés dans de nombreux domaines d'application : biologie, simulation multi-agents, acquisition de connaissances. Plus d'une vingtaine d'équipes en France travaillent avec cet outil.

Aalaadin situe l'analyse de systèmes multi-agents à un niveau descriptif et à un niveau méthodologique que nous allons décrire ci-dessous.

III.7.2.1. Modèle descriptif

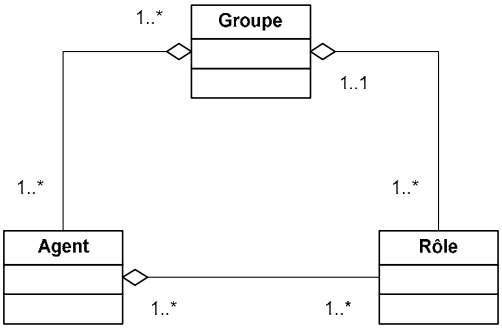
Le niveau descriptif correspond aux concepts primaires d'agent, de groupe et de rôle. C'est à ce niveau que se décrit une organisation multi-agents réelle. Les trois notions de base d'Aalaadin sont :

L'agent : aucune contrainte ou pré-requis sur l'architecture interne de l'agent n'est posée et aucun formalisme ou modèle particulier n'est supposé pour en décrire le

comportement. L'agent est simplement décrit comme une entité autonome communicante qui joue des rôles au sein de différents groupes. Le caractère très générique de cette définition est volontaire : le constructeur de système appliquera à cette notion d'agent l'architecture d'action ou de communication idoine.

Le groupe : le groupe est défini comme la notion primitive de regroupement d'agents. Chaque agent peut être membre d'un ou plusieurs groupes. Dans sa forme la plus simple, un groupe peut être vu comme un moyen d'identifier par regroupement un ensemble d'agents. D'une manière plus évoluée, le groupe peut être vu comme un SMA usuel. Un point majeur de cette définition est que les différents groupes peuvent se recouper librement. Un groupe peut être fondé par n'importe quel agent, par contre, l'admission à un groupe existant résulte d'une demande formulée par l'agent candidat.

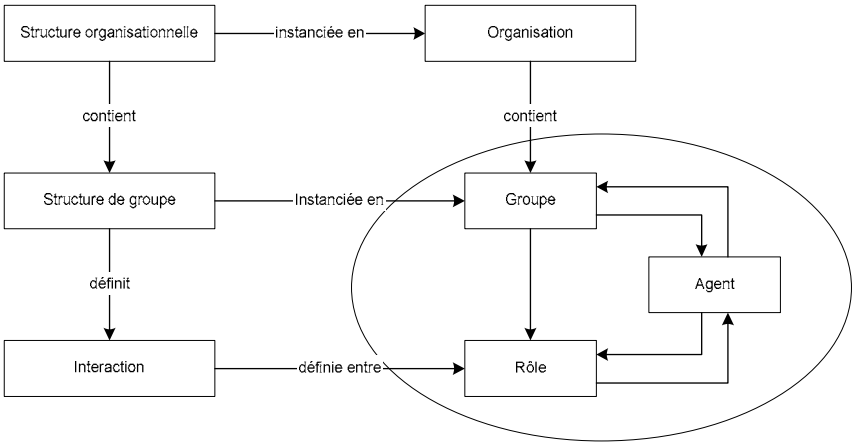
Le rôle : le rôle est une représentation abstraite d'une fonction, d'un service ou d'une identification d'un agent au sein d'un groupe particulier. Chaque agent peut avoir plusieurs rôles, un même rôle peut être tenu par plusieurs agents, et les rôles sont locaux aux groupes. De même qu'avec l'admission dans un groupe, la tenue d'un rôle doit être demandée par l'agent et n'est pas forcément accordée.



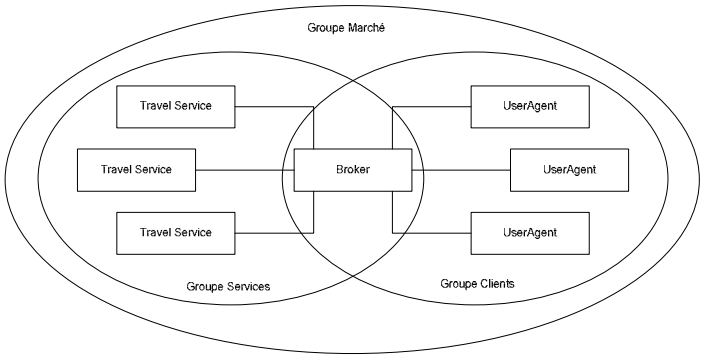
La maîtrise de l'hétérogénéité des situations d'interaction est rendue possible par le fait qu'un agent peut avoir plusieurs rôles distincts au sein de plusieurs groupes : toute communication entre agents ne peut avoir lieu que si ces agents sont membres d'un même groupe. Le modèle décrit l'organisation du SMA et définit de plus la notion de fonctions d'admissions, qui spécifie et régit la dynamique des rôles tenus les agents (c'est à dire la demande d'un rôle, son maintien, sa révocation).

III.7.2.2. Niveau méthodologique

Le niveau méthodologique définit l'ensemble des rôles possibles, spécifie les interactions et décrit les structures abstraites de groupe et d'organisation. Quelques termes supplémentaires sont définis ici qui, à la différence du trio " agent - groupe - rôle ", ne sont pas représentés directement dans une organisation multi-agents. Ils sont introduits en tant que guide de conception, le but étant de pouvoir spécifier une structure organisationnelle abstraite à partir de laquelle un système multi-agents puisse être développé et exprimé dans les concepts de base. Ce niveau correspond à la définition des rôles possibles, à la spécification des interactions et à la description des structures abstraites que sont les groupes et l'organisation. La figure ci-dessus présente une vision d'ensemble du modèle Aalaadin.



La structure de groupe est une description abstraite d'un groupe. Elle identifie la totalité des rôles et des interactions qui peuvent survenir au sein d'un groupe. Notons que la structure de groupe peut se trouver instanciée d'une manière partielle dans le SMA final : il n'est pas requis que l'ensemble des rôles définis dans la structure de groupe se retrouve simultanément dans le groupe réel, ceci dépendant de la dynamique interne du groupe réel.



La structure organisationnelle est définie comme un ensemble de structures de groupe décrivant un modèle d'organisation multi-agents. Cette structure organisationnelle correspond à la spécification complète du problème initial. Les différentes structures de groupe mises en œuvre - spécifiables indépendamment - permettent une gestion viable de l'hétérogénéité des langages de communications et des modèles de domaine et d'agents, le tout au sein d'un même système.

Tout comme la correspondance entre structures de groupes et groupes réels, notons que l'organisation réelle finale n'est qu'une manifestation possible de la structure organisationnelle. Elle peut ne pas inclure tous les groupes définis dans la structure organisationnelle abstraite. La figure ci-dessus montre un exemple d'organisation avec un broker appartenant à plusieurs groupes. Un broker vendeur appartient au groupe des services et un broker client appartient au groupe des clients. Une description plus détaillée peut être lue sur [Gutknecht & Ferber, 1998b].

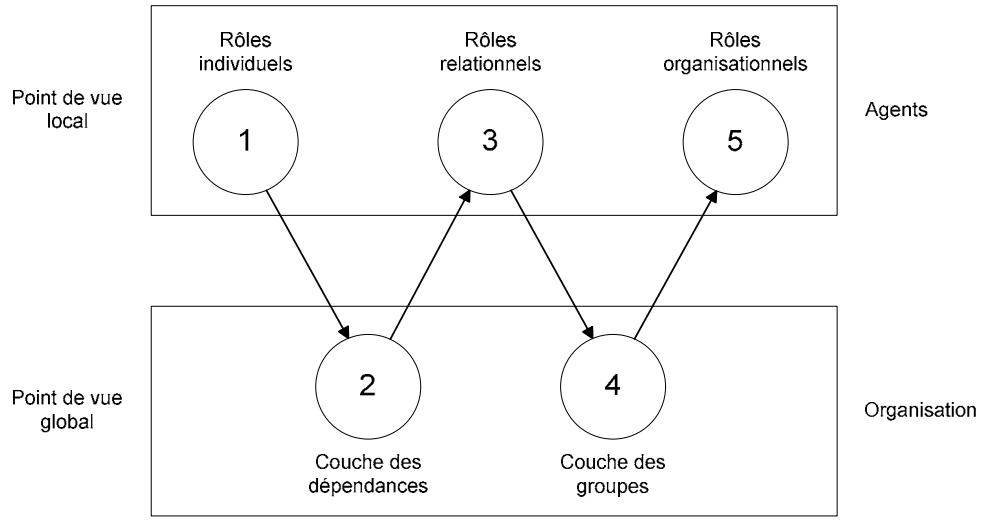
Cette méthodologie est très complète et a retenu toute notre attention mais contraint le concepteur de SMA à « penser » en terme d'agent, de groupe et de rôle, ce qui est assez contraignant.

III.7.3. Cassiopée

La notion d'organisation et d'interaction est primordiale dans les systèmes multi-agents adaptatifs. Le principe fondateur de la méthodologie Cassiopée est d'articuler l'analyse

et la conception d'un système multi-agents précisément autour de la notion d'organisation [Collinot & al., 1996]. Cassiopée utilise plusieurs concepts tels que le rôle, l'agent, la dépendance et le groupe. Un agent est défini comme un ensemble de trois niveaux différents de rôles (individuels, relationnels et organisationnels). La méthode propose ensuite un bouclage et une stabilisation sur la définition de cinq couches :

- La couche des rôles individuels qui contient la définition des rôles des agents et qui définit donc leurs types. Cela consiste à identifier les comportements élémentaires requis pour la réalisation des tâches considérées comme collectives.
- La couche des dépendances qui définit les dépendances entre les rôles (fonctionnelles, basées sur les ressources et sur les buts).
- La couche des rôles relationnels qui définit les influences exercées par les agents entre eux. Cela consiste à analyser la structure de l'organisation basée sur les dépendances entre les comportements élémentaires de la couche précédente. Un graphe d'influence est alors généré.
- La couche des groupes qui définit les groupes qui peuvent apparaître en respect des influences précédentes
- La couche des rôles organisationnels qui concerne la dynamique des groupes. Cela consiste à spécifier les comportements qui permettront aux agents de gérer la formation, le maintien ou la dissolution des groupes.

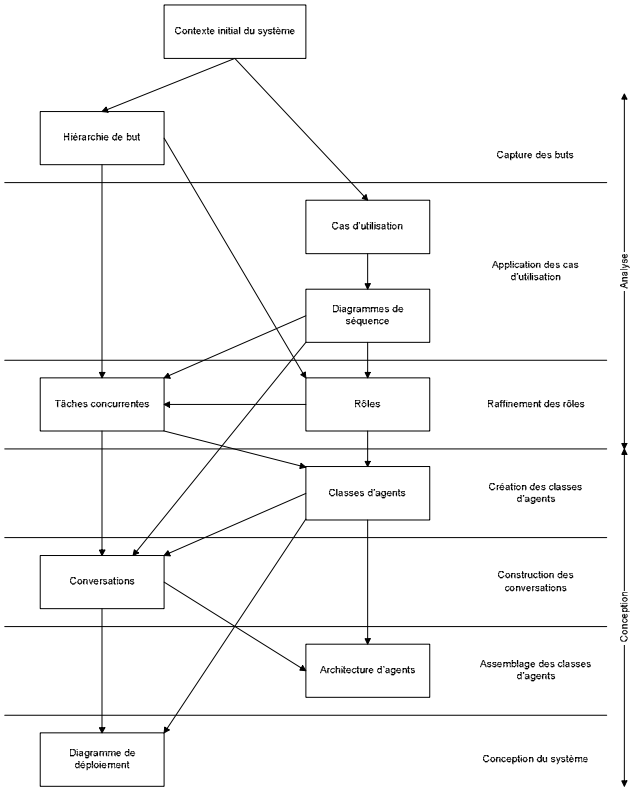


On retrouve assez nettement l'influence du modèle Aalaadin [Gutknecht & Ferber, 1999b] dans cette méthodologie avec la notion de groupe. Nous n'accordons pas autant d'importance à la notion de groupe et d'organisation, même si nous prenons en compte cet aspect là. Cette méthodologie présente néanmoins un certain intérêt.

III.7.4. MaSE

MaSE repose sur une extension des méthodologies orientées objet comme OMT (Rumbaugh's Object Modeling Technique) et UML auxquelles sont ajoutées des caractéristiques comme les buts, sensors et effectors ; les agents étant considérés ici comme des objets actifs ayant un but. Les notations orientées objets sont aussi modifiées pour prendre en compte la sémantique liée aux agents. Selon Wood et DeLoach [DeLoach, 1999], MaSE [Mase, 2000] est similaire à Gaia au niveau du respect de la généralité et du domaine d'application supporté mais propose en plus de la génération automatique de code grâce à l'outil « MaSE Tool ». Les restrictions de MaSE sont celles de Gaia avec en plus des

interactions entre agents 1 à 1 et ne propose pas le multi-cast. Cette méthode est basée sur un concept de rôle, également utilisé dans la plupart des méthodologies aujourd'hui car les rôles sont dirigés par un ou des buts. Cela est donc particulièrement adapté aux SMA. En général un système doit réaliser un ensemble de sous-buts pour accomplir un but global. Un but est une abstraction d'un ensemble de conditions fonctionnelles. L'approche générale de l'analyse avec MaSE consiste à définir des buts généraux à partir d'un ensemble de conditions fonctionnelles et à définir ensuite les rôles nécessaires pour accomplir ces buts. Bien qu'une correspondance directe entre les buts et les rôles soit possible, MaSE propose l'emploi de « cas d'utilisation » pour valider les buts systèmes et dériver un ensemble initial de rôles. Cette méthodologie consiste en deux grande phases : L'analyse et la conception.



L'analyse comporte trois étapes qui consistent à recueillir les buts, faire des cas d'utilisation et de raffiner les rôles pour satisfaire les buts. Elle consiste à produire un ensemble de rôles. Les rôles sont identiques aux rôles tenus dans une entreprise (président, vice-président etc.) qui ont des responsabilités spécifiques, des droits et des relations afin de satisfaire l'objectif global de la société. On peut ici faire une analogie avec la méthodologie GAIA. On utilise un diagramme hiérarchique de buts qui est un graphe acyclique et orienté, où les nœuds représentent les buts et les arcs représentent des relations avec les sous-buts. Ce n'est pas un arbre car un sous-but peut avoir plusieurs pères. Il est possible que dans certains cas, on imagine plusieurs buts globaux ; on définit alors un but plus abstrait qui sert de père à ces différents buts globaux qui deviennent donc des sous-buts. Une fois qu'on a un but global, on le décompose en sous-buts qui satisfont chacun une partie du but situé plus haut dans la hiérarchie. La décomposition de buts n'est pas qu'une simple décomposition fonctionnelle. Cette décomposition en buts continue jusqu'à ce qu'une autre décomposition résulte d'une condition fonctionnelle (feuille ou tâche de bas niveau) et non d'un sous-but. Un but peut avoir quatre natures possibles : résumé, partitionné, combiné et non-fonctionnel.

La phase de conception, quant à elle, comporte quatre étapes : création des classes d'agents, construction des conversations, assemblage des classes d'agent et conception du système. On peut remarquer que ce modèle propose l'affinage d'un rôle en tâches. Bien que le schéma semble faire comprendre qu'il n'y a qu'un seul flux de haut en bas pour les différentes phases, le processus est en réalité itératif. Par exemple, un but dérivé de la phase d'acquisition des buts peut être tracé jusqu'à un rôle spécifique, des tâches et la classe agent. De même, la classe agent peut être retrouvée au travers des tâches et des rôles pour le but qu'il devait satisfaire.

MaSE est actuellement en cours d'évolution pour intégrer la notion de développement d'ontologies. A noter qu'un tel outil existe déjà dans l'environnement de développement ZEUS de British Telecom. Cette méthodologie a retenu notre attention pour son affinage de

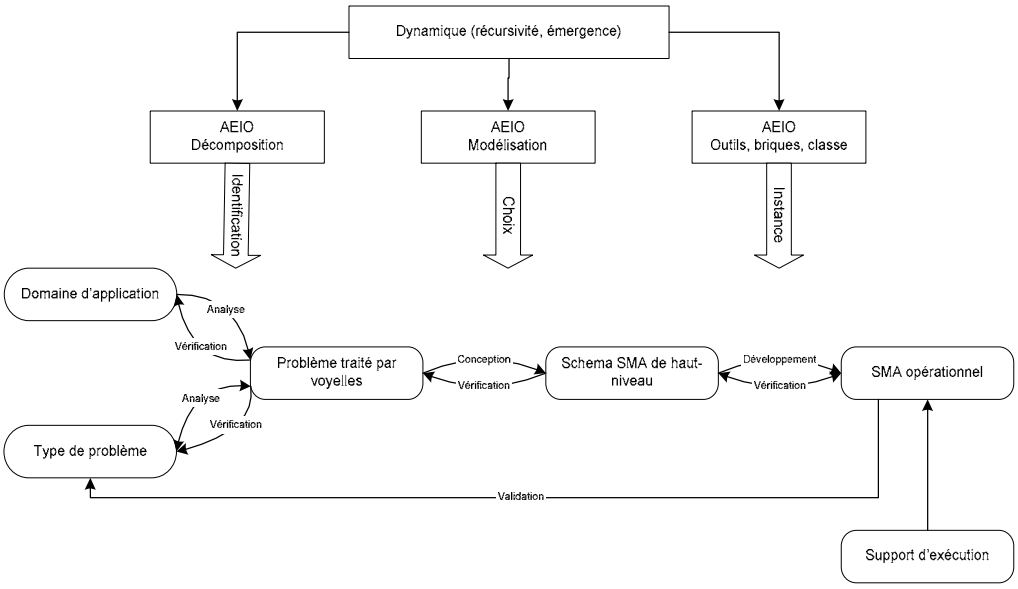
rôles en tâches mais elle ne semble pas bien adaptée au changement dynamique de rôles et de tâches dans le cycle de vie d'un agent. La critique sera donc la même que GAIA sur l'aspect dynamique.

III.7.5. VOYELLES

L'approche Voyelles, créée par Yves Demazeau du laboratoire IMAG, analyse les systèmes multi-agents selon quatre points de vue : Agents, Environnements, Interactions, et Organisations (les voyelles A,E,I,O), d'égale importance paradigmatique [Demazeau, 1995].

A noter que dans des systèmes de simulation comme la réalité virtuelle, on trouve parfois la voyelle U (oubliée dans AEIO), U comme Utilisateur, pour prendre en compte la participation active de l'opérateur humain à la simulation (l'homme est dans la boucle).

La méthode VOYELLES a donné naissance à la plateforme de développement nommée MASK (Multi-Agent System Kernel) et à la plateforme VOLCANO. Vous pourrez vous reporter à la thèse de Pierre-Michel Ricordel pour de plus amples d'informations.



L'équipe MAGMA utilise une approche intégrée des systèmes multi-agents basée sur la décomposition de ceux-ci en quatre parties (ou briques) [Demazeau, 1995] :

- Agents, qui concernent les modèles (ou les architectures) utilisés pour la partie active de l'agent, depuis un simple automate à un complexe système à base de connaissances.
- Environnements, qui sont les milieux dans lesquels sont plongés les agents. Ils sont généralement spatiaux dans la plupart des applications multi-agents.
- Interactions, qui concernent les infrastructures, les langages et les protocoles d'interactions entre agents, depuis de simples interactions physiques à des interactions langagières par actes de langage.
- Organisations qui structurent les agents en groupes, hiérarchies, relations, etc.

Outre cette décomposition en quatre briques, l'approche Voyelles est guidée par trois principes. Le premier est le principe déclaratif. Comme nous venons de le voir, et d'un point de vue déclaratif, un système multi-agents est composé d'agents, d'environnements, d'interactions, et d'organisations.

SMA = Agents + Environnement + Interactions + Organisations

D'un point de vue computationnel, les fonctionnalités d'un système multi-agents incluent les fonctionnalités individuelles des agents enrichies des fonctionnalités qui résultent de la valeur ajoutée par le système multi-agents lui-même, parfois appelée intelligence collective. Ceci constitue le principe fonctionnel.

Fonction(SMA) = Fonctions(Agents) + Fonction collective

Enfin, pour capturer l'esprit de ce que doit être la programmation orientée multi-agents, les systèmes multi-agents peuvent être considérés à un niveau d'abstraction supérieur comme des entités multi-agents. Ce dernier principe est appelé le principe de récursion.

SMA* = SMA

D'après le paradigme Voyelles, pour un problème à résoudre (ou un système à simuler) dans un domaine donné, l'utilisateur choisit le modèle d'agent, le modèle d'environnement, le modèle d'interaction et le modèle d'organisation qu'il instancie ensuite, ainsi que leurs dynamiques selon les trois principes Voyelles, de manière à engendrer le système multi-agents computationnel et déployable qui résoudra le problème dans le domaine considéré.

L'analyse d'un SMA sous tous ces aspects nous parait digne d'intérêt mais la prise en charge dynamique des rôles et des tâches ne nous a pas paru facile à représenter.

III.7.6. SCALA

La méthodologie développée dans le projet SCALA (Système Coopératif d'Agents Logiciels Autonomes) [Degirmenciyan & Marc, 2002] est très récente et présente beaucoup d'intérêt dans la mesure où elle intègre des notions d'allocation dynamique de tâche et des niveaux de spécialisation des agents. Le comportement des agents est orienté par les tâches. Les agents doivent parfois se réorganiser lorsque de nouveaux événements surgissent. Le comportement global du système est modélisé de manière fonctionnelle dans un graphe de dépendances entre les comportements élémentaires, qui sont des tâches devant être accomplies par le système. Elle se place en amont et en aval de Cassiopée. Elle présente les caractéristiques suivantes :

- modélisation au niveau global (macroscopique) par le graphe de dépendances fonctionnelles,
- modélisation au niveau local (microscopique): pré- et post-conditions des tâches,
- Définition des sous graphes par approche proscriptive à partir des évènements déclencheurs et des objectifs,
- Définition des agents, de leur niveau de spécialisation
- Simulation et allocation dynamique des tâches.

SCALA est composée de huit étapes :

1. Définition du graphe des dépendances fonctionnelles : elle qui permet au concepteur de modéliser le comportement global du SMA à un haut niveau d'abstraction. Ce graphe est composé de tâches et de comportements élémentaires. Il est possible d'exprimer des contraintes de synchronisation, d'exclusion ou d'inhibition. Un ou plusieurs agents peuvent exécuter une tâche. A ce niveau, les tâches ne sont pas allouées aux agents. La définition du graphe fournit la connaissance nécessaire pour gérer la coopération entre les agents.
2. Définition des sous-graphes : ce sont des graphes partiels du graphe des dépendances fonctionnelles. Chaque sous-graphe est associé à un but qui répond à un événement

partic ulier. Chaque but est associé à un sous-graphe décrivant les différentes procédures possibles pour l'atteindre. Les agents choisiront dynamiquement la procédure à suivre en fonction de la situation courante.

3. Identification des événements applicables : les agents sont sensibles à différents événements qui surviennent dynamiquement durant la simulation. Un nouvel événement implique que les agents prennent en compte un nouveau sous-graphe.
4. Description des comportements élémentaires et des tâches : les tâches sont des sous-buts caractérisées par un ensemble de méthodes. Certaines tâches peuvent être interruptibles et possèdent des pré et post-conditions. Ces notions sont liées à la perception de l'agent et à ses connaissances. Les pré-conditions spécifient les conditions d'exécution de la tâche et permettent le choix de la méthode. Les post-conditions décrivent les conséquences engendrées comme la modification d'une ressource. Il faut spécifier à cette étape le nombre d'agents nécessaires à l'exzécution de la tâche ainsi que les compétences nécessaires.
5. Définition des agents : on décrit ses compétences et niveaux de spécialisation, ses ressources, son rôle(s), les protocoles de communication, son but et les événements auxquels l'agent est sensible.
6. Définition des groupes d'agents : on peut spécifier dans cette phase si l'agent appartient à un groupe.
7. Organisations sociales : elles définissent les relations internes d'un groupe. Les agents jouent des rôles assignés dynamiquement en fonction de la situation. Il y a deux types d'organisations, hiérarchique et communautaire qui peuvent évoluer en cours de simulation.
8. Description des protocoles de communication : cela dépend du type d'organisation. Dans le cas d'une organisation communautaire, on utilise un système d'appel d'offre, ce qui permet par exemple à un agent de rechercher les agents nécessaires à l'exécution d'une tâche commune. Dans le cas d'une organisation hiérarchique, l'agent « gestionnaire » peut déléguer une tâche à un agent ou un groupe d'agents en fonction des connaissances qu'il a sur les tâches via le graphe des dépendances.

Cette méthodologie semble présenter beaucoup d'atouts pour modéliser les SMA dans un contexte fortement dynamique. Ce projet, mené chez Dassault Aviation entre dans le cadre de la simulation de missions aériennes, domaine très proche, contextuellement, de notre projet de simulation temps-réel ANTIC. Un des objectifs du projet est notamment de constituer des librairies de mécanismes et de protocoles réutilisables, ce qui va exactement dans l'objectif que nous nous sommes fixés. Un langage de haut niveau d'abstraction facilite la modélisation et la conception de systèmes multi-agents adaptatifs. Deux points restent à améliorer dans SCALA : la gestion des contraintes temporelles et la planification réactive. C'est avec ce dernier point que les travaux de SCALA peuvent rejoindre ceux de Amal El Fallah.

III.8. Vers une méthode de représentation des agents

Certaines méthodologies ont choisi d'étendre la notation UML [Odell & al., 2001], [Bergenti & Poggi, 2000], [Yim & al., 2000] et privilégient la modélisation des interactions entre agents afin de décrire, par exemple, des protocoles d'interaction. Des projets comme ADELPHE ou MESSAGE [Message, 2001] ont tenté d'utiliser UML en y ajoutant certaines caractéristiques. Elles laissent aussi la possibilité d'attribuer différents rôles à un agent contrairement à des méthodologies comme Gaia [Wooldridge & al., 2000] qui ne modélisent que des relations statiques. En outre, l'interface d'une classe est statique en programmation par objets et donc en UML, alors qu'elle est dynamique pour un agent. Le spectre d'utilisation

de la notation UML peut être étendu d'une part, par la notion de stéréotype et d'autre part, par la définition de nouveaux types de schémas pouvant être spécifiés au niveau du méta-modèle décrivant la notation UML elle-même.

UML (Unified Modeling Language) [Odell, 1998] unifie et formalise les méthodes de plusieurs approches orientées objets, incluant Booch, Rumbaugh (OMT), Jacobson, et Odell. UML prend en charge plusieurs types de modèles :

- les cas d'utilisation : la spécification des actions que le système ou la classe peut réaliser en interaction avec les acteurs extérieurs. Ils sont souvent utilisés pour décrire comment un utilisateur communique avec son logiciel.
- les modèles statiques : ils décrivent la sémantique statique des données et des messages d'une manière conceptuelle et d'un manière plus proche de l'implémentation (il s'agit des diagrammes de classes et de packages).
- les modèles dynamiques : ils incluent les diagrammes d'interaction (i.e., les diagrammes de séquence et les diagramme de collaboration), les schémas d'état et les diagrammes d'activité.
- les modèles d'implémentation : ils décrivent la répartition des composants sur différentes plateformes (i.e., les modèles de composants et les diagrammes de déploiement).
- le langage de contrainte par objet (OCL), qui est un langage formel simple pour exprimer la sémantique dans une spécification UML. Il peut être utilisé pour définir des contraintes sur le modèle, des invariants, des pré-conditions et des post-conditions d'opérations et des chemin de navigation dans un réseau de navigation.

Les faiblesses d'UML pour la représentation des systèmes multi-agents ont conduit une équipe de chercheurs travaillant dans différentes entreprises ou universités (Siemens, University Paderborn, Intelligent Automation, Fujitsu…) à concevoir A-UML[25]. L'objectif est de mettre au point des sémantiques communes, des méta-modèles et une syntaxe générique pour les méthodologies agents. A-UML est un des fruits de la coopération entre FIPA (Foundation of Intelligent Physical Agents) et l'OMG (Object Management Group).

Par rapport à UML, Agent-UML propose des extensions pour la représentation des agents et notamment sur trois représentations UML qui sont les packages, les templates, et les diagrammes. Cela résulte dans un nouveau type de diagramme appelée le « protocole » qui fut développé dans FIPA 1999 et qui sera d'ailleurs certainement pris en compte par l'OMG dans le cadre de développement de UML version 2.0. Les sémantiques du modèle sont représentées par un méta-modèle dont sa structure est définie par la syntaxe OCL. Il existe aussi des extensions et des contraintes à ce méta-modèle.

Aujourd'hui A-UML n'est pas encore un langage de modélisation fini et adopté par la communauté car aucune spécification finale n'a été publiée officiellement mais plusieurs publications sur des extensions d'UML ont étés publiées par les membres du groupe. Actuellement, leurs travaux se concentrent sur les interactions [Bauer & al., 2001], plus spécifiquement sur les protocoles d'interaction, mais aussi sur d'autres aspects connexes comme les langages d'interaction, la coordination, les rôles des agents. D'autres travaux portent également sur les architectures [Yim & al., 2000], et les ontologies [Cranefield & al., 2001].

L'idée générale de A-UML est de combler les déficiences de UML pour la modélisation des systèmes à agents. Parmi ces déficiences, on trouve :

[25] www.auml.org

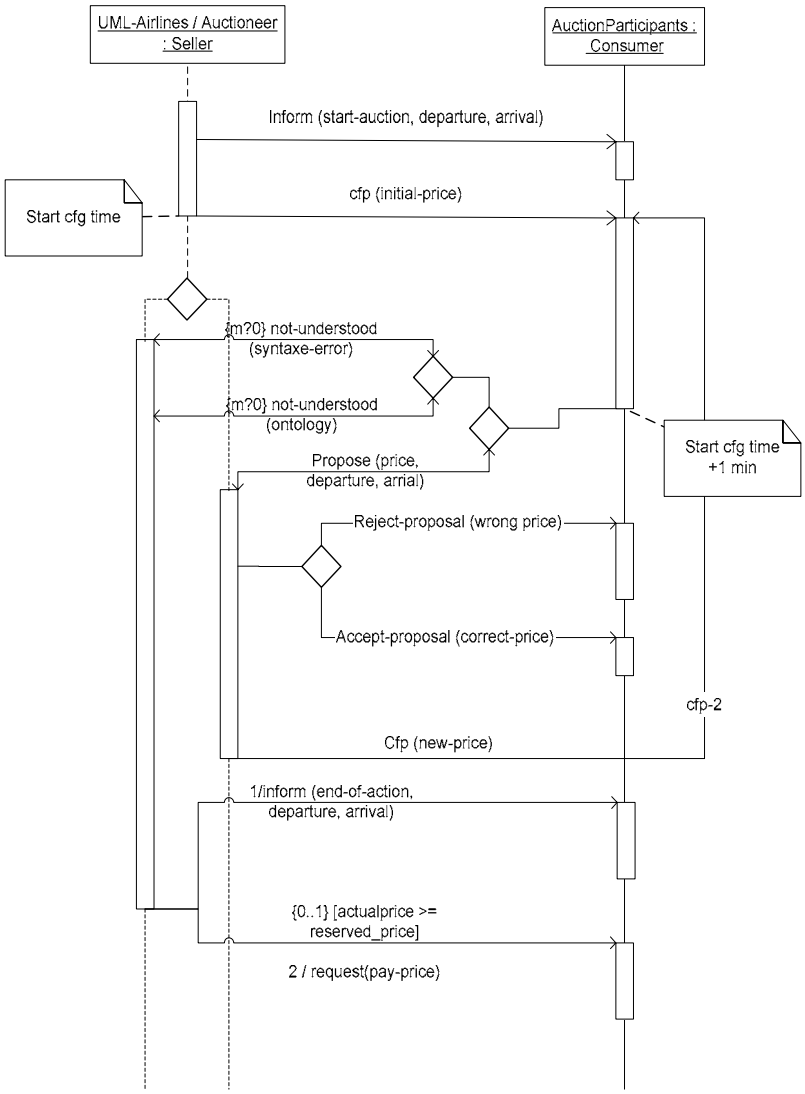
- Des relations entre classes statiques (agrégation, généralisation, et association) mais qui semblent tout de même adéquats. Il est possible d'utiliser des associations de classes et des stéréotypes pour étendre UML avec des relations spécifiques pour les agents.
- Les accointances sont des relations importantes entre agents. Il s'agit d'une relation dynamique entre des instances et UML n'est pas très adapté pour les représenter. Il faudrait pouvoir représenter des choses comme : A1 est en relation d'accointance avec A2 sous les conditions C1, C2, Cx sont réunies.
- Un certain nombre de concepts de haut niveau (comme les engagements, les contrats, etc.) peuvent être relativement bien représentés avec UML mais d'autres (comme les croyances et les intentions) ne le peuvent pas.
- Il est difficile de représenter l'état interne des agents. Il faudrait un modèle proposant des concepts de haut niveau « cognitif » (BDI, BC, GAP, …).
- UML n'est pas efficace pour représenter des connaissances fonctionnelles (buts, planification, process, etc.). Pourtant beaucoup de méthodologies agents utilisent les buts et la décomposition de buts en sous-buts.
- Il n'est pas évident que les approches de modélisation à états finis soient adaptées pour les agents. Les agents ont des espaces d'états vastes qu'il n'est pas évident de partitionner en un plus petit nombre de macro-états de plus haut niveau. Les agents peuvent apprendre et s'adapter à différentes choses et des paramètres comme les croyances interagissent pour influencer le comportement de façons subtiles. Ces systèmes sont dynamiques, non linéaires et ont un comportement émergeant.

A-UML peut représenter un cycle de vie complet d'un système à agents. Intéressons-nous pour l'instant à une première partie d'A-UML qui concerne la spécification des protocoles d'interaction d'agent (AIP) qui est une partie de la spécification des modèles dynamiques des systèmes à agents. En UML ce modèle est présenté par des diagrammes d'interactions, des diagrammes d'états, et des diagrammes d'activités :

1. Les diagrammes d'interaction, i.e., les diagrammes de séquences et les diagrammes de collaboration sont utilisés pour définir des comportements de groupes d'objets. En général, un diagramme d'interaction représente le comportement d'un cas d'utilisation. Ces diagrammes sont surtout utilisés pour définir des interactions basiques entre des objets au niveau de l'invocation des méthodes. Ils ne sont pas bien adaptés pour décrire des interactions sociales complexes des SMA.
2. Les diagrammes d'état sont utilisés pour modéliser le comportement d'un système complet. Ils définissent tous les états possibles des objets et décrivent comment un objet change d'état en fonction d'un événement reçu. Ils sont bien adaptés pour décrire le comportement d'un seul objet dans différents cas d'utilisation mais peu adaptés pour décrire le comportement d'un groupe d'objet coopérants.
3. Les diagrammes d'activité sont utilisés pour décrire des ensembles d'événements/actions pour plusieurs objets et cas d'utilisation. Ils sont bien adaptés à tout.

Les protocoles d'interaction proposés par A-UML combinent des diagrammes de séquences à des diagrammes d'états. Il décrit :

1. une forme de communication avec une séquence attendue de messages entre les agents accomplissant différents rôles et des contraintes sur le contenu des messages.
2. Une sémantique qui consiste à des actes de communication (CA) à l'intérieur de la forme de communication.



La figure ci-dessus montre une négociation pour vente de ticket d'avion.

On constate la récente émergence de quelques projets modélisés en A-UML comme des projets de SCM (Supply Chain Management) [Huget, 2002]. Et puisque la technologie ne cesse d'évoluer, se prépare déjà une amélioration d'A-UML très récente, nommée MA-UML pour les agents mobiles [Hachicha & al., 2002].

III.9. Conclusion

Nous avons passé en revue quelques méthodologies d'analyse et de conception de SMA qui conviendraient de près ou de loin à notre présente étude. Il ressort de cette étude qu'une méthodologie est souvent en relation avec son modèle d'organisation d'agents et qu'il est difficile de trouver une méthode générique applicable universellement. Parmi celles-ci, certaines sont mieux adaptées que d'autres pour la conception de SMA adaptatifs. Il est bien évident que les méthodologies basées sur l'objet ne sont pas suffisantes pour concevoir des systèmes multi-agents car elles ne prennent pas en compte les caractéristiques d'autonomie et de socialité de l'agent ainsi que les caractéristiques de complexité des interactions et de

dynamicité du système. C'est la raison pour laquelle, aujourd'hui, de nombreux travaux de recherche visent à améliorer les méthodologies actuelles [Tveit, 2001], [Iglésias & al.,1999], [Wood & DeLoach, 2000], [Wooldridge & Ciancarini, 2000]. Parmi les méthodologies existantes, nous avons retenu GAIA pour son approche basée sur les rôles et les interactions mais elle présente le désavantage de s'intéresser à une structure organisationnelle statique, les capacités des agents et des services qu'ils offrent n'évoluent pas dans le temps. Cette contrainte est lourde pour notre approche qui se veut totalement dynamique. Cassiopée et Aalaadin utilise fortement la notion de groupe. Ces approches sont intéressantes mais trop restrictives à un modèle d'organisation (AGR). MaSE présente beaucoup d'intérêt avec une notion de hiérarchie de but et de tâches concurrentes. Cependant la prise en compte de la dynamique d'évolution n'apparaît pas clairement et cette méthodologie ne semble pas être très adaptées au AMAS. SCALA est très probablement celle qui a retenu le plus notre intérêt avec une prise en compte du niveau de spécialisation des agent et une allocation dynamique de tâches. Le contexte de SCALA (aviation militaire) est très proche de notre contexte d'application (simulation d'intervention pour les sapeurs-pompiers) et la méthodologie semble parfaitement applicable à notre projet. Un seul bémol à SCALA, la gestion du temps et la planification qui ne sont pas encore prise en charge. Le projet ANTIC, présenté au chapitre huit, requiert une prise en charge du temps pour la gestion des scénarii de simulation. En ce qui concerne le système de représentation de la dynamique des SMA, la notation AUML nous parait tout à fait digne d'intérêt.

Chapitre IV. Le modèle YAMAM

« Il n'y a pas de grandeur pour qui veut grandir. Il n'y a pas de modèle pour qui cherche ce qu'il n'a jamais vu » – Paul Eluard.

Résumé : Nous présentons dans ce chapitre un nouveau modèle d'organisation pour les SMA. Celui-ci peut s'appuyer sur plusieurs méthodologies existantes sans être véritablement dépendant de l'une d'entre elles. Ce modèle repose sur un substrat appelée « l'énergie », qui permet aux agents de se reconformer dynamiquement en fonction des changement de l'environnement.

IV.1. Avant-Propos

Nous avons recensé dans un chapitre précédent quelques méthodologies d'analyse et de conception de SMA. Chacune se focalise sur des points particuliers avec souvent, pour dénominateur commun, une représentation des notions d'interaction et d'organisation. Parmi ces méthodologies, certains sont assez génériques et peuvent être utilisées pour tout type de SMA et permettre ainsi une implémentation avec différents outils ; certains étant bien entendu plus adaptés que d'autres.

Des travaux récents ont porté sur des modèles d'organisation pour les SMA et les SMA adaptatifs. On voit à présent une certaine émergence de résultats d'études débutées il y a plus de cinq ans. Gasser introduit dans son système Mace [Gasser, 1992] les concepts de rôles pour définir son SMA. Ferber et son équipe ont axé leurs recherches sur le niveau organisationnel. Ferber propose une architecture fort intéressante dans son modèle Aalaadin basée sur le concept d'un triplet Agent-Groupe-Rôle. Un agent jouant un ou plusieurs rôles dans un ou plusieurs groupes. Ferber propose également dans son ouvrage [Ferber, 1999] des approches permettant d'analyser et de classer les SMA. Le Strugeon [LeStrugeon, 1995] décrit des règles de réorganisation en proposant une approche de l'organisation par arborescence en tant que répartition des tâches et des comportements. On peut citer aussi les travaux de B. Durand [Durand, 1996] sur l'utilisation de mécanismes de rôles de schémas d'organisation de SMA. [Foisel, 1996] propose un modèle ayant un aspect statique pour décrire les structures d'organisation et un aspect dynamique pour les faire évoluer. Un modèle hiérarchique est implémenté dans la plate-forme SMA Magique [Bensaid & Mathieu, 1996]. Enfin des méthodes comme Cassiopee [Collinot & al., 1996] fournissent un support intéressant à la conception d'organisations.

Selon les motivations de leurs auteurs, certains travaux se focalisent davantage sur un concept particulier comme la notion de rôle ou la notion d'interaction. Certains sont même particulièrement orientés pour la résolution de problèmes spécifiques. Tous ces travaux de recherche présentent chacun un intérêt certain mais nous n'avons pas retrouvé dans un seul et même modèle d'organisation toutes les caractéristiques qui nous semblaient essentielles pour concevoir une architecture modulaire d'agents à reconformation dynamique. Nous avions également un besoin de prise en charge dynamique d'un rôle par un agent et si certains modèles proposent la notion de rôles et d'agents, tous ne permettent pas à un agent de prendre

en charge dynamiquement un rôle. De même la décomposition plus fine d'un rôle n'est pas systématiquement intégrée dans les méthodologies étudiées dans le chapitre trois.

Notre problématique originelle était de fournir un outil logiciel effectivement opérationnel pour répondre aux différents projets de simulation de notre laboratoire. Nous avons choisi cependant de ne pas nous focaliser sur notre seule problématique de simulation et de concevoir un modèle suffisamment générique pour répondre à plusieurs types de besoins.

Nos travaux se situent au carrefour des différentes approches évoquées précédemment en proposant un nouveau modèle d'organisation pour les SMA adaptatifs et un middleware[26] orienté-agent pour implémenter ces organisations d'agents. C'est la conception de ce logiciel adaptatif qui nécessitait en amont un nouveau modèle qui nous permette d'organiser nos agents, de façon fondamentalement différente de ce que proposent les autres modèles. Avant la phase de conception de notre plate-forme, nous avons donc décidé de nous atteler à définir un nouveau mode d'organisation d'agents, qui sera présenté en détail dans ce chapitre. Ce modèle adopte non seulement une modularité maximale mais constitue aussi une première approche vers un système auto-organisé au sens de Varela. Cette voie de recherche justifie à elle seule la conception d'un nouveau modèle d'organisation d'agents.

IV.2. Présentation du modèle YAMAM

Traditionnellement, le concepteur d'un SMA se préoccupe des actions de ses agents et de son comportement. Ce comportement résulte de l'espace d'état, défini par les valeurs possibles de ses variables internes, lui conférant ainsi un semblant d'état mental régit par des principes fondamentaux tels que le désir, l'intention, le but, les croyances etc. L'agent possède aussi une dimension sociale, c'est-à-dire qu'il communique et interagit avec d'autres agents pour accomplir ses buts. Il est souvent postulé que des structures sociales d'agents se formeront par émergence des interactions.

J. Ferber [Ferber & Ghallab, 1988] considère que la construction d'un système complexe ne peut se faire raisonnablement sans prendre en compte les concepts organisationnels. Il propose donc dans son modèle Aalaadin une organisation possible d'agents reposant sur le triplet Agent – Groupe – Rôle. On pourrait parler de modèle d'organisation spécifique ou encore de design pattern[27]. Un design pattern [Pree, 1998] étant une manière de regrouper des concepts (des classes) pour leur réutilisation. En d'autres termes, ils fournissent un support de développement et une réutilisation des frameworks, eux-mêmes étant des hiérarchies de classes plus des modèles d'interactions qui peuvent être transformés en applications complètes au travers de divers types de spécialisation.

Un agent représente ici une entité autonome communicante, un rôle représente une fonction ou un service et un groupe fournit une notion simple de regroupement d'agents. Madkit [Gutknecht & Ferber, 1999a] est né de cette proposition d'organisation et constitue aujourd'hui une bonne plateforme de développement. Dans la lignée de l'école Ferber, nous considérons qu'une organisation peut être décrite uniquement par sa structure inhérente. Les phénomènes d'interaction, de coordination, de coopération sont des concepts plus importants que l'agent lui-même. On entend souvent dire que c'est surtout ce que les autres individus

[26] Un middleware est une couche applicative de niveau intermédiaire entre un système d'exploitation et une application

[27] Selon la définition de son inventeur, un Design Pattern décrit un problème récurrent dans un environnement donné puis décrit une solution à ce problème de telle manière que cette solution soit réutilisable à chaque fois qu'on rencontre le problème sans cependant qu'elle soit deux fois exactement la même.

pensent de nous qui font de nous ce que nous sommes. La présence d'un agent n'a de sens que lorsqu'il est plongé dans un environnement dans lequel évoluent d'autres agents, sinon, il est réduit à l'état d'un simple processus informatique communicant.

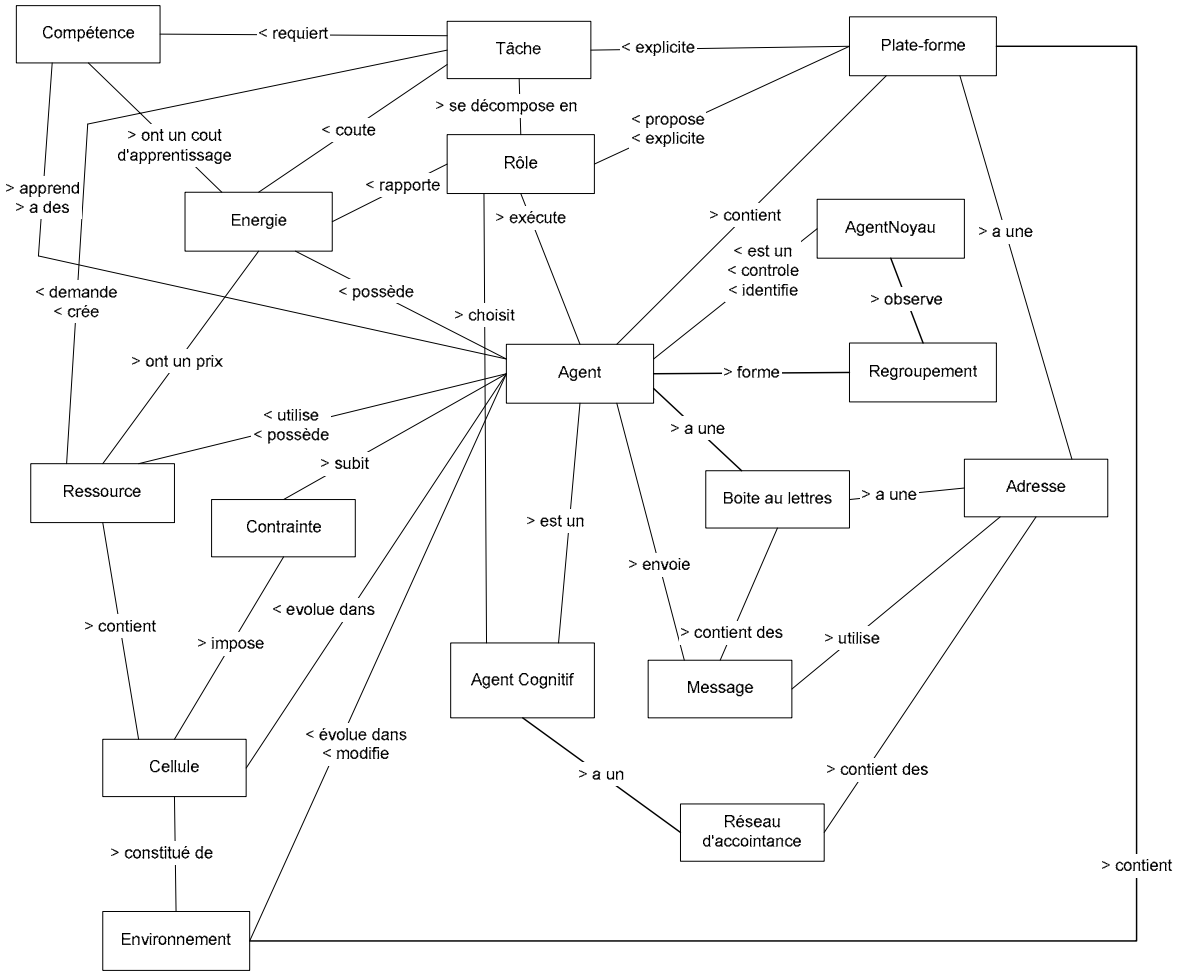
Les relations liant les agents aux autres agents sont donc primordiales par rapport à l'agent lui-même et à son comportement. Son comportement individuel étant plus la résultante de ses interactions en société. Nous nous sommes donc intéressés aux structures d'organisations d'agents qui régissent tout le fonctionnement du système multi-agents. Il n'est pas question ici de concevoir une nouvelle méthodologie d'analyse. Nous nous reposerons sur des méthodologies existantes, notamment GAIA, Cassiopée ou SCALA.

Notre démarche consiste à réaliser une série d'ontologies de domaines, d'en extraire leurs principaux concepts communs et de créer un modèle complet autour de cette forme d'organisation. La démarche est sensiblement identique à la conception d'un logiciel standard, excepté que nous intégrons des notions supplémentaires, liées à la nature intrinsèquement dynamique des organisations d'agents.

IV.2.1. Ontologie liée au modèle

Notre modèle est né de l'intersection entre des ontologies de plusieurs projets de notre laboratoire (ANTIC, SGTC[28], …). Les phases d'analyse de chacun de ces projets manipulaient un certain nombre de concepts communs comme les tâches, les ressources, l'environnement… Nous présentons ci-dessous une vue partielle d'une ontologie liée à un projet de simulation.

[28] SGTC : Système de Gestion de Terminaux à Conteneurs, projet consistant à optimiser le place des conteneurs sur les quais et dans les navires.



La place de l'agent dans son environnement restera à définir par le concepteur, mais l'environnement étant déterminé pour un type de système à modéliser, il ne changera pas de structure fondamentale et donc n'interviendra pas au niveau organisationnel. Pour cette raison, nous n'en parlons que dans la section concernant la conception et l'implémentation de la plate-forme. L'agent joue simultanément un rôle (*n* rôles si on intègre la notion de temps) dans un environnement en possédant un certain nombre de compétences nécessaires à la tenue de son ou ses rôles. Un rôle est toujours local à un environnement et repose sur une liste de tâches à accomplir séquentiellement ou non.

Nous avons extrait de ces ontologies quatre composantes essentielles : L'agent, le rôle, les tâches et les compétences. Ces quatre composantes principales donnent naissance au modèle organisationnel simple de YAMAM présenté ci-dessous.

IV.2.2. Le modèle organisationnel simplifié

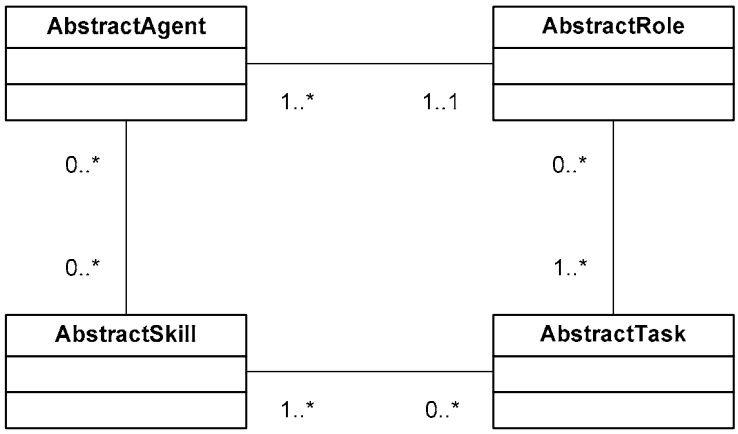
Nous proposons dans notre modèle d'organisation d'agent, une manière de lier l'agent à son rôle et à ses tâches en utilisant ses compétences[29] propres.

Il est baptisé YAMAM (Yet Another Multi-Agent Model), non sans un petit clin d'oeil à Lex & Yacc [Nino, 1995] et [Johnson, 1975].

Il repose sur quatre principes : l'agent (A), la compétence (C), le rôle (R) et la tâches (T). La figure suivante montre une modélisation UML de ce quadruplet A-C-R-T.

[29] La notion de compétence est développée plus loin dans ce chapitre

Nous détaillerons dans les paragraphes suivants chaque entité de ce modèle d'organisation.



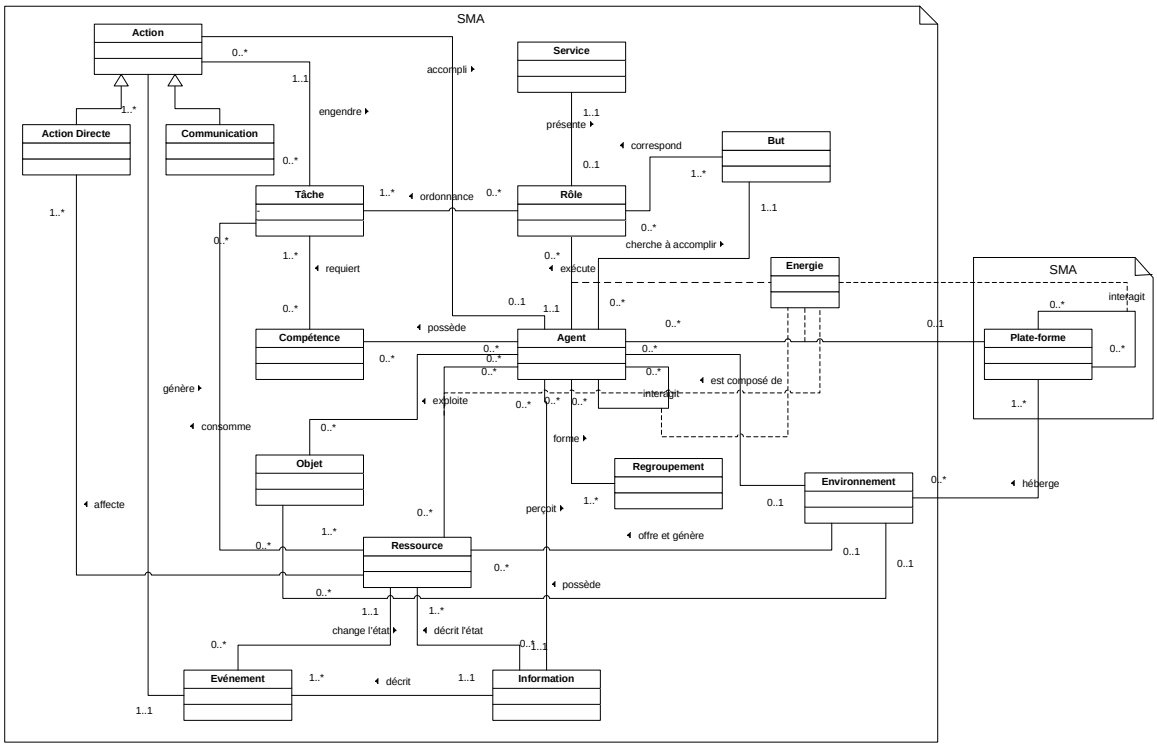
Modélisation des notions d'agent, de tâches, de rôle et de compétence en représentation UML

Une description du modèle YAMAM peut être trouvée dans [Savall & al., 2001].

IV.2.3. Vers un modèle organisationnel complet

Nous retrouvons ce quadruplet (modèle simple) dans un modèle plus complet présentant les différentes entités que nous retrouvons dans un SMA : des ressources, des objets, un environnement… Une plate-forme héberge un ou plusieurs environnements. Un environnement, des objets, des ressources et des agents composent un SMA. L'agent est lié intrinsèquement aux notions de rôle, de tâche et de compétence. Tout cet ensemble est régit par un système d'échange d'énergie que nous détaillerons plus loin dans ce chapitre. A un niveau plus haut de granularité, une plate-forme interagissant avec d'autres plates-formes, via un réseau qui représente un environnement implicite, forme un autre SMA qui utilise par exemple, les objets et les ressources du réseau pour communiquer. Les plates-formes interagissent entre elle avec le même système d'échange d'énergie.

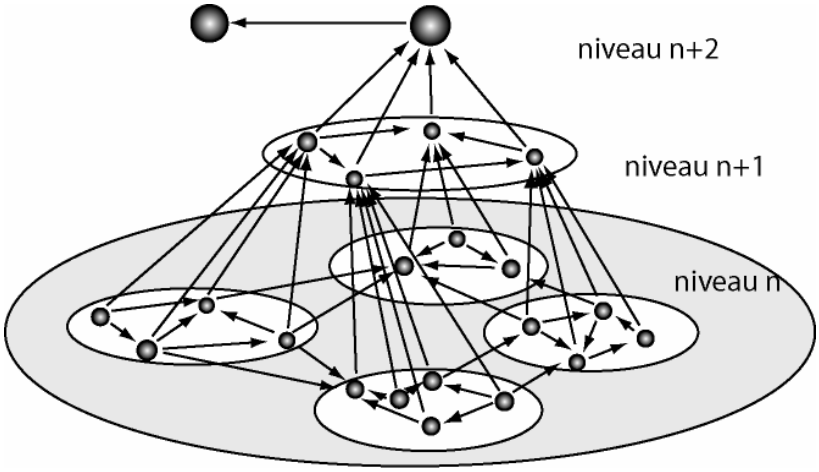
Les regroupements d'agents sont observés par un agent spécial de la plate-forme dans le but de proposer « artificiellement » des coopérations d'agents. Un agent collabore en priorité avec d'autres agents présents dans son réseau d'accointance. Ce réseau permet à un agent de mémoriser les coordonnées d'un agent possédant un rôle ou des compétences particulières. Un agent peut ainsi sous-traiter une tâche rapidement à un autre agent qui possède déjà les connaissances nécessaires. Cependant, un tel agent sous-traitant n'est pas nécessairement connu du premier et la recherche d'un agent requiert une dépense d'énergie. L'agent noyau peut proposer le rapprochement entre agents pour minimiser la dépense énergétique de la plate-forme. A noter qu'un agent « broker » (courtier) pourrait aussi avoir le rôle d'effectuer cette mise en relation entre les agents.



La notion d'organisation n'est pas centrale sur ce diagramme. Le modèle Aalaadin voyait une organisation comme un simple regroupement d'agent. Une organisation est présentée ici comme un ensemble d'agents qui tiennent des rôles. La façon dont sont organisés les agents constitue la structure de groupe. L'organisation peut être hiérarchique dans laquelle chaque agent a un « grade ». Typiquement une organisation militaire. Il se peut qu'il n'y ait aucune organisation ce qui génère immanquablement une certaine anarchie et un dysfonctionnement certain… La dynamique d'auto-organisation du système engendrera la prise en charge dynamique des rôles par les agents.

Niveaux de complexité et d'organisation

Dû aux différents points de vue d'abstraction fonctionnelle, il est possible de représenter un système de différentes façons. D'une manière générale, un système peut être décomposé en plusieurs sous-systèmes. Pour ce qui nous concerne, la décomposition d'un système multi-agents en agents peut être récursive dans le sens où un agent peut être lui-même un système multi-agents. En effet, un agent n'est pas nécessairement un composant atomique mais il peut être au contraire un assemblage de composants en interaction. Ce concept de niveau d'organisation permet de manipuler le système au niveau de *granularité* adéquat. Par exemple, le monde économique est un ensemble d'entreprises en interaction. A un autre niveau, le fonctionnement d'une entreprise est un ensemble d'agents (employés) organisés et interagissants. A un autre niveau encore, un agent-employé peut être décomposé en cellules biologiques ; une cellule biologique étant elle-même divisée en un ensemble de macro-molécules etc.

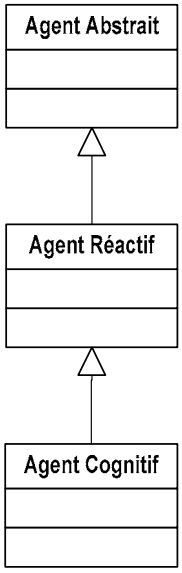


Nous présentons donc sur le diagramme l'organisation d'un SMA complet. Ce SMA peut être lié directement avec un autre SMA, comme de simples agents pourraient communiquer ensemble. Il est donc possible d'abstraire un ensemble complet d'agent par une seule entité, plus facilement manipulable, qu'on appelle le SMA.

IV.2.3.1. L'agent

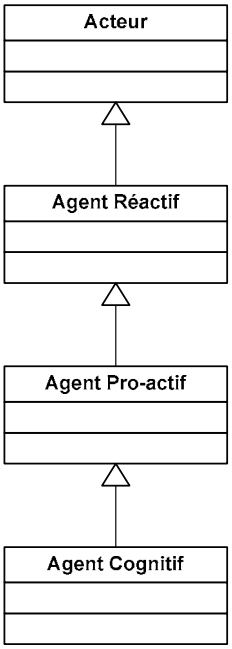
Il n'y a pas d'a priori sur ce qu'est un agent. Il reste à un niveau de description conceptuelle, une entité plus ou moins autonome et communicante plongée dans un environnement qui peut être implicite ou explicite. Nous laissons volontairement une certaine liberté dans la définition de l'agent, pour éviter des barrières lors de l'exploitation de notre modèle dans des cadres relevant plus de la mobilité de processus que de la notion d'organisation d'agent au sens où les puristes l'entendent.

Nous avons construit notre agent générique sur le principe suivant : un agent a des caractéristiques originellement réactives. Nous verrons dans le prochain chapitre qu'un agent réactif ne peut acquérir de lui-même de nouvelles compétences.



La représentation que j'ai choisie dans mes travaux diffère de celle présentée dans le chapitre « Agents et SMA ». Elle intègre l'agent réactif dans l'agent cognitif, considérant qu'une réaction d'ordre « cognitive » est un raffinement plus complexe basé sur une réaction basique d'un agent réactif. On s'accorde à dire que l'homme est un agent cognitif, que ses décisions sont le résultat d'une intense cogitation ; Pourtant lorsque vous posez votre main sur une flamme, un réflexe inconscient surpasse toute réflexion philosophique transcendante débattant de la sagesse qualifiée de déplacer sa main de façon à atténuer la douleur qui pourtant se réduit à une simple information. J'ai choisi une autre forme de représentation n'intégrant pas, de ce fait, la notion d'agent hybride. Cette nouvelle représentation ne réduit pas le problème de l'altérité entre l'agent réactif et l'agent cognitif et ne définit pas non plus pour quel niveau de complexité on doit utiliser l'un ou l'autre. Le modèle YAMAM, adoptera cette forme de représentation. La représentation en notation UML est fournie dans la figure suivante ci-contre.

L'agent est du type réactif ou cognitif. A noter que nous incluons dans le terme d'agent réactif, la notion d'acteur. L'acteur se contente d'exécuter des tâches prédéterminées sans aucune autonomie. L'acteur est une sorte de « sous-agent » réactif, ne répondant même pas à des stimuli. Il est réduit à l'état de processus et se contente d'accomplir son rôle sans prendre en compte les perturbations de l'environnement.



L'agent réactif prend en compte les différentes informations véhiculées via les stimuli lui parvenant. Il va donc réagir en fonction des nouvelles informations qu'il reçoit de l'environnement ou des autres agents. A un stimulus, il réagit par une action prédéterminée.

L'agent cognitif, quant à lui intègre une notion de pro-activité. Cette notion est génératrice de l'autonomie. L'agent cognitif adopte donc un comportement initialement pro-actif qui lui permet de générer et de réaliser des buts. Il va donc chercher à accomplir des nouveaux rôles et ne se contentera pas seulement d'exécuter des rôles imposés par son « créateur »[30].

Nous considérons que l'agent dit « cognitif » est un raffinement d'un agent réactif dans la mesure où ses compétences sont de plus haut niveau. Nous traduisons cette notion de raffinement directement dans un diagramme d'héritage de classe UML. L'agent cognitif sera donc un agent évolué possédant un système décisionnel élaboré.

Seul l'agent noyau peut le forcer à acquérir dynamiquement une nouvelle compétence. Si on souhaite au contraire que l'agent puisse évoluer au cours du temps, il faut alors utiliser directement un agent dit « cognitif » héritant ses caractéristiques d'un agent réactif. Ainsi, plus on augmente les compétences de l'agent plus il devient cognitif, mais cette frontière n'est volontairement pas délimitée précisément pour ne pas dériver sur des concepts d'agent hybride.

L'agent réactif est en effet la forme la plus basique d'un agent, il n'a pas de système décisionnel et n'est pas capable d'apprendre de nouvelles compétences. Il ne réagit qu'à des stimuli prédéterminés correspondant à l'exécution de son rôle. Tout au long de son existence, l'agent réactif ne changera pas de rôle sauf si l'agent en charge de la plate-forme ne le décide. L'intérêt des agents réactifs est d'effectuer des tâches indispensables au fonctionnement du système quelle que soit sa situation (réduction à un acteur) ou de simuler des systèmes réactifs (cas de la fourmilière).

L'agent cognitif est une forme d'agent plus évoluée puisqu'il est tout d'abord capable d'acquérir de nouvelles compétences auprès d'autres agents. De plus, l'agent cognitif est pourvu d'un semblant d'« état mental ». Celui-ci lui est utile pour prendre des décisions assurant la survie de l'agent, garantir sa pérennité, et tirer des expériences sur les choix qu'il a pu faire auparavant. Dans tous les cas, un agent connaît les ressources dont il dispose. Nous verrons cela ultérieurement dans un chapitre sur l'organisation des sociétés d'agents.

IV.2.3.2. Le rôle

Un rôle peut représenter un service ou une fonction. Cela peut être une forme d'identification d'un agent. Un agent prend en charge un ou plusieurs rôles dans le temps

[30] Nous verrons dans le prochain chapitre que le développeur d'applications agents n'affecte pas une mission (un rôle) à un agent spécifique car c'est à l'agent de prendre en charge la mission de son choix, proposée sur la plate-forme. C'est un premier pas vers la notion d'autonomie.

mais un seul est traité simultanément (la cardinalité est noté 1..1 sur le diagramme UML). Un rôle implique l'accomplissement d'une ou plusieurs tâches, donc dans le cas le plus simple, tenir un rôle se réduit à accomplir une seule tâche. Nous considérons qu'un agent peut tenir un rôle s'il possède suffisamment de compétences pour cela. Nous ajoutons donc à notre modèle la gestion d'une ou plusieurs compétences nécessaires à un rôle.

Par exemple, si le rôle d'un agent est de scruter l'arriver d'un email, il doit savoir communiquer avec le protocole TCP/IP, savoir interroger le serveur en émettant une requête puis il doit savoir analyser le résultat renvoyé par le serveur. Le rôle est aussi un regroupement de tâches à accomplir selon un ordre déterminé. Le rôle peut s'exécuter par exemple toutes les 5 minutes. Il est possible de considérer que ce rôle est constitué d'une seule tâche « *scruter_mail_task »* ou bien de décomposer le rôle en deux tâches : *« émettre_requete_serveur_task* » puis « *analyser_réponse_task* ». Pour cela, l'agent doit savoir communiquer donc posséder la compétence adéquate. De la même façon que le rôle peut être décomposé en une ou deux tâches, il est possible de lier la tâche à une ou des compétences. La tâche « *scruter_mail_task* » peut être liée à la compétence du même nom « *scruter_mail_skill* » ou éventuellement à deux compétences « *emettre_requete_serveur_skill* » et « *analyser_réponse_skill* ». Dans le cas où le rôle est décomposé en deux tâches, on lie chaque tâche à une compétence.

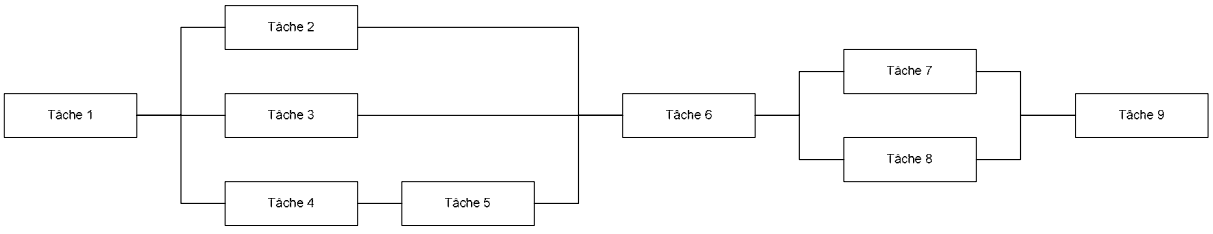
Dans le cas le plus simple, un rôle consiste à accomplir un certain nombre de tâches de façon séquentielle et non répétitive. Il se peut cependant que des tâches doivent être accomplies à des instants précis ou de façon cyclique. Il est aussi possible qu'une tâche ne puisse être accomplie sans le résultat d'une autre tâche. Nous touchons alors au domaine de l'ordonnancement et de la planification de tâche [ElFallah & Haddad, 1996]. Il est possible de représenter les agents et leurs rôles avec des diagrammes UML de « cas d'utilisation » mais UML standard ne prend pas en charge la dynamique d'interaction des agents.

IV.2.3.3. Les tâches

Une tâche est une action élémentaire. Elle nécessite une ou plusieurs compétences pour sa réalisation. Un agent devra donc posséder sinon acquérir les compétences nécessaires à la réalisation d'une tâche pour tenir un rôle.

Un rôle peut nécessiter la réalisation de plusieurs tâches et doit donc décrire leur ordonnancement. Une tâche peut par exemple dépendre du résultat d'une autre tâche.

L'agent doit donc lire dans le rôle pris en charge la façon dont les tâches doivent être ordonnancées et traitées.



La gestion des tâches est locale à un agent. Nous travaillons aussi sur l'interfaçage avec un ordonnanceur pour qu'un agent attente les résultats d'une tâche réalisée par un autre agent pour poursuivre la réalisation de ses propres tâches. Cela permet d'utiliser une réelle notion de planification de tâches chez les agents. A noter qu'un agent a la possibilité de « sous-traiter » une tâche à un autre agent. Auquel cas, un agent n'ordonnance pas ses tâches

mais ordonnance d'autres agents à la façon d'un chef d'équipe. On utilise le terme d'agentification systématique des tâches ou des services.

Un agent doit pouvoir exécuter certaines tâches en parallèle. Cela implique néanmoins un thread par tâche et une synchronisation des threads lorsque qu'une troisième tâche attend le résultat de deux autres exécutées en parallèle. Nous verrons dans la partie implémentation que Phoenix crée un agent fils (donc un thread) et délègue la tâche à exécuter en parallèle à ce fils.

Créer un agent juste pour déléguer une tâche, tel qu'un calcul, ne se justifie pas forcément car un agent possède une occupation mémoire et processeur relativement importante et engendre des coûts de communication. Donc si la tâche se réduit à un simple calcul, l'agentification de la tâche engendre plus d'inconvénients que d'avantages. De plus si on respecte la définition de l'agent qui doit être une entité plus ou moins autonome et communicante ; toutes ces qualités sont bien inutiles pour exécuter un calcul basique. Cependant la création d'un agent peut se justifier si le résultat du calcul peut être donné par un autre agent ayant déjà réalisé ce calcul ultérieurement. Auquel cas, il n'y a qu'à transmettre l'information d'agent à agent sans réaliser à nouveau le calcul. Nous verrons ultérieurement une petite application (toy-problem) qui consiste à fait calculer une suite de Fibonacci à des agents – chacun mémorisant son résultat de calcul. Un agent à parfois l'alternative entre exécuter le travail lui-même ou le sous-traiter. Sous-traiter est cependant risqué dans la mesure où il faut trouver un agent compétent et vérifier ensuite le travail réalisé. La simple recherche d'un agent sous-traitant peut se faire dans le réseau d'accointance, sur la plate-forme locale ou sur plusieurs plates-formes. Cela engendre à chaque fois des coûts supplémentaires. La recherche d'un agent sous-traitant peut aussi échouer, auquel cas c'est une perte nette pour l'agent demandeur.

Si un agent a pour mission d'effectuer un calcul lourd en temps CPU, il peut y consacrer un temps considérable s'il se trouve sur une plate-forme PC disposant d'une faible puissance. En revanche, cet agent peut sous-traiter à un autre agent tournant sur une plate-forme hébergée sur un super-calculateur, auquel cas le résultat lui sera plus vite retourné. Un agent est dépendant non seulement de ses compétences mais aussi des ressources qui lui sont attribuées. Ainsi, un agent peut ajouter dans son réseau d'accointances des agents spécialisés et performants dans la réalisation d'une tâche particulière. C'est l'expérience de l'agent qui au cours de sa vie va connaître et sélectionner d'autres individus. C'est à l'image de l'entreprise qui sélectionne ses fournisseurs. Un fournisseur peut produire du travail rapide et fiable s'il dispose des ressources pour cela. Un autre, plus petit, rendra le travail dans un délai plus important car disposant de moins de ressources. A contrario, un agent très spécialisé peut aussi être surchargé du à la forte demande. Auquel cas un autre agent peut être moins spécialisé peut parfois fournir un résultat dans un meilleur délai.

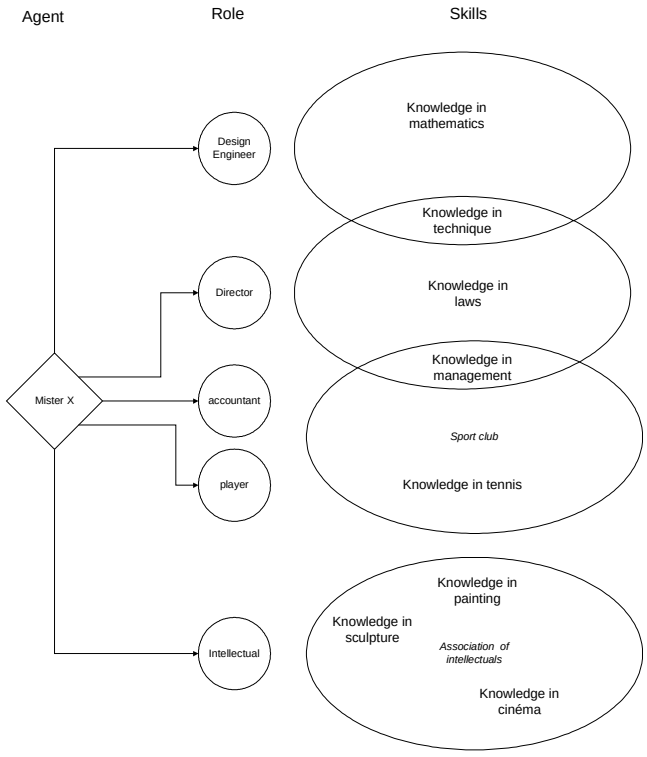
Nous verrons dans un chapitre ultérieur qu'il est non trivial dans certains cas de quantifier les ressources associées aux agents. Celles-ci pouvant être des possibilités calculatoires ou la valorisation d'un bon réseau d'accointance. On pourrait faire un parallèle avec un bon commercial d'entreprise qui possède son fichier client.

IV.2.3.4. Les compétences

Les actions traitées par l'agent sont bien souvent intégrées à l'agent lui-même dans la plupart des modèles. Nous avons ajouté une fonctionnalité essentielle qui est la notion de compétence. Une compétence étant une entité regroupant un savoir et un savoir-faire. Nous définissons une compétence (*skill*) comme une unité de connaissance atomique ou non, permettant la réalisation d'une tâche donnée. Le concept de compétence doit être vu au même

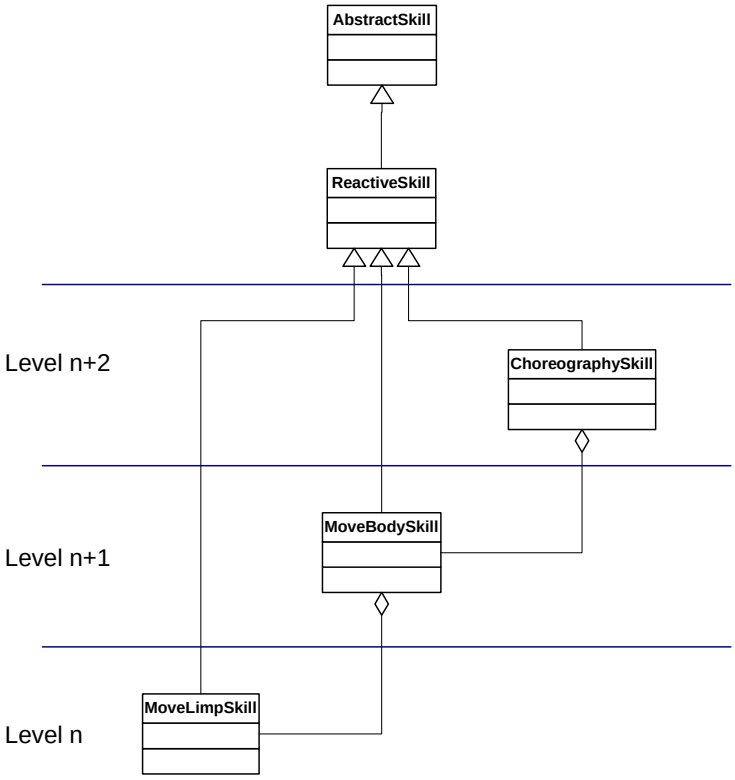
niveau organisationnel que le rôle ou les tâches car le rôle est dépendant directement des compétences. Cette notion de compétence est également présente dans le modèle MAGIQUE [Routier & al., 2000] et [Routier & al., 2001]. Cela permet de découpler l'agent de son savoir-faire.

Un agent va jouer des rôles grâce aux compétences qu'il possède ou désire posséder. Les structures de groupe et de rôle, ne sont pas statiques et évoluent dans le temps. Il en est de même pour la notion de compétence. Un agent peut acquérir ou perdre une compétence. Certains modules de compétences peuvent être modifiés dynamiquement. Nous le verrons dans la partie suivante. Ainsi, « la puissance des structures d'organisations vient de la capacité des agents à se joindre à de tels groupes et à y acquérir de nouvelles aptitudes qu'ils n'auraient pas obtenues sinon et qu'ils utiliseront dans d'autres groupes » [Gutknecht & Ferber, 1998c]. Dans Aalaadin, un agent peut donc faire partie d'un groupe s'il possède des aptitudes ou compétences particulières. S'il désire adhérer à un groupe, il devra s'assurer d'avoir les compétences requises. C'est parce qu'il possède des compétences, qu'il peut jouer un rôle dans un groupe. L'approche de J. Ferber semble dire qu'un agent se joint à un groupe et donc acquiert la connaissance. Notre approche est inversée, c'est parce que notre agent possède un certain nombre de compétences qu'il pourra accomplir un rôle. Si ses compétences augmentent, l'agent pourra éventuellement endosser de nouveaux rôles et donc contribuera au changement de l'organisation. Une compétence peut être utilisée dans différents rôles.



Il y a évidemment plusieurs niveaux d'abstraction car une compétence d'un niveau *n* peut être l'agrégation de plusieurs compétences de niveau *n-1*. Une compétence dite « atomique » pourrait être la faculté d'additionner 1+1 ou la faculté de bouger un membre de son corps. Une compétence de plus haut niveau pourrait être de savoir réaliser toute forme d'addition ou de déplacer tout son corps en actionnant ses membres de façon synchronisée. Une compétence de plus haut niveau encore serait de tenir une comptabilité entière ou de réaliser une chorégraphie.

L'intrication entre rôle et compétence est non triviale car l'accroissement des compétences a une répercussion sur le ou les rôle(s) joué(s). Un agent qui ne sait que se déplacer ne va pas tenir le même rôle qu'un agent capable de se déplacer de façon coordonnée et harmonieuse dans le cadre d'une chorégraphie par exemple.



Le changement de rôle ou la prise en charge d'un nouveau rôle changera la structure organisationnelle de l'agent. On peut considérer que l'agent lui-même, lié à ses rôles, ses tâches et ses compétences, est auto-organisé. Cet agent sera plongé avec d'autres agents dans un environnement, le tout constituant un autre système auto-organisé à plus grande échelle. Il y a une notion de réorganisation en continu du système complexe dans lequel est plongé l'agent. Théoriquement, il est envisageable que la modification et la restructuration de la connaissance d'un agent soit à la base de la dynamique d'un système.

Dans cette version du modèle YAMAM, nous ne distinguons pas les compétences réactives des compétences cognitives. Un agent, réactif ou cognitif, peut donc utiliser une compétence de la même manière. Nous travaillons cependant sur un système de mémoire, propre à l'agent cognitif qui permettrait à un agent cognitif d'exploiter différemment ses compétences en fonction de son expérience. Nous étudions un système de rétro-action sur la connaissance : un agent va instancier une compétence brute « pré-compilée » et cette compétence sera raffinée, modifiée et enrichie par les actions de l'agent, par ses visions, pas ses croyances, d'une manière générale, par ce qui constitue sa connaissance… Ainsi deux agents ayant la même connaissance au départ n'auront pas forcément les mêmes comportements. On peut voir cela comme un paramétrage de la compétence. Si un agent cognitif peut céder une compétence brute, il ne peut en aucun cas la céder avec l'expérience liée à son utilisation. Cette expérience (son savoir et son savoir-faire) lui est propre. Elle est structurée en fonction des caractéristiques de l'agent, de ses croyances, de ses craintes, de ses envies, et de façon générale de son état mental. Un agent ne possède pas la même structure interne qu'un autre agent. Cette recherche en étant à ses débuts, nous n'en parlerons pas davantage.

IV.2.4. Des systèmes régies par l'énergie

Dans notre étude sur les SMA et les théories de l'auto-organisation et du chaos (Chapitres 1 et 2), nous nous sommes posés la question sur ce qui pouvait intervenir chez l'agent dans son choix d'un but ou d'un rôle. Les entités constituant un système complexe sont en interaction et socialement dépendantes. Les actions des agents peuvent être plus ou moins altruistes mais il semble qu'elles reposent toutes sur une notion universelle de recherche d'énergie.

> *L'énergie est une notion abstraite. Pour le physicien, elle représente la différence d'état dans la transformation physique d'un corps (par exemple : l'eau chaude a gagné de l'énergie par rapport à l'eau froide). Elle permet d'évaluer cette transformation et de comparer les transformations entre elles. L'énergie peut se transmettre d'un corps à l'autre (par exemple : un choc entre deux billes). Les formes communes d'énergie sont l'énergie mécanique (potentielle et cinétique), l'énergie thermique, l'énergie électromagnétique (électricité, lumière), l'énergie nucléaire, et l'énergie chimique. L'énergie peut se diviser et se transformer mais* ***la somme des énergies transformées est toujours égale à l'énergie initiale*** *(premier principe de thermodynamique). Dans l'exemple d'un moteur électrique, l'énergie électromagnétique se transforme en énergie mécanique (mouvement) ET en énergie thermique (chaleur). Malheureusement, on n'arrive JAMAIS à récupérer toute l'énergie transformée, une partie se dissipe en chaleur dans la "nature" (deuxième principe de thermodynamique : l'entropie). Dans l'exemple d'un circuit électronique, l'effet joule est dû à la transformation de l'électricité (énergie électrique) en chaleur (énergie calorifique).*

Tout être vivant cherche à absorber une forme énergie pour la maintenir au moins un seuil critique. On retrouve aussi cette notion d'échange d'énergie dans le registre de la spiritualité [Redfield, 1994] – chaque individu cherche à capter de l'énergie au détriment d'un autre. La méthode pour y parvenir dépend de sa psychologie et de son comportement. D'une manière générale, cette énergie existe dans la nature et peut prendre différentes formes et passer d'un état à un autre.

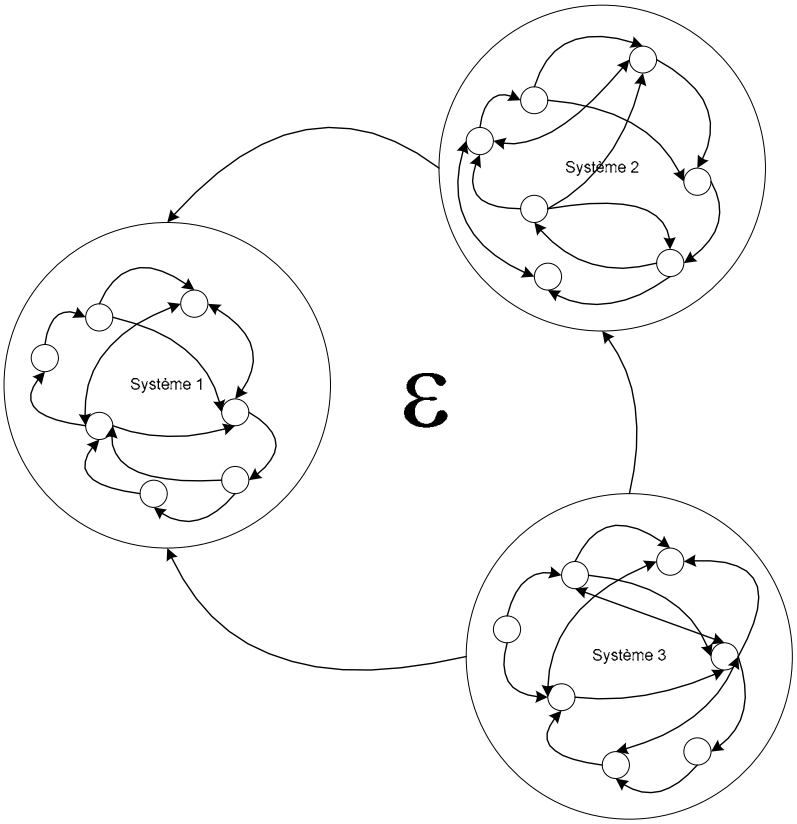
Dans la nature, l'insecte (ou l'animat) cherche à rester simplement dans son domaine de viabilité. C'est un agent réactif qui n'a probablement pas conscience d'exister. L'absence de conscience de soi est liée à l'absence de notion d'apprentissage et de sentiment. Par exemple, ne ressentant pas le sentiment de peur, ils ne craignent pas la mort et agissent au final dans le seul intérêt de la colonie, sans même en avoir notion. Les insectes cherchent à assurer la survie de la colonie et par extension la leur et celle de leur progéniture. Le substrat d'échange est l'énergie nutritive. Ils cherchent ensuite à conserver leur énergie vitale individuelle en la maintenant au dessus d'un seuil critique. Dans cet état, les insectes peuvent ensuite se spécialiser et adopter un rôle en adéquation avec leur morphologie. Certaines fourmis fourragent et d'autres partent en guerre. Les premières nourrissent les secondes et les secondes défendent les premières. Dans la mesure où la nourriture est en quantité suffisante, la colonie peut se développer. Dans le cas contraire, le nombre d'insecte décroît pour stabiliser la démographie en fonction des ressources disponibles. Cela est valable à tous les niveaux de la chaîne alimentaire.

Un animal plus évolué comme le rat ou le chien a un fonctionnement plus complexe. Il est plus cognitif et possède une faculté d'apprentissage. Ses sensations sont la résultante de ses perceptions et il se forge des sentiments qui sont le résultat des sensations. Il se rend donc compte que certaines actions lui provoquent des sensations agréables et d'autres non. Il va donc chercher à maximiser son plaisir et minimiser ses peines. Comme l'insecte, il recherche

en premier lieu à se maintenir dans une zone de viabilité et en deuxième lieu à maximiser son plaisir. Le substrat d'échange est toujours l'énergie nutritive mais un second peut être une forme d'énergie apportée par le « bien être ». Le bonheur des uns fait le malheur des autres.

L'être humain, stade ultime dans l'évolution des espèces, possède une conscience de lui-même et possède une notion du temps. Cette conscience lui permet d'apprendre et de planifier ses actions. Comme les animaux évolués, il va chercher à se maintenir dans une zone de viabilité, à maximiser son plaisir mais également à maximiser son intérêt personnel en planifiant des actions qui peuvent le servir à court, moyen ou long terme. De nature cognitive, il peut rechercher aussi une autre forme d'énergie (la satisfaction intellectuelle ou spirituelle).

L'état d'un système (en développement, en équilibre et en décroissance) dépend de sa quantité d'énergie. Cette énergie circule dans le système et prend différentes formes. Dans un état d'équilibre du système, l'énergie de ne perd pas et passe d'un individu à un autre. Lorsque le système entier perd son équilibre, l'énergie passe dans un autre système, indépendant du premier. Au contraire, si de l'énergie est gagnée dans un système, elle a été soustraite d'un autre système.



Flux d'énergie inter et intra systèmes

IV.2.5. Application de la notion d'énergie à notre modèle d'organisation

Nous souhaitons intégrer dans notre modèle YAMAM, un système natif permettant d'auto-organiser la plate-forme. Nous avons deux alternatives :

- une approche mathématique ou algorithmique visant à utiliser une fonction calculable pour trouver une solution optimale. Cette fonction globale doit prendre en compte plusieurs contraintes :
    1. maximiser la multiplication des agents ce qui tend à améliorer la robustesse (principe de redondance) et la rapidité du traitement (parallélisme)

2. maximiser la spécialisation des agents, ce qui tend à améliorer leurs performances
3. minimiser les conflits et le flux de messages de négociation échangé ce qui permet d'accroître la rapidité de la simulation
4. restreindre chaque agent dans son domaine de viabilité (niveau d'énergie).

Cette fonction doit être évaluée en temps réel et doit pouvoir échapper à des minima locaux. Pour cela, des techniques du type du recuit simulé peuvent être utilisées. Cette approche est centralisée et doit intégrer un « super-agent » qui pilote les autres. Elle ne va pas du tout dans le sens de la distribution totale des agents et ne leur laisse pas d'autonomie.

- une approche systémique, qui s'oppose radicalement à la première approche, et qui postule que les interactions de nos agents peuvent apporter une solution, qui cependant n'est pas optimale sur le facteur temps. Cette approche, que nous adopterons, intègre les perturbations de l'environnement et se base sur une émergence d'auto-organisation des agents. La solution provient de la dynamique intrinsèque du système.

Un système multi-agents peut parfois faire émerger des propriétés étonnantes mais ce n'est pas pour autant que les agents s'auto-organisent entre eux. L'exemple des insectes qui trient des brindilles est un cas typique. Il n'y a aucune communication entre les agents donc aucun échange. Les agents ne se coordonnent pas particulièrement mais les brindilles se retrouvent pourtant triées.

Nous recherchons une forme de coordination entre agents qui ne s'appuierait pas sur un quelconque algorithme. Nous considérons qu'un système multi-agents ne peut s'organiser librement sans un substrat, appelé ici *Energie*, qui relie les entités le constituant.

Si les SMA sont particulièrement bien adaptés pour simuler le fonctionnement d'une économie [Guessoum & Durand, 1996], nous allons voir que les principes économiques sont aussi adaptés pour assurer la régulation d'un SMA. L'échange de monnaie dans un système économique répond à la même dynamique que celle évoqué précédemment dans les différents systèmes.

Nous reprenons donc le principe de la transformation de l'énergie évoquée dans le précédent paragraphe mais appliqué à un seul substrat. La circulation de cette notion abstraite d'énergie est basée sur le principe de l'économie de marché. Le schéma de flux d'énergie inter et intra-systèmes présenté ci-dessus est tout à fait appliqué à la circulation de la monnaie. Un système peut représenter un foyer, une entreprise, une région ou un pays. De la monnaie est en libre échange entre les entreprises d'un même pays et les états entre eux peuvent s'échanger de la monnaie (devises). Nous allons utiliser ces principes monétaires pour concevoir le système de flux d'énergie dans notre modèle d'organisation YAMAM. Une plate-forme héberge des agents qui s'échangent de l'énergie. Comme des entreprises, des agents peuvent s'échanger de l'énergie entre plusieurs plates-formes.

Nous présentons dans les prochains paragraphe cette notion qui régie notre modèle et proposons un parallèle avec les théories de l'économie, notamment le monétarisme.

IV.2.5.1. Théorie de l'énergie basée sur le monétarisme

La première pensée monétariste est née avec Jean Bodin au XVI^{e} siècle. Il est notamment le premier économiste à avoir établi un lien entre l'importance de la masse monétaire et la hausse des prix. Cette idée fut reprise et formalisée au XX^{e} siècle par Irving Fisher, qui donna sa forme la plus aboutie à la théorie quantitative de la monnaie. Le courant

monétariste, animé par des néolibéraux, repose sur une interprétation particulière de l'équation de Fisher. Elle pose l'égalité entre la masse monétaire (M) multipliée par sa vitesse de circulation (V) et la masse des paiements correspondant au volume des transactions (T) effectué à un niveau de prix (P). C'est l'équation devenue fameuse en économie: MV = PT. Autrement dit, si la masse des paiements (PT) est égale à 100 et que la même pièce serve à effectuer deux échanges dans la journée (V = 2), il faudra une masse monétaire (M) de 50. Or, les monétaristes considèrent, d'une part, que la vitesse de circulation (V) est stable puisqu'elle dépend de variables psychosociologiques ou d'habitudes de paiement et d'autre part que, dans une économie qui utilise ses capacités de production à 100 %, le volume de production (T) est donné. Ainsi, toute augmentation de la masse monétaire (M) se traduit directement par une augmentation du niveau des prix (P) qui n'est autre qu'un indicateur d'inflation. Ce fonctionnement est repris en partie dans notre modèle pour quantifier l'énergie échangée et pour fixer le prix d'une compétence ou d'une tâche sous-traitée.

IV.2.5.2. Les conséquences de l'analyse monétariste

L'école monétariste, représentée par Milton Friedman [Friedman, 1968] et [Friedman, 1995], est soucieuse de limiter au minimum l'intervention de l'État et rejette les politiques fiscale et budgétaire keynésiennes en prônant l'usage de la politique monétaire comme instrument d'une croissance économique stable. En effet, dans les années 1960 et 1970, l'évolution des économies a fait apparaître un chômage important, malgré la hausse des prix (stagflation), phénomène qui paraissait incompatible avec les explications keynésiennes. Un certain nombre d'économistes se sont alors rabattus sur la théorie quantitative de la monnaie, qui présente l'avantage de donner une explication simple à la hausse des prix. Selon les monétaristes, l'inflation provient de l'intervention des autorités, qui créent trop de monnaie; aussi l'État doit-il renoncer à ses interventions (la caractéristique principale des monétaristes est une foi solide dans le postulat que le marché, grâce aux informations fournies par le système des prix, permet la meilleure allocation des ressources). Ainsi, le chômage existant serait « naturel » (correspondant à un minimum inéliminable dans une économie) ou même « volontaire », puisque tous ceux qui veulent travailler au salaire courant peuvent le faire. À cette thèse, développée principalement par Friedman, les « nouveaux classiques » (R.E. Lucas) ont ajouté l'idée de l'impuissance des autorités monétaires, car toute création de monnaie supplémentaire annoncée par les autorités sera annulée par les comportements « rationnels » des agents qui anticiperont l'inflation inévitable. La macroéconomie aboutirait ainsi à la négation de toute possibilité de politique économique.

Nous avons utilisé ce principe dans la conception du système d'énergie de YAMAM essentiellement pour sa simplicité de mise en œuvre dans la mesure ou le noyau (l'état) n'intervient pas dans la régulation de l'échange d'énergie. Nous pouvons alors simplement laisser les agents négocier leurs rôles et leurs acquisitions de compétences. Avec ce point de vue « idéal », dénué de toute considération sociale, nous pouvons laisser des agents mourir sans aucun état d'âme lorsqu'ils n'ont plus d'énergie. Cruel mais efficace !

IV.2.5.3. Principe de la fiscalité

La fiscalité répond, en première approche, à des objectifs tels que la redistribution et la correction des imperfections du marché. Sachant que toute nouvelle mesure fiscale s'accompagne d'incitations ou de désincitations, celle-ci s'appliquant sur les principales grandeurs économiques, le problème de la théorie économique est d'identifier et d'analyser les imperfections du marché, et d'y appliquer des mesures correctrices, c'est-à-dire principalement de réduire l'inégalité des revenus et de corriger les externalités.

Nous avons repris partiellement cette notion en imposant les agents. L'objectif n'est pas d'aider d'autres agents en difficulté mais de financer des services communs comme la recherche de compétence ou de rôle (par le noyau). Certain noyau impose plus les agents locaux que d'autres. Cela permet d'équilibrer la charge dans la mesure où un agent trop imposé va spontanément décider de migrer sur une autre plate-forme ayant une politique fiscale plus souple.

IV.2.6. Réutilisation du concept dans notre modèle

La notion d'énergie est centrale dans YAMAM et constitue la véritable valeur ajoutée du modèle. Cela permet de maintenir le système dans un état d'équilibre dynamique assurant une faculté d'adaptation à un changement d'environnement puisqu'il se restructure en permanence. Un agent peut mourir sans déstabiliser le système ou intégrer un environnement sans opération particulière sur la plate-forme. La plasticité induite par cet échange d'énergie offre beaucoup d'avantages notamment sur la stabilité du système assurée par cet équilibre dynamique entre redondance et spécialisation des agents.

Selon Adam Smith, philosophe et économiste, les actions des individus sont motivées par la recherche de leur intérêt personnel. Le moyen d'enrichir la nation est de laisser les individus s'enrichir, car en oeuvrant pour leur intérêt personnel, ils enrichissent la nation entière (c'est la notion de la « main invisible »). Autrement dit, en cherchant à satisfaire son objectif personnel, un agent va contribuer au bon fonctionnement du système entier. Jean-Baptiste Say, présenté comme un disciple de Smith, considère que les produits s'échangent comme les produits et que l'offre crée sa propre demande. Il étend aussi la notion de production aux services et non seulement aux biens matériels.

Dans notre modèle YAMAM, le but de l'agent est de trouver des rôles et les mener à bien pour maintenir ou accroître son niveau d'énergie. Survivre en trouvant des rôles est donc sa priorité. Il choisit de préférence ceux exploitant au maximum ses compétences, ce qui lui rapporte plus d'énergie. Pour jouer des rôles, il lui faut posséder et/ou trouver des compétences. Le but de l'agent se résume donc à chercher des rôles qui exploitent pleinement ses compétences afin de maximiser la fonction Energie. Lorsque les rôles les plus rémunérateurs en énergie sont réalisés, les agents cherchent toujours à augmenter leur capital d'énergie en prenant en charge des petits rôles moins rémunérateurs. L'aphorisme de Spinoza « La nature a horreur du vide » s'applique typiquement dans ce cas dans la mesure où un rôle ne reste jamais longtemps en attente d'un agent.

L'économiste Léon Walras explique que si les conditions de la concurrence pure et parfaire sont réunies, c'est-à-dire si l'ordre naturel est respecté, l'économie se maintient en état d'équilibre automatiquement. Les caractéristiques de cet équilibre sont exposées par Walras puis par Vilfredo Pareto dans son manuel d'économie politique. Pour que ces conditions de la concurrence pure et parfaite soient réunies, il est nécessaire qu'il y ait :

- une atomicité du marché, c'est-à-dire une multitude d'offreurs et de demandeurs afin qu'aucun ne puisse avoir un impact sur les prix
- une homogénéité des produits ; les produits doivent être comparables afin que la concurrence joue uniquement sur les prix
- une fluidité du marché ; à tout moment, chacun peut entrer ou sortir du marché
- une transparence du marché ; toutes les informations doivent être parfaitement disponibles afin de pouvoir faire jouer la concurrence.

Ces quatre points essentiels sont repris dans notre modèle d'organisation multi-agents.

Par hypothèse, les agents économiques sont des « homo oeconomicus », c'est-à-dire des êtres rationnels qui cherchent à maximiser leur profit s'ils sont producteurs et à maximiser leur satisfaction et minimiser leur travail s'ils sont consommateurs. Ils sont capables de décider en connaissance de cause et peuvent prévoir les conséquences de chacune de leurs décisions. Pour l'individu, la rationalité suppose dès lors une conscience objective de son intérêt et une capacité à juger et à anticiper. Herbert Simon mettra en cause cette rationalité parfaite de l'agent économique en disant que les individus ne recherchent pas toujours une solution optimale mais s'arrêtent consciemment ou non à la première solution satisfaisante qu'ils découvrent.

Selon Pareto, sur les trois marchés principaux, le marché des biens et des services, le marché du travail et le marché du capital se confrontent une offre et une demande qui sont le résultat de l'agrégation des demandes et des offres individuelles. Sur chaque marché, l'offre est une fonction croissante du prix et la demande est une fonction décroissante. La libre variation des prix assure l'équilibre. Si une modification intervient sur l'un des marchés, elle se répercute immédiatement sur les autres marchés qui compensent cette variation de façon à maintenir l'équilibre. La fixation des prix s'effectue par tâtonnement. Des ajustements successifs permettent de fixer un prix d'équilibre, égalisant ainsi la quantité offerte et la quantité demandée sur chaque marché. L'équilibre général est ainsi atteint. Un déséquilibre ne peut être ponctuel. La loi de Walras stipule que la demande totale (somme des demandes de chaque marché particulier) est forcément égale à l'offre globale (somme des offres de chaque marché particulier) : un déséquilibre sur un marché suppose donc un déséquilibre symétrique sur un autre marché. Walras et Pareto montrent que l'équilibre général est mathématiquement possible. Le problème est de savoir s'il existe un système de prix qui égalise l'offre et la demande sur tous les micro-marchés en même temps. On se retrouve donc avec un système d'équations (équation d'offre, équation de demande et équation d'équilibre entre l'offre et la demande) dont les inconnues sont les quantités et les prix. Walras explique qu'il existe autant d'équations que d'inconnues et donc que le système peut avoir une solution. L'équilibre général est ce fait possible. Un déséquilibre ne peut perdurer que si les conditions de la concurrence pure et parfaite ne sont pas réunies : une intervention extérieure au marché comme celle de l'Etat est condamnable car elle altère l'ordre naturel et perturbe donc le fonctionnement autorégulateur du marché. La notion d'optimum économique a été définie par Pareto. Une situation économique est optimale au sens de Pareto lorsque l'on ne peut pas améliorer la satisfaction d'un individu sans détériorer celle d'au moins un autre individu dans une proportion égale ou supérieure.

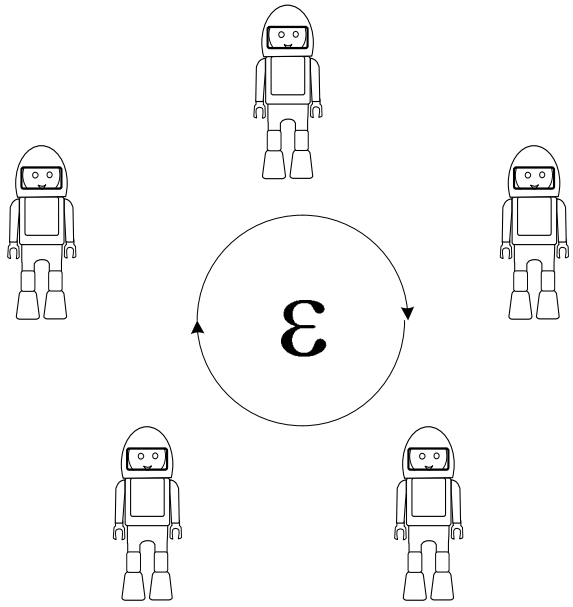
Malgré un principe d'équilibre dynamique, l'économie répond aussi à des fonctionnements chaotiques. Il existe aussi des théories des crises et des cycles. Selon ces théories, l'économie subirait des fluctuations qui reviennent avec une certaine régularité et une certaine périodicité. Les cycles longs sont appelés « cycles de Kondratieff[31] ». Un cycle mineur ou « cycle de Kitchin » est un cycle court d'une durée approximative de 40 mois. Ce cycle est de faible amplitude. Selon Schumpeter, la croissance économique s'explique essentiellement par l'action du progrès technique et c'est l'irrégularité de celle-ci qui expliquerait les irrégularités de la croissance.

Le modèle d'organisation de SMA peut théoriquement subir de mêmes fonctionnements chaotiques. Nous n'en avons pas constaté pour le moment mais il est probable qu'un SMA construit à grande échelle puisse se comporter de la sorte. A noter que

[31] Nicolas Kondratieff, 1892-1930

les automates cellulaires peuvent parfois se comporter curieusement. Nous présentons en annexe quelques exemples de stabilisation d'automates cellulaires.

Cette énergie intervient à chaque instant. Lorsqu'un agent accompli un rôle, il gagne de l'énergie. S'il délègue des tâches de son rôle, il cède de l'énergie à l'agent sous-traitant. S'il achète une compétence, il cède de l'énergie. L'utilisation de ressources lui fait aussi perdre de l'énergie. Enfin la plate-forme l'impose (percepteur) et lui extorque de l'énergie régulièrement. La plate-forme réinjecte cette énergie dans des ressources que les agents pourront consommer pour l'exécution de leurs rôles. Le fonctionnement est similaire à l'état qui nous prélève des impôts pour concevoir des infrastructures utiles à tous comme un réseau routier. Dans tous les cas, la somme totale d'énergie circulant dans une plate-forme est une constante. Les dépenses énergétiques dans la nature ne se perdent pas non plus. Un robot ménager par exemple transforme une énergie électrique en une énergie mécanique. Nous verrons dans le prochain chapitre comment est organisé ce système d'échange d'énergie entre les différentes entités du logiciel.



L'agent peut convoiter un rôle lui rapportant plus d'énergie que son rôle actuel. Si ce rôle requiert des compétences qu'il ne possède pas, il peut décider d'un investissement. Son comportement pourrait s'apparenter à une recherche d'emploi : tant qu'il a des compétences inutilisées il cherche des rôles qui les utilisent, lorsqu'il convoite un rôle mais qu'il lui manque une compétence, il entreprend d'acheter cette compétence. Pour faire un parallèle, lorsqu'un agent est en recherche d'emploi il s'inscrit à l'ANPE puis consulte les annonces, il sera prévenu de toute nouvelle annonce jusqu'à ce qu'il soit employé à son maximum de compétences.

Les compétences sont recherchées en premier lieu dans les accointances de l'agent, puis en sous-traitant au noyau. Si le noyau échoue dans sa recherche, il peut en dernier lieu lancer une requête sur d'autres noyaux. Plus la demande est difficile à satisfaire, plus l'énergie requise est importante. L'agent peut donc renoncer à un rôle si l'investissement est trop important. Si l'agent trouve toutes les compétences requises, il endosse le rôle, sinon il en cherche un autre jusqu'à épuiser les rôles proposés.

Le principe de fonctionnement étant celui de l'économie de marché, un agent possédant quelques compétences peut prendre des rôles en charge et ainsi les "monnayer" pour augmenter son énergie (son capital). En possédant plus d'énergie, il peut décider d'investir dans de nouvelles compétences ou de sous-traiter à des agents ayant déjà les compétences. L'acquisition de nouvelles compétences est pour lui un investissement en énergie donc il ne choisira d'investir que lorsqu'il sera certain de pouvoir franchir son seuil de rentabilité en proposant ses services. Un agent n'est donc pas systématiquement tenté d'apprendre sans arrêt sans travailler. Il doit rentabiliser sa connaissance

Les agents ont le but commun de maximiser leur fonction énergie. Pour cette raison, un agent va souvent entrer en collaboration ou en conflit avec d'autres agents. On retrouve partout la loi de l'offre et la demande. Les grands principes de cet échange d'énergie sont :

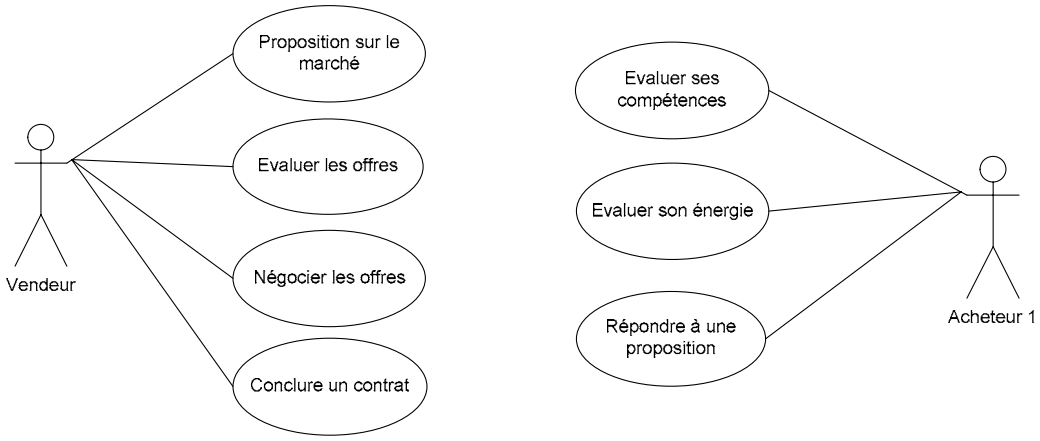
- Les agents les plus faibles sont amenés à disparaître.
- Les agents les plus riches vont souvent engendrer davantage de richesse.
- L'énergie est gagnée par un agent au détriment d'un autre.
- Un agent possédant peu de richesse peut en gagner s'il propose aux autres agents un service de qualité.
- Un agent peut décider de dépenser de l'énergie en investissant dans une compétence si c'est dans le but d'en gagner plus ensuite.
- L'agent noyau joue un peu le rôle de l'état qui collecte des impôts pour assurer le fonctionnement du système global.
- L'agent qui juge payer une redevance trop forte peut décider de migrer vers une plate-forme où il serait moins taxé et bénéficierait de meilleurs services.
- Chaque agent a un coût de fonctionnement même à ne rien faire.

Plus la machine est puissante, plus elle pourra gérer un grand nombre de transactions. Plus les services des agents seront valorisés, plus le noyau recevra d'énergie en taxant les transactions. Le noyau doit cependant assurer la qualité de ses services au risque de voir les agents migrer sur une plate-forme concurrente. Ce fonctionnement engendre automatiquement la mort des agents "inutiles". C'est sur ce principe dynamique qu'est basé le fonctionnement de YAMAM.

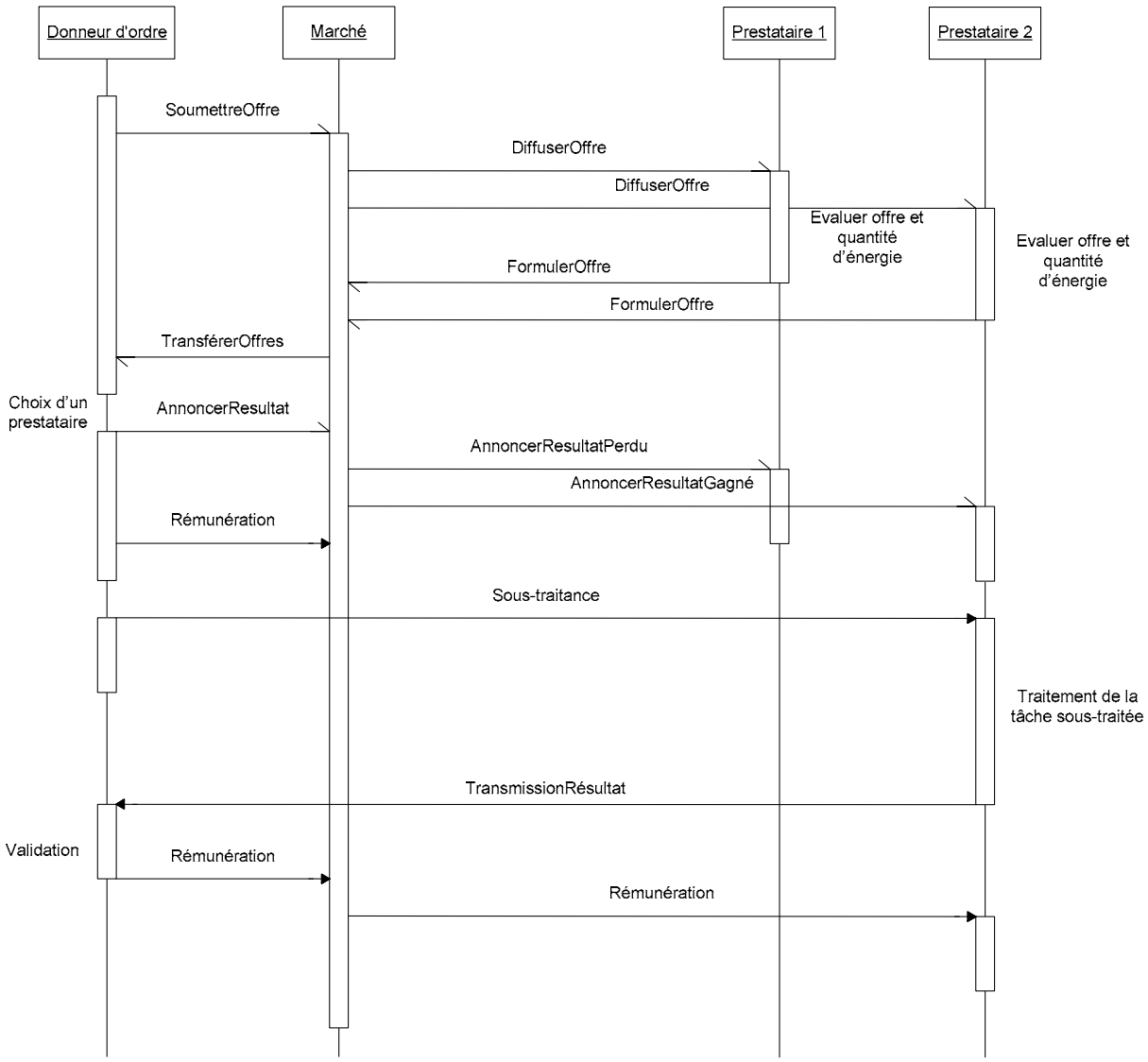
L'intérêt d'une telle approche est d'équilibrer la plate-forme et de permettre à travers ces agents une autorégulation du système.

IV.2.7. La coordination entre les agents

Les agents ont besoin de communiquer et de se coordonner. Lorsqu'un rôle est affiché sur le marché, les agents compétents vont faire des offres pour répondre à celui qui a soumis le rôle (le « vendeur »), de la même façon que se déroule un appel d'offre classique. Le vendeur va sélectionner les cinq meilleures propositions. S'ensuit alors un « marchandage » durant lequel chacun des agents clients va tenter de remporter le contrat. Il peut être amené à baisser ses coûts sans jamais descendre en deçà de son coût de revient.

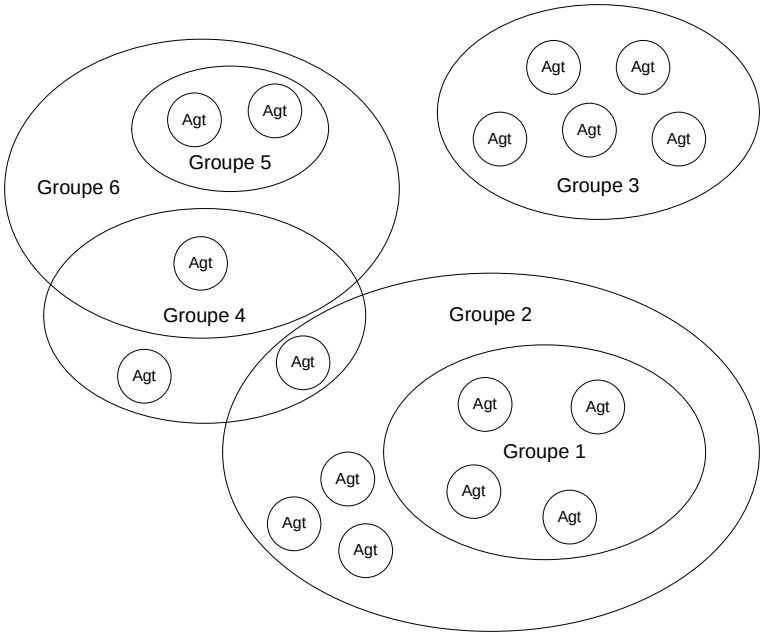


Le protocole de négociation est dérivé du Contract Net Protocol [Smith, 1980]. Ce protocole était à l'origine un outil pour la résolution de problèmes distribués. Un contrat est défini comme un ensemble de tâches à accomplir, ce qui s'apparente dans YAMAM à un rôle. Des agents font des offres pour réaliser les tâches proposées par d'autres agents. Les offres sont évaluées selon certains critères et le contrat est conclu. Il y a eu plusieurs implémentations de ce protocole de haut niveau dans des domaines divers [Yang & al., 1998].



IV.2.8. Les groupes d'agents

Nous utilisons une terminologie identique à celle d'Aalaadin en ce qui concerne le groupe. Un groupe est une simple agrégation d'agents. Cette notion de groupe présente dans le modèle Aalaadin de Ferber et physiquement implémentée dans MadKit, est seulement implicite dans notre modèle. Les agents ignorent faire partie d'un groupe contrairement à Aalaadin où l'agent intègre sciemment un groupe. Un groupe d'agent peut être vu comme des agents jouant des rôles identiques ou ayant des buts ou des compétences en commun. Différents groupes peuvent se recouper ou s'inclurent. Un agent peut donc être membre d'un ou plusieurs groupes.



Dans la plate-forme, la notion de groupe est présente uniquement dans le noyau[32] pour analyser les organisations des agents (reconformations morphologiques des structures de groupes). Le noyau peut décider de créer un groupe lorsqu'il juge que deux agents ou plus ont des caractéristiques, des rôles ou des buts en commun. Les groupes sont de différents niveaux. Par exemple un agent peut posséder une compétence commune avec un autre agent, la notion de groupe ne sera pas très significative. En revanche si des agents possèdent un rôle identique (donc tout un ensemble de compétences en commun), le groupe va être d'une importance plus élevée.

Un agent rencontrant un autre agent jouant un rôle similaire va l'ajouter dans son réseau d'accointances puisqu'il est probable qu'ils suivent des buts identiques et donc possèdent et recherchent des compétences semblables. On peut y voir là une notion de groupe implicite. Un agent évolue dans un environnement. De part le caractère intrinsèquement dynamique d'un SMA, des organisations se forment et évoluent dans le temps. On voit apparaître dans ces organisations certains regroupements d'agents qu'il est intéressant de prendre en compte pour mieux appréhender le comportement des agents. Le noyau peut observer par exemple une émergence d'un comportement de grégarisme ou rendre compte d'une situation chaotique.

[32] Le noyau est considéré dans YAMAM comme une entité indépendante du modèle effectuant des observations. C'est implémenté différemment sur la plate-forme. C'est un ensemble de classe qui gère le cycle de vie des agents, les recherches de rôles et de compétences, la place de marché etc. On peut lui adjoindre des rôles plus spécifiques comme l'analyse du comportement du système. C'est en quelque sorte un agent superviseur passif.

L'intérêt de cette représentation interne des groupes est d'optimiser les ressources et les compétences des agents. Lorsqu'un agent cherchera par exemple à déléguer une tâche, l'agent noyau saura quel agent possède un rôle ou une tâche identique. Cela permettra une mise en relation entre agents plus facile. Un agent ne connaît pas nécessairement les agents exécutant des rôles identiques. C'est un petit peu comme des communautés sur Internet qui partagent les mêmes passions ou les mêmes professions. C'est toujours enrichissant d'être en contact avec d'autres personnes qui peuvent éventuellement répondre aux questions que nous nous posons. A titre d'exemple, les newsgroups comme *comp.ai.neural-nets* regroupent des individus s'intéressant aux réseaux de neurones. En se groupant en communautés, on partage plus facilement ses connaissances et donc le groupe constitué de ces différents individus est globalement plus compétitif qu'un individu isolé et chaque individu profite des connaissances du groupe. C'est également ce qui se passe dans le cadre de syndicats ou de diverses associations.

Un agent peut faire une requête au noyau s'il désire connaître ce qu'il a en commun avec un autre agent. Cela peut lui être utile pour affiner l'organisation de son réseau d'accointances et pour se faire une représentation personnelle de son environnement (dans le cas d'un agent cognitif). Cette représentation interne de l'organisation dans le noyau est aussi utile à des fins d'analyse morphologique des structures d'organisation émergentes.

IV.2.9. Les objets et les ressources

Pour réaliser une action, un agent dispose de ressources et peut manipuler des objets de son environnement. Nous n'imposons aucune contrainte sur ces deux notions de façon à laisser une certaine liberté au développeur.

IV.2.10. L'environnement

L'environnement est une notion conceptuelle permettant aux agents de se repérer. Nous voyons l'environnement comme une vaste zone d'échange. Plus particulièrement, un environnement réactif est une classe regroupant un certain nombre de cellules que les agents utilisent comme parcours élémentaire pour se déplacer d'un endroit à un autre. L'environnement se réduit à une carte liant des cellules adjacentes, comme un automate cellulaire. Nous gérons deux ou trois dimensions ce qui permet de simuler par exemple un trafic routier ou des galeries souterraines de colonies d'insectes sur plusieurs niveaux [Dorigo & Gambardella, 1997]. L'environnement offre la possibilité d'une communication indirecte, ce qui permet de communiquer d'un agent à un autre par l'intermédiaire d'un autre agent ou d'un autre objet. Les ressources n'étant jamais inépuisables, Phœnix gère des classes génériques permettant de manipuler des ressources comme de l'eau, de la nourriture, des temps processeurs etc. Des agents peuvent entrer en concurrence pour utiliser des ressources.

L'environnement peut être explicite comme dans une simulation d'un système biologique. Il peut être implicite comme dans la construction d'agent effectuant des taches sur un réseau informatique. Un agent peut scruter l'arrivée d'un email, un autre peut optimiser le matériel via le protocole SNMP, un autre peut analyser les trames circulant d'un point à un autre etc. Chaque agent joue un ou plusieurs rôles mais l'environnement n'est pas décrit explicitement, c'est le réseau lui-même. Nous laissons donc cette notion très ouverte et n'imposons rien au développeur.

Dans le cas d'un environnement 2D, on utilise un système de coordonnées cartésiennes. L'environnement est découpé en cases. Une ressource ou un agent est localisé dans une ou plusieurs cases. Certaines cases peuvent être inaccessibles, ou peuvent posséder une contrainte (par exemple, deux agents ne peuvent pas être simultanément sur un pont ou

dans un couloir). L'environnement peut avoir une incidence sur l'agent car ce dernier perçoit l'environnement au moyen de trois primitives :

- récupérer le champ de vision : retourne l'ensemble des cases que peut voir l'agent.
- récupérer l'ensemble des cases accessibles : permet à l'agent de savoir sur quelles cases il peut se déplacer
- déplacer l'agent : permet à l'agent de se mouvoir.

L'agent peut aussi utiliser l'environnement comme moyen de communication indirect. C'est le cas des fourmis qui déposent des phéromones.

IV.3. Méthodologies applicables au modèle YAMAM

Nous avons vu dans le chapitre précédent différentes méthodologies permettant l'analyse des SMA. De la même façon que Aalaadin présente un triplet AGR, YAMAM est un modèle organisationnel basé sur quatre concepts (Agent, Rôle, Tâche, Compétence). Nous retrouvons le concept d'agent et de rôle dans les deux modèles mais YAMAM ne fait pas apparaître le niveau organisationnel (les groupes) dans le modèle. Les regroupements d'agents sont considérés comme implicites. Il est donc difficile d'appliquer intégralement Aalaadin avec notre modèle d'organisation mais cette méthodologie présente néanmoins beaucoup d'intérêt.

De par la nature de notre modèle, toutes les méthodologies ne lui sont pas applicables pour l'analyse préalable à la conception d'un SMA. Notre modèle propose de découper les rôles en tâches. Un agent est dit téléonomique s'il est dirigé par des buts. L'exécution d'un rôle peut être un but en soi mais un but global peut requérir l'exécution de plusieurs rôles. Cette distinction dépend du niveau de granularité de l'analyse. Dans le cas où un rôle est un but, les tâches requises par le rôle peuvent être considérées comme des sous-buts. De ce point de vue, les méthodes telles que MaSE ou VOYELLES sont applicables. Dans MaSE par exemple, on peut assimiler les tâches de YAMAM comme des sous-buts. Néanmoins l'exécution d'une tâche est indispensable pour la tenue du rôle, ce qui n'est pas toujours le cas d'un sous-but.

La méthode SCALA présente aussi beaucoup d'intérêt pour notre modèle dans la mesure où elle intègre des notions de spécialisation des agents et d'allocation dynamique de tâches. C'est finalement celle que nous avons retenue pour l'ensemble de ces qualités et c'est celle qui s'est avérée la plus adaptée à notre modèle YAMAM.

IV.4. Conclusion

Le modèle YAMAM offre une certaine souplesse de fonctionnement en autorisant une organisation basée sur le découplage des rôles en tâches, ainsi que sur le découplage des agents et de leurs compétences. Ce découplage permet au modèle d'être utilisé dans bon nombre de logiciels comme des systèmes de groupware, d'ordonnancement ou de simulation. Il s'agit donc d'un modèle organisationnel générique qui présente une articulation de ces quatre entités. Un autre point fort du modèle est l'utilisation d'un système d'échange d'énergie entre toutes les entités composant le système logiciel. Ce substrat permet au système de s'auto-organiser en éliminant les agent les plus faibles et en favorisant l'équilibre dynamique entre la redondance des agents et leur spécialisation. Ce concept fort suffit à distinguer YAMAM des autres modèles existants. Nous n'avons pas prouvé théoriquement les apports de notre modèle mais cela pourrait faire l'objet d'une prochaine étude en laboratoire.

Un modèle n'étant qu'une construction abstraite, nous en proposons en revanche dans les prochains chapitres une réification en un middleware multi-agents que nous pourrons utiliser pour la construction d'applications distribuées.

Chapitre V. Etude comparative des plates-formes existantes

« On ne saurait mieux comparer l'absurdité des demi-mesures qu'à celle des mesures absolues » - Georges Courteline.

Résumé : ce chapitre présente la synthèse d'une étude comparative de plusieurs plates-formes multi-agents universitaires ou commerciales. Parmi l'offre très importante, nous avons retenu celles qui présentent le plus d'intérêt afin d'en extraire les caractéristiques les plus significatives.

V.1. Avant-Propos

Il existe de nombreux outils pour faciliter le développement de Systèmes Multi-Agents. Ces différents outils sont de nature et de complexité variable. La comparaison des plates-formes SMA est assez peu décrite dans la littérature scientifique [Logan, 1999] et [Müller, 1998]. Il est en effet délicat de présenter objectivement un comparatif ne reposant que sur des critères sélectifs flous. Il est bien entendu possible de décider arbitrairement d'un ensemble de caractéristiques comparatives mais bien souvent l'intérêt d'une plate-forme particulière repose sur un critère qui n'est précisément pas commun à plusieurs plates-formes.

Il est ensuite possible de recenser des caractéristiques techniques comme l'emploi d'une technologie par rapport à une autre mais cela ne constitue pas nécessairement un critère de choix pour l'utilisateur final. De plus les critères de comparaison sont différents selon la culture du lecteur. Un chercheur ne va pas s'intéresser aux mêmes critères qu'un industriel. L'universitaire va vérifier que la plate-forme repose sur un modèle organisationnel ou sur une méthodologie permettant de spécifier une problématique pour aboutir à l'élaboration par itération d'une structure organisationnelle d'agent. Il s'intéressera ensuite à la modélisation proprement dite de la plate-forme. Des critères comme la célérité d'exécution, la documentation et le suivi des versions sont plus importants chez un industriel qui prête naturellement attention à la pérennité de ses applications informatiques. L'interopérabilité est également un argument fort dans le choix d'un outil informatique, plus encore que la portabilité. Le cheval de bataille de Sun et de Microsoft repose précisément sur cette distinction pour leurs environnements de développement, respectivement Java et .Net. Certains outils sont en outre plus adaptés pour la simulation de grandes quantités d'agents (Madkit, Swarm…) alors que d'autres sont au contraire plus adaptés pour la mise en œuvre de quelques agents cognitifs.

Les fonctions offertes par les outils logiciels peuvent être conçues et programmées de manières diverses. Afin d'assurer une certaine uniformisation des développements, il existe des standards internationaux comme [FIPA, 2002] qui proposent des spécifications permettant de construire des plates-formes selon un modèle similaire et de les faire communiquer selon des protocoles également similaires. Toutes les plates-formes ne respectent pas ces standards et toutes n'ont pas nécessairement de méthodologie de conception en amont des développements.

L'un des objectifs originels du groupe ASA, [ASA, 2002] était précisément de réaliser une étude comparative sur les plates-formes d'agents. Ce groupe de travail a proposé de classer les plates-formes SMA selon les critères suivants :

- *Caractéristiques générales* : il s'agit d'une fiche d'identité de la plate-forme comportant des informations sur l'état de l'outil, les auteurs, le type d'applications, la configuration logicielle et matérielle etc.
- *Modèles multi-agents* : c'est une description des concepts multi-agents disponibles sur la plate-forme comme les caractéristiques individuelles et sociales des agents, la structure organisationnelle, les modèles de coordination, la description de l'environnement etc.
- *Caractéristiques physiques*: c'est une description des caractéristiques physiques des systèmes multi-agents implantables au travers de la plate-forme.
- *Environnement de développement* : c'est la description de la méthodologie, du langage ou de l'API…
- *Environnement d'exécution* : il recense les outils disponibles, une description de la documentation, le support etc.

Nous avons passé en revue de nombreuses plates-formes SMA dans un rapport interne [Savall, 2001]. Ce rapport s'appuie sur un recensement d'outils présentés sur les sites [AgentLink, 2002], [AgentBuilder, 2002] et [AgentCities, 2002] et sur des rapports comme [Versteeg, 1998]. Ces descriptifs de plates-formes se divisent en deux types : les plates-formes commerciales et les plates-formes universitaires. Nous n'allons pas énumérer dans ce document toutes les plates-formes existantes mais simplement citer celles qui nous paraissent les plus intéressantes et sur lesquelles nous nous sommes appuyés pour la conception et le développement de notre propre outil.

V.2. Un standard international

Il existe aujourd'hui un consortium international fondé en 1996 destiné à produire des standards pour l'interopération d'agents logiciels hétérogènes et donc pour « normaliser » le développement sauvage des plates-formes SMA. Il s'agit de la FIPA (Foundation for Intelligent Physical Agents) [FIPA, 2002]. C'est probablement l'influence de KQML [Finin & al., 1997], langage d'interaction devenu norme de facto, qui a indirectement donné naissance à cette fondation. Par la combinaison d'actes de langages, de logique des prédicats et d'ontologies publiques, la FIPA cherche à offrir des moyens standardisés permettant d'interpréter les communications entre agents de manière à respecter leur sens initial, ce qui est bien plus ambitieux que XML, qui ne standardise que la structure syntaxique des documents. Afin d'atteindre ce but, le FIPA émet des standards couvrant :

- Les applications (applications nomades, agent de voyage personnel, applications de diffusion audiovisuelles, gestion de réseaux, assistant personnel…)
- Les architectures abstraites, définissant d'une manière générale les architectures d'agents
- Les langages d'interaction (ACL), les langages de contenu (comme SL, CCL, KIF ou RDF) et les protocoles d'interaction
- La gestion des agents (nommage, cycle de vie, description, mobilité, configuration)
- Le transport des messages : représentation (textuelle, binaire ou XML) des messages ACL, transport (par IIOP, WAP ou HTTP) de ces messages.

Ces standards évoluent, et sont régulièrement mis à jour, ainsi que de nouveaux standards qui sont nouvellement proposés. Les standards qu'édicte la FIPA ne constituent pas vraiment une plate-forme de *construction* multi-agents. Ce n'est pas non plus l'objectif que s'est fixé la FIPA. Tout au plus, la FIPA normalise une plate-forme *d'exécution* standardisée dans un but d'interopérabilité. Ces normes s'appliquent donc pour la plupart en phase de déploiement. Elles n'abordent pas les phases d'analyse ni de conception. Elles peuvent cependant guider certains choix d'implémentation.

V.3. Caractéristiques requises

Nous avons entrepris une étude sur les plates-formes SMA pour répondre à différentes problématiques inhérentes à des projets de recherche et développement. Ces projets, et notamment le simulateur ANTIC[33], nécessitaient certaines caractéristiques telles que la mobilité d'agents sur réseau, le développement modulaire d'agents, l'acquisition dynamique de compétence, le découpage fin du rôle tenu par un agent et un système de délégation. Nous souhaitions aussi pouvoir stopper une simulation et la restaurer dans l'état où elle se trouvait au moment de l'arrêt (persistance).

Dans la mesure où nous voulions pourvoir réutiliser le produit dans d'autres domaines, nous recherchions aussi des caractéristiques comme la gestion complète du cycle de vie d'un agent, une certaine légèreté de l'infrastructure et une bonne maintenabilité. Ce dernier point passe par un langage répandu dans l'industrie logicielle. Malgré l'intérêt offert par des langages comme REBOL [Auveriot, 2002], nous recherchions un langage d'implémentation répandu comme Java ou C++ pour assurer la pérennité de nos futurs développements et rendre les développements plus modulaires grâce à l'utilisation de « widjets[34] ».

En outre, nous souhaitions maîtriser intégralement le code de manière à éventuellement pouvoir intervenir sur un dysfonctionnement de l'outil ou apporter une évolution. Une plate-forme en Open Source était donc la cible privilégiée de nos investigations.

Afin de garantir une ouverture du produit vers d'autres plates-formes agents, nous avons orienté notre recherche sur un produit qui respecte les normes et les standards en vigueur. Cette volonté de respect des standards nous orientait naturellement vers un produit « FIPA compliant » construit sur un modèle d'architecture standard et qui autorise la communication entre agents selon un protocole reposant également sur un standard. Néanmoins une plate-forme échangeant des messages entre agents au format KQML ou MASIF [Milojicic & al., 1998] aurait retenu notre attention.

Afin d'autoriser une communication avec des logiciels du marché, nous avons souhaité orienter notre choix vers un produit offrant des possibilités de communications standardisées tel que le format XML. Cela nous permet en principe de faire communiquer assez facilement un progiciel quelconque avec une plate-forme agent.

Nous souhaitions enfin le produit fortement orienté réseau avec une méthode de distribution d'objets tel que RMI, DCOM, HORB[35] ou CORBA pour invoquer nos agents à distance.

[33] Voir le chapitre 9
[34] Les widjets sont des composants logiciels réutilisables
[35] HORB est une technologie open source d'objet distribué en JAVA ; http://www.horb.org/

V.4. Etude des produits retenus

Nous avons retenu les Aglets[36], développés par IBM au laboratoire de recherche de Tokyo, pour l'implémentation de la migration physique d'agents d'un hôte à un autre et pour leur mécanisme de sécurité. Un aglet est un objet JAVA qui peut se mouvoir d'une machine hôte à une autre. Un aglet s'exécutant sur un hôte peut stopper son exécution, se distribuer sur un autre hôte et reprendre son exécution. Il existe un mécanisme de sécurité [Karjoth & al., 1997] permettant à une machine hôte d'accueillir un aglet inconnu et l'exécuter en toute sécurité. La documentation est succincte mais il est possible d'acquérir le livre [Lange & Oshima, 1998] qui facilite la prise en main. Le « Aglet SDK » est un produit Open Source. Il supporte correctement JAVA 1.1 mais il est possible de le faire fonctionner en bricolant un peu sous Java 1.2. Certains benchmarks [Silva & al., 2000] ont montré une certaine lenteur d'exécution mais reportent une réelle fiabilité. Le SDK est téléchargeable sur http://sourceforge.net/projects/aglets/. Malgré ses qualités, ce produit n'est pas souvent suivi.

Nous avons également trouvé un intérêt à l'étude du produit Concordia [Wong & al., 1997] et [Paciorek & Wong, 1998] développé par Mitsubishi Electric Research Laboratories. C'est un environnement de développement et de gestion d'applications d'agents mobiles sur réseau écrit en JAVA. Concordia met l'accent sur des fonctionnalités comme la mobilité, la sécurité et la persistance. Un agent voyage de manière sécurisée [Walsh & al., 1999] d'un endroit à un autre sur le réseau en effectuant différentes sous–tâches sur chaque machine hôte du réseau. Si les données sont situées sur un hôte particulier, l'agent effectuera les traitements concertants ces données à l'endroit le plus proche possible de la source pour minimiser le trafic réseau. On trouve aussi la notion d'itinéraire pour ajouter des destinations intermédiaires à un voyage. Les agents peuvent reprendre un traitement interrompu par une interruption du réseau, c'est-à-dire que l'état interne de l'agent est maintenu lorsqu'il voyage sur le réseau. C'est un mécanisme de persistance intéressant. Enfin les agents collaborent pour réaliser des tâches. Une application pouvant être décomposée en plusieurs sous-tâches [Castillo & al., 1998]. Le produit possède certains bugs ennuyeux et manque cruellement de performance. Il ne semble pas véritablement maintenu, ce qui est fort dommage. Nous retiendrons essentiellement de ce produit son mécanisme de persistance et de mobilité.

Mobiware [Angin & al., 1998] n'est pas un toolkit SMA mais un outil utilisé pour piloter les réseaux et paramétrer les qualités de service. Cet effort porte par extension sur la création de nouveaux équipements réseau plus adaptés pour supporter les applications mobiles. Nous nous sommes intéressés à ce produit dans l'idée d'inclure une partie de ses fonctionnalités dans notre middleware afin d'optimiser la bande passante des réseaux. Il est programmé en JAVA et en CORBA.

Un autre toolkit digne d'intérêt pourrait être FIPA OS de Nortel Network [Poslad & al., 2000] et [Buckle, 2000]. Ce produit développé en JAVA 1.2 est Open Source, il permet de développer rapidement des agents respectant les spécifications FIPA que ce soit pour les communications entre agents, la gestion des agents, le transport des messages ou l'intégration d'agent. JADE, développé à l'Université de Parme était un produit un peu similaire qui proposait dans sa version 1.2 un respect des spécifications 1.2 de FIPA97. La dernière version à ce jour est la 2.6 en licence LGPL [Bellifemine & al., 2001], elle requiert JAVA 1.2.

Le respect des standards est souvent un « argument de vente » pour des plates-formes SMA. Le logiciel Voyager, développé par Recursion Software, implémente beaucoup de technologies « à la mode ». Parmi ces technologies, on peut retenir CORBA, COM, XML, SOAP, EJB etc. Le produit est téléchargeable en version d'évaluation sur leur site. Aucun

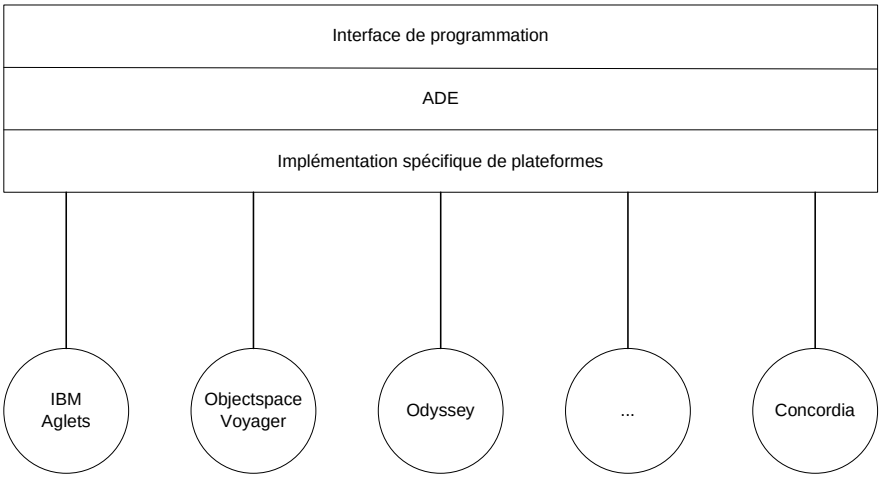
[36] Site web des aglets : http://www.trl.ibm.co.jp/aglets/

article scientifique ne présente individuellement ce produit. Il est possible de se reporter à des articles généraux évaluant les performances de plusieurs plates-formes SMA [Silva & al., 2000] qui signalent que ce produit manque particulièrement de performance et souffre de nombreux dysfonctionnements.

La mobilité[37] est aujourd'hui une caractéristique essentielle pour les plates-formes agent. Certaines sont même développées exclusivement pour satisfaire ce besoin de mobilité [Aramira, 2002] ou [Minar & al, 1999]. AgentX, conçu et commercialisé par International Knowledge Systems, est un ensemble de librairies agents distribuées en Java 1.3. Les librairies fournissent un ORB plus facile à utiliser que RMI ou CORBA. Agentx permet aussi la création d'agents mobiles autonomes. Les agents se déplacent sur un réseau et peuvent se rattacher à un ordinateur hôte. Cela permet de décomposer un programme complexe en plusieurs entités et de les distribuer sur des machines hétérogènes, ce qui permet d'utiliser un réseau de micro-ordinateurs comme un mainframe multi-processeurs. Ces fonctionnalités intègrent donc le concept de P2P et de Grid Computing avec des communications sécurisées[38].

La mobilité inclut aussi le support des PDA[39]. Des outils comme Grasshopper de IKV++, communiquent selon le standard MASIF et CORBA et sont accessibles aux agendas électroniques sous Windows CE. La version actuelle est la 2.2.4. Grasshopper [IKV, 1999] est particulièrement utilisé dans les applications de commerce électronique, dans la recherche d'information sur le web, dans l'informatique embarquée et dans les services de télécommunication. Ce produit ne semble pas souffrir de dysfonctionnement particulier et ses performances semblent honorables.

Tous ces produits présentent chacun de l'intérêt au niveau des fonctionnalités mais peu d'entre eux ne proposent de modèle sur lequel s'appuyer. Ces outils ne présentent qu'un ensemble de fonctions autorisant la création d'agent, la migration etc. L'intérêt d'un SMA repose notamment sur les interactions entre agents. C'est la raison pour laquelle l'Université de Potsdam a conçu ADE (*Architecture type-based Development Environment for agent-based application systems*). Il ne s'agit pas d'une plate-forme à proprement parler mais plutôt d'un produit qui encapsule d'autres plates-formes et qui joue un rôle de médiateur entre elles.



[37] La mobilité d'un agent consiste à le faire migrer d'un hôte à un autre. L'invocation d'objet à distance est souvent assimilée à de la mobilité alors qu'il ne s'agit pas d'un déplacement physique des agents.
[38] Il y a peu de documentation sur cet outil commercial encore en cours d'évolution exceptée une page web accessible via l'URL www.iks.com.
[39] Personal Digital Assistant

ADE [Horn & al., 1998] et [Reinke & al., 1999] est adressable via CORBA, c'est-à-dire à partir de langages comme JAVA ou C++. Curieusement on ne trouve que très peu de documentation sur ce produit qui repose pourtant sur une bonne idée. Un projet similaire fut entrepris par le CRIM [Magnin & Alikacem, 1999] dans le cadre de l'outil GenA.

La capacité d'apprendre ou de manifester un comportement intelligent peut résulter de l'utilisation de diverses techniques d'intelligence artificielle. Développé par Adaptive Objects, Direct IA[40] (Direct Intelligent Adaptation) est un SDK de haut niveau qui offre la possibilité à un agent ou groupe d'agent, d'apprendre, d'anticiper et de sélectionner leurs actions. Un agent est capable de s'adapter aux environnements dynamiques. Il peut découvrir de nouveaux comportements et tester leur efficacité dans le temps. Ce SDK est la résultante d'un travail de 10 ans au centre de recherche MASA[41]. DirectIA repose sur des techniques propriétaires d'Intelligence Artificielle et se présente sous la forme d'un ensemble de librairies C++ pour plateformes Windows et peut servir pour des applications ludiques (wargames), des applications de simulation ou de formation. Peu d'informations sont disponibles sur ce produit.

La décomposition modulaire de rôle ou de tâche est présente dans certains outils comme iGENTM développé Par CHI Systems. Ce logiciel inclut un outil de haut niveau de construction d'agents pour le développement d'applications "intelligentes" basées sur le principe des systèmes experts. iGEN™ est basé un modèle psychologique appelé le Cognitive Network of Tasks (COGNET) et est implémenté en C++ comme une architecture de BlackBoard [Zachary & al., 2001]. Il permet aux agents cognitifs de s'adapter à des évènements nouveaux, à changer les priorités des tâches et à ajouter des tâches non prévues. Il existe aussi un système à base de cas qui lui permet de s'adapter en fonction de la ressemblance avec des cas précédents. Ce système peut être intéressant pour un module « décisionnel » des agents cognitifs. Il n'est malheureusement pas possible de l'évaluer pour le moment puisqu'il ne sera disponible en téléchargement qu'en fin d'année 2003 pour les universitaires.

Dans le même ordre d'idée, le module décisionnel des agents peut être modélisé sous forme de BDI. Le logiciel JAM [Hubert, 1999], conçu par Intelligent Reasoning Systems, est une architecture base sur le triplet Croyance-Désir-Intension. Elle est basée sur le système de raisonnement procédural (PRS) de Georgeff, Ingrand, Rao, Lansky... Contrairement à la plupart des plates-formes existantes aujourd'hui qui sont opérationnelles dans un domaine souvent restreint, celle-ci peut être utilisée dans beaucoup de domaines d'application. Elle supporte à la fois le "goal-based reasoning top-down" et le "bottom-up data-driven reasoning". Un agent JAM possède un modèle du monde qui est une base de données représentant les croyances de l'agent. Il possède aussi un interpréteur qui lui permet de décider quoi faire et quand. Si un niveau méta de raisonnement n'est pas défini, JAM sectionne les buts en fonction de leurs priorités. Sont pris en compte : la réalisation du but (atteindre un espace d'état particulier), la maintenance (revenir sur un espace d'état) et la performance (coût de l'opération). Il est possible de prendre en compte plusieurs buts simultanément avec l'interruption et la reprise de chaque but (intention). La planification des buts inclut des pré-conditions et des post-conditions. La construction de plan inclut les itérations séquentielles, les sous-buts, les segments de plan atomiques (qu'il est impossible de stopper), des exécutions déterministes et non déterministes, l'exécution parallèle de plusieurs segments de plan, des synchronisations de buts, des traitements d'erreurs. Le code source Java

[40] Le site web de DirectIA est consultable sur http://www.directia.com
[41] MASA website http://www.animaths.com

et la documentation sont disponibles en téléchargement gratuitement pour un usage non commercial.

La simulation multi-agents intéressant les biologistes nécessite une grande quantité d'agents informatiques pour reproduire par exemple le fonctionnement d'une fourmilière. La gestion d'un grand nombre d'agents nécessite une conception et une implémentation différentes de la plate-forme. Il est impossible de simuler une fourmilière si chaque agent est géré par un thread. Pour palier à ce problème, des plates-formes comme Madkit utilisent des systèmes de sondes pour actualiser la simulation. De même des produits comme Swarm [Daniels, 1999] sont particulièrement adaptés pour la simulation de grand nombre d'agents. Développé à l'origine à l'Université de Santa Fe, le programme Swarm est très esthétique et très rapide mais de nombreux bugs sont recensés. On décrit de nouveaux agents à l'aide du langage Java interfacé aux librairies C++ à l'aide du JNI. Swarm supporte les plates-formes Sun Solaris, HP/UX, DEC Alpha, SGI, GNU/Linux sur Intel 386 et PowerPC ainsi que sur Windows NT. Swarm est développé en Java et Objective-C. Nous considérons cette possibilité de simuler de grands systèmes intéressante mais elle n'est pas primordiale dans notre recherche.

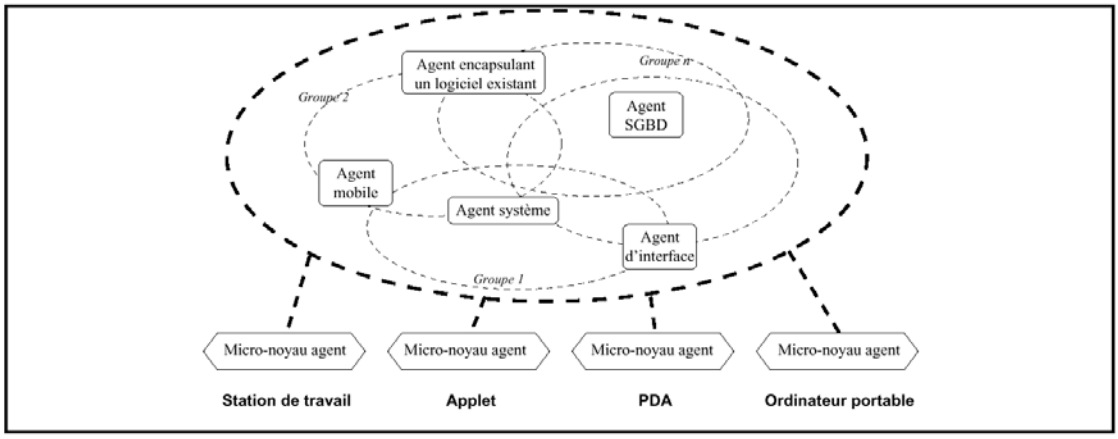
Un des produits les plus complets que nous avons rencontré est ZEUS [Nwana & al., 1999], créé par British Telecom. C'est un environnement d'agents "collaboratifs" écrit en JAVA. L'interface graphique est visuelle et la boite à outils est très complète (avec en particulier un éditeur d'ontologie) mais très lente ce qui rend son utilisation désagréable. Les exemples fournis ne semblent pas fonctionner correctement. Il se compose sous la forme de composants logiciels. Chaque agent se compose d'une couche de définition, d'une couche d'organisation et d'une couche de coordination. La première est construite juste au-dessus de l'API JAVA et représente le raisonnement de l'agent et ses capacités d'apprentissage, ses buts, ses ressources, ses compétences, ses croyances, ses préférences etc. La deuxième couche décrit les relations de l'agent avec les autres agents. Enfin la troisième couche décrit la coordination et les techniques de négociations possédées par l'agent. Les protocoles de communication sont placés au-dessus de la couche de coordination et implémente la communication inter-agents. Zeus semble une très bonne base de développement, la plate-forme est riche fonctionnellement, la documentation est correcte, mais la mise à jour du produit ne se fait pas régulièrement. La dernière version date de mai 2001.

La plupart de ces systèmes se présentent comme des bibliothèques de fonction pour la création d'agents sur réseau. Peu d'entre eux reposent sur une méthodologie de développement ou sur un modèle organisationnel. Il existe cependant des outils qui s'appuient sur des modèles conceptuels. Des outils comme Magique, développé au LIFL ou Madkit au LIRMM sont des plates-formes intéressantes.

Madkit [Gutknecht & Ferber, 2000] est, quant à lui, un logiciel gratuit est développé au LIRMM[42]. C'est une plate-forme multi-agents complète écrite en Java et en licence GPL et LGPL. Le programme s'installe très simplement et l'interface graphique est agréable. Le noyau central de la plate-forme Madkit est très réduit, puis est étendu à l'aide d'agents spécialisés. Les agents peuvent être programmés en divers langages comme Scheme, Python ou Jess. L'organisation de ce système multi-agents est basée sur un modèle conceptuel AGR de Aalaadin [Gutknecht & Ferber, 1998a] dans lequel les agents sont considérés comme des entités actives, qui communiquent avec d'autres agents et peuvent prendre un rôle au sein d'un groupe. Les communications entre agents et la gestion des groupes sont décentralisées. La plate-forme MadKit est réalisée en JAVA 1.3 et base son implémentation sur les concepts

[42] Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier. http://www.lirmm.fr

d'agents, de groupe et de rôle, ainsi que trois grands principes d'architecture (un micro-noyau agent, une agentification des services et un modèle graphique componentiel).



Contrairement à d'autres plates-formes, MadKit utilise des agents standards pour gérer l'envoi de message en distribué, la migration, la sécurité et d'autres aspects similaires. Cela rend la plate-forme particulièrement adaptable, vu que les services agents peuvent être remplacés à loisir. Madkit est un produit assez bien maintenu, bien programmé et reposant sur une véritable méthodologie. Ces années d'expérience en font un produit assez robuste et performant [Gutknecht & al., 2000].

DIMA[43] (Développement et Implémentation de Systèmes Multi-Agents) est un environnement de développement de systèmes multi-agents dont le développement a débuté en 1993, dans le cadre de la thèse de Z. Guessoum [Guessoum, 1996]. A l'origine développé en SmallTalk [Guessoum, 1998], il fut porté ensuite en Java [Guessoum & Briot, 1999]. Le modèle de DIMA propose une architecture de méta-niveau pour décomposer le comportement d'un agent en une structure de comportements. Cette architecture comprend une première strate composée de modules décrivant les différents comportements d'un agent et une deuxième strate représentant le méta-comportement de l'agent. Ces modules représentent les différents comportements d'un agent tels que la perception (interaction agent-environnement), la communication (interaction agent-agent) et la délibération. Un agent peut ainsi avoir un ou plusieurs modules qui peuvent être réactifs ou cognitifs. Elle permet à l'agent de raisonner et d'agir sur ses différents comportements. DIMA offre la possibilité de créer des agents de type réactif, cognitif et hybride ; ce dernier type allie des capacités réactives à des capacités cognitives, ce qui leur permet d'adapter leur comportement en temps réel à l'évolution de leur univers. Ces agents peuvent être organisés en utilisant la notion de groupes et de rôles. Excepté quelques articles scientifiques, il n'y a pas de support pour ce produit ni de site web et donc aucune possibilité de l'évaluer. Il ne sera donc pas retenu dans le cadre de notre étude.

MAGIQUE (Multi-AGent hiérarchIQUE) est un framework proposant une approche orientée agent pour la résolution de problèmes. Développée au laboratoire d'informatique de Lille, c'est à la base un modèle d'agents qui propose une structure organisationnelle hiérarchique [Nourredine & Mathieu, 1997b]. Un agent est récursivement constitué d'agents mais il peut également être décrit comme une hiérarchie d'agents. Cette structure propose un mécanisme de délégation de compétences entre agents, facilitant ainsi le développement. Ce produit est certainement celui qui se rapproche le plus de notre propre plate-forme, dans la mesure où il considère un agent comme une entité possédant un certain nombre de compétences. Ces compétences permettent à un agent de tenir un *rôle* dans une application

[43] http://www-poleia.lip6.fr/~guessoum/dima.html

multi-agents. Un agent est construit incrémentalement à partir d'un agent élémentaire "vide" par enrichissement dynamique de compétences. Un agent atomique possède deux compétences de base : une pour interagir et l'autre pour apprendre de nouvelles compétence. Un agent est un agent atomique qui a appris de nouvelles compétences. Ces compétences peuvent évoluer (par échanges entre agents) au cours de l'existence de celui-ci, ce qui implique que les *rôles* qu'il peut jouer (et donc son statut) peuvent également évoluer au sein du SMA. Une compétence est un ensemble de fonctionnalités qui peuvent être exploitées par un agent [Routier & al., 2001]. Tout est exprimable en terme de compétences, même la manière de gérer les compétences elles-mêmes. D'un point de vue plus pragmatique, une compétence est vue comme un composant dont l'interface publique désigne les capacités qu'un agent peut exploiter. Le mécanisme de délégation et la modularité induite par les compétences facilitent la réutilisabilité des agents et de leurs compétences dans différents contextes ou applications. On trouve des agents réactifs et pro-actifs avec des bases de croyances et une fonction de calcul des priorités. Ces bases de croyance sont vérifiées et appliquées en chaînage avant. A noter que le terme « *d'agent intelligent »* (pour lequel se pose d'ailleurs l'interminable classique problème de la définition du terme intelligent) n'est qu'un cas particulier d'*agent* muni de compétences faisant intervenir des techniques d'intelligence artificielle (réseaux de neurones, systèmes experts, logique temporelle, etc.). Cette notion est également très proche de celle présentée dans nos travaux.

Dans MAGIQUE les agents sont définis au sein d'une plate-forme d'accueil située sur une machine. Une application MAGIQUE peut fonctionner selon un mode centralisé, dans lequel tous les agents fonctionnent sur la même plate-forme, ou selon un mode distribué dans lequel les agents tournent sur des machines différentes. Les agents communiquent dans ce cas via leur plate-forme respective en utilisant le mécanisme RMI. La plate-forme propose un mécanisme basé sur l'utilisation d'un agent appelé *superviseur* qui va mettre en liaison différents agents compétents et permettre une invocation de méthode (compétence) sans nécessairement se préoccuper de qui la possède. Le superviseur est un agent comme les autres excepté le fait qu'il a la responsabilité de la communication entre les agents qu'il gère. La définition d'un agent superviseur se fait par ralliement d'au moins un autre agent à celui-ci..

MAGIQUE[44] est développé en JAVA et possède environ 200 classes. Le produit est fonctionnel et se positionne comme une bonne plate-forme SMA. La documentation est assez succincte mais les petits exemples fournis permettent de bien comprendre le fonctionnement.

V.5. Conclusion

Nous avons recensé un certain nombre d'outils logiciels permettant de faciliter la création d'un SMA. Certains produits sont opérationnels et d'autres à l'état de prototype, voire même au stade de l'idée. La plupart des produits commerciaux ne reposent pas sur des modèles conceptuels mais se positionnent comme de simples bibliothèques de fonctions souvent programmées en Java ou en C++. Les produits universitaires reposent plus souvent sur un modèle organisationnel mais ne rivalisent que très rarement avec les logiciels commerciaux sur le plan du développement, excepté peut être pour Madkit ou Magique. Nous n'avons pas retrouvé dans un seul et même produit toutes les caractéristiques requises pour le développement de nos projets SMA.

De plus un produit Open Source fournit le code de l'application mais pas nécessairement le dossier de conception incluant les spécifications fonctionnelles, les

[44] Le site web de la plate-forme MAGIQUE est accessible via http://www.lifl.fr/MAGIQUE/

diagrammes de modélisation… Ces documents sont pourtant indispensables pour étendre les fonctionnalités d'un logiciel.

Nous considérons que l'intérêt intrinsèque d'un SMA repose sur la distribution et l'auto-organisation et pensons que l'apport d'un substrat d'échange commun contribuerait à favoriser une bonne organisation des agents. Tous les toolkits ne respectent pas les standards FIPA ou ne proposent pas une mise en œuvre des derniers standards de l'industrie logicielle. Chacun est donc souvent adapté pour répondre à une problématique particulière et n'offre pas systématiquement une ouverture et une modularité suffisante pour couvrir un vaste ensemble de projets. Des tests de performances ont montré que la plupart des plates-formes souffrent de problèmes de lenteurs et de certains dysfonctionnements bloquants. Nous pouvons donc en conclure qu'une majorité des plates-formes SMA manquent aujourd'hui de maturité pour supporter de grands projets informatiques. Cela était particulièrement vrai il y a trois ans, date à laquelle nous nous sommes posés la problématique du développement de notre propre outil.

Nous nous sommes aussi posé le problème du modèle organisationnel sur lequel nous souhaitions construire notre outil. Après un tour d'horizon sur les quelques modèles et méthodologies existants, nous avons entrepris d'organiser nos agents avec le quadruplet défini par le modèle YAMAM, présenté précédemment. Ce seul choix suffit à justifier la conception de notre propre plate-forme d'agents.

Au vu de ces remarques, nous avions entrepris la conception de notre propre plate-forme SMA, qui sera présentée au chapitre 7. A noter qu'en quelques années, les plates-formes « concurrentes » ont aussi évolué en adoptant de nouveaux standards, de nouveaux langages de développement et se sont fiabilisées par capitalisation des retours d'expérience.

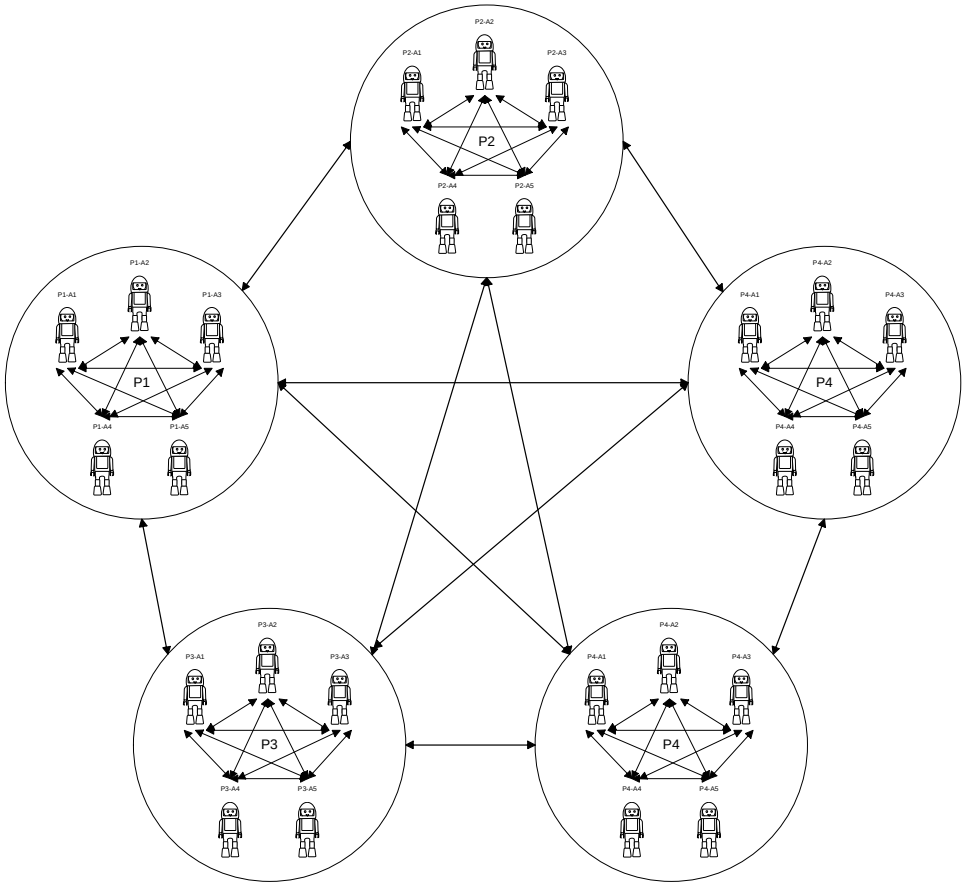
Chapitre VI. Choix technologiques

« Le progrès technologique n'abolit pas les obstacles; il en change simplement la nature » - Huxley, Aldous

Résumé : Un projet informatique requiert la mise en œuvre de différentes technologiques pour sa phase de développement. Ce chapitre présente les choix effectués pour la conception de notre plate-forme multi-agents. Cela implique le choix d'une architecture, d'un langage, d'un système de communication et de bien d'autres composants indispensables. Ce chapitre présente plus un travail d'ingénierie que de recherche et de nombreux termes techniques sont utilisés.

VI.1. Introduction

Dans un chapitre précédent, nous avons exposé les fondements de notre architecture organisationnelle YAMAM. Nous avons ensuite dressé un état de l'art des plates-formes multi-agents existantes pour lesquelles nous avons tenté de décrire leurs principales fonctionnalités et les choix d'implémentation de leurs concepteurs. Nous sommes à présent en phase de choisir les solutions technologiques les plus adaptées pour construire notre propre plate-forme SMA. Le choix sera motivé par l'intérêt intrinsèque des différentes technologies mais aussi parfois contraint par un cahier des spécifications fonctionnelles et techniques. Le choix se fera bien entendu selon l'évolution de la technologie, c'est-à-dire en tenant compte des technologies qui puissent assurément être maintenues dans le temps. Nous privilégierons les technologies soutenues par de grands constructeurs ou éditeurs pour assurer une certaine pérennité à notre outil logiciel. Outre les fonctions requises par une plate-forme SMA individuelle, nous souhaitons ajouter des fonctionnalités liées à la coopération des plates-formes elles-mêmes. L'agentification doit être réalisée au maximum.



Une plate-forme, composée d'agents (système multi-agents de niveau local) est vue alors comme un méta-agent à plus haut niveau, à tel point que plusieurs plates-formes reliées entre elles constituent également un système multi-agents via le réseau (système multi-agents de niveau global). Cet aspect de la granularité n'est pas sans rappeler [Brazier & al., 1995]. Cette architecture distribuée nous renverra à des technologies comme le Peer-to-Peer et le Grid Computing, qui seront étudiées dans ce chapitre.

VI.2. Recueil des fonctionnalités

Nous souhaitons développer un middleware qui présente les caractéristiques suivantes :

- Gestion complète du cycle de vie d'un agent
- Mobilité des agents
- Persistance du système
- Ouverture vers d'autres systèmes existants
- Implémentation dans un langage non ésotérique
- Flexibilité, Maintenabilité et réutilisabilité
- Légèreté de l'infrastructure
- Système d'élimination automatique des agents les moins performants
- Sécurité de fonctionnement et sécurité des communications
- Robustesse au changement
- Optimisation de la charge et de la bande passante
- Système décisionnel et état mental des agents cognitifs
- Exploitation maximum des ressources informatiques (CPU, Disque…)

- Système de délégation de tâches
- Découplage des rôles en tâches
- Interconnexion et coopération des plates-formes elles-mêmes
- Respects de normes et de standards en ce qui concerne les modèles d'implémentation et les protocoles de communication

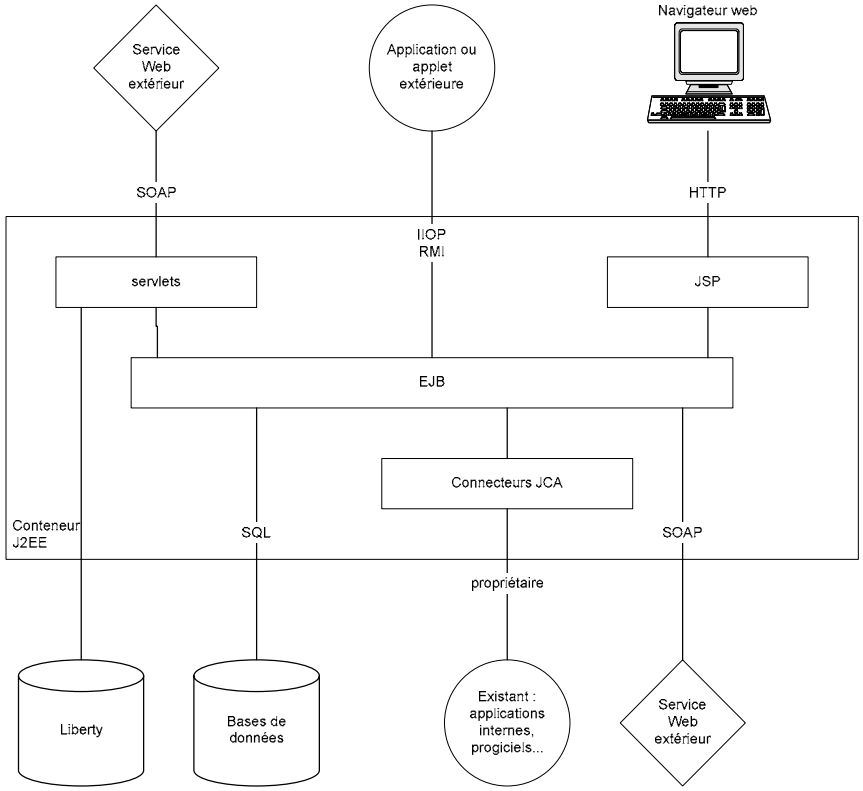
Pour cela nous allons passer en revue différentes technologies tant en éliminant d'amblée toutes celles ne pouvant répondre directement à nos attentes. Nous nous focaliserons donc uniquement sur celles présentant une alternative possible à nos choix finaux.

VI.3. Comparaison de J2EE et .Net

A ce jour, il y a deux technologies de base qui se présentent pour le développement et le déploiement des services et des applications web : l'univers JAVA, formalisé dans la spécification J2EE et l'univers Microsoft sous l'appellation .Net (*prononcer Dot Net*). Il n'y a pas de correspondance immédiate entre les technologies présentes dans J2EE ou dans .Net. La construction et le déploiement de services web mettent en œuvre telle ou telle technique ou tel et tel mécanisme fourni par une implémentation J2EE et .NET. et pas nécessairement la totalité de leurs fonctionnalités respectives.

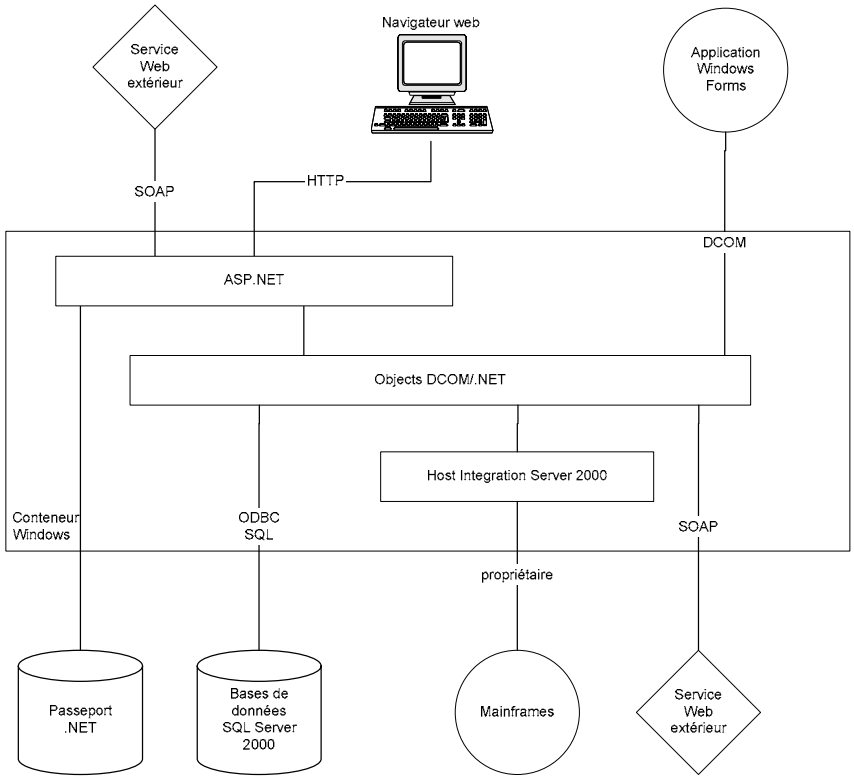
VI.3.1. Architecture J2EE pour les services web

Historiquement, J2EE a été conçu pour le développement d'applications côté serveur écrites en JAVA. J2EE peut être utilisé pour construire des sites web, des composants logiciels et des applications complètes. Sun a ensuite livré des spécifications de bibliothèques Java complémentaires à J2EE pour le traitement des documents XML. Les services web, comme les composants logiciels J2EE, voient leur implémentation hébergée par un conteneur interposant un certain nombre de services de gestion entre le programme client et l'application elle-même. Ce conteneur peut être un conteneur EJB, une servlet ou une page JSP. Certains protocoles de communication propriétaires sont également utilisables dans J2EE, comme RMI ou IIOP, en complément des protocoles et des standards des services Web.



VI.3.2. Architecture .NET pour les services web

Le système de Microsoft, appelée .NET, est un framework qui comprend le CLR, ASP.NET, ADO.NET, .NET Enterprise Services et d'autres bibliothèques, et qui s'installe au dessus des systèmes d'exploitation Windows XP, 2K et NT4. Le prolongement de l'évolution du système d'objets COM, puis des composants logiciels DCOM/COM+, de Microsoft trouve son expression dans le CLR (Common Language Runtime). Cette innovation est antérieure aux efforts de standardisation des services web. Reprenant du modèle Java l'idée de la machine virtuelle d'exécution et prolongeant l'idée originelle de COM de compatibilité des formats binaires, le CLR permet de bénéficier de tous les avantages des langages interprétés et des langages compilés – le CLR exécute un langage intermédiaire comparable au bytecode Java en le compilant au vol (just in time) pour l'architecture physique sous-jacente. Le CLR, complété par divers compilateurs engendrant le code dans ce langage intermédiaire permet donc de décrire ces applications dans un grand nombre de langage de programmation (ou dans un mélange), tous ramenés à un même modèle d'exécution. Ce tour de force est rendu possible par l'adoption d'un système commun de types utilisé par le CLR et auquel sont ramenés les différents types des langages individuels.



Simultanément, les grands modules de services ont été rhabillés au moyen de bibliothèques CLR, les rendant ainsi accessibles à tous les langages de programmation de façon cohérente. Les nouvelles possibilités techniques offertes par le CLR et la nouvelle génération des interfaces utilisateurs (Web Forms et Windows Forms), combinées à l'adoption des standards XML et services Web, ont ainsi donné naissance à une plateforme riche et complète de déploiements d'applications et de services à base de composants logiciels classiques et à base de services Web, ou résultant de l'intégration des deux.

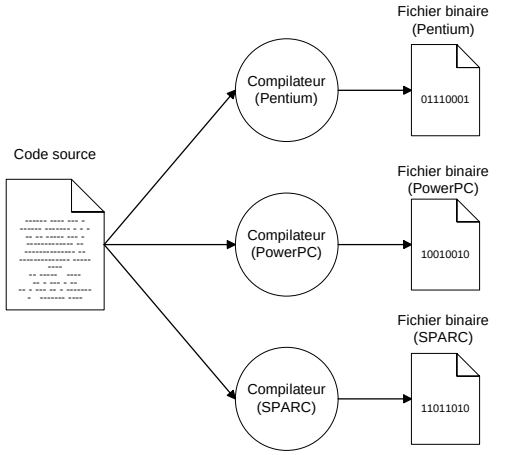
VI.4. Le langage de développement

A la date où nous avons entrepris le développement de la plate-forme Phoenix, l'architecture .NET de Microsoft n'était pas encore disponible en version finale. Nous nous sommes orientés sur le langage JAVA qui présentait néanmoins beaucoup d'avantages. Ce langage est apparu vers la fin de 1995 et obtint rapidement un énorme succès. L'objectif de JAVA est de constituer un outil universel capable de connecter les utilisateurs aux informations, que celles-ci proviennent de serveurs Web de bases de données, de fournisseurs d'informations ou de toute autre source imaginable. Il s'agit d'un langage de conception très performant qui a été adopté par la majorité des fournisseurs. Ses caractéristiques intégrées de sécurité offrent un sentiment de confiance aux programmeurs comme aux utilisateurs des applications. De plus, Java incorpore des fonctionnalités qui facilitent grandement certaines tâches de programmation avancées comme la gestion des réseaux, la connectivité des bases de données ou le développement d'applications multitâches.

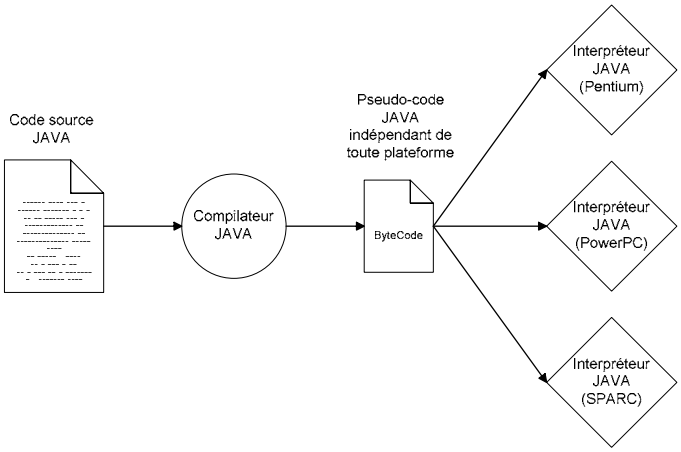
La réputation de Java en tant que langage informatique est néanmoins exagérée: Java est assurément un *bon* langage de programmation, sans doute, même l'un des meilleurs mais il n'a, pour le moment, pas tenu toutes ses promesses initiales.

VI.4.1. Les avantages de Java

L'un des avantages évidents de ce langage est une bibliothèque d'exécution qui se veut indépendante de la plateforme: en théorie, il vous est possible d'utiliser le même code pour Windows 95/98/NT, Solaris UNIX, Macintosh, etc. Cette propriété est indispensable pour une programmation sur Internet (cependant, par rapport à la disponibilité sur Windows et Solaris, les implémentations sur d'autres plates-formes ont toujours un léger décalage).



Compilateur classique avec génération directe de fichiers binaires



Génération d'un bytecode java « universel » interprétable par les interpréteurs des plateformes

Un autre avantage de ce langage de programmation réside dans le fait que la syntaxe de Java est analogue à celle de C++ ce qui le rend économique et professionnel. Le fait de créer une autre version d'un langage C++ n'est cependant pas suffisant. Le point clé est le suivant : il est beaucoup plus facile d'obtenir du *code sans erreur à l'aide de java qu'avec* C++. Pourquoi ? Les concepteurs de Java ont beaucoup réfléchi à la raison pour laquelle le code C++ contenait autant d'erreurs. Cette réflexion les a amenés à ajouter dans Java des fonctions destinées à éliminer la possibilité de créer du code contenant les types d'erreurs les plus courants (selon certaines estimations, le code C++ contient au moins une erreur toutes les cinquante lignes) :

- Les concepteurs de java ont supprimé l'allocation et la libération de mémoire manuelles. La mémoire dans java est allouée et libérée automatiquement. Vous n'avez *jamais* à vous préoccuper de pertes de mémoire.
- Ils ont éliminé l'arithmétique des pointeurs introduisant du même coup une vraie gestion de tableau. La notion de référence sur une zone mémoire remplace avantageusement celle de " pointeur", car elle supprime la possibilité d'écraser toute zone mémoire à cause d'un compteur erroné.
- Ils ont éliminé toute possibilité de confusion entre une affectation et un test d'égalité ans une instruction conditionnelle. Une instruction if (ntries - 3) ne pourra pas franchir l'étape de la compilation
- Ils ont supprimé l'héritage multiple en le remplaçant par une nouvelle notion d'interface dérivée d'Objective C. Les interfaces vous offrent tout ce que vous pouvez obtenir à partir de l'héritage multiple, sans la complexité de la gestion de hiérarchie d'héritage multiple.

VI.4.2. Caractéristiques

Les créateurs de Java ont écrit un livre blanc qui présente les caractéristiques fondamentales de Java. Ce livre est articulé autour des 11 termes suivants :

- ***Distribué :*** *Java possède une importante bibliothèque de routines permettant de gérer les protocoles TCP/IP tels que HTTP et FTP. Les applications Java peuvent charger et accéder à des données sur Internet via des* URL *avec la même facilité qu'elles accèdent à un fichier local sur le système. « Nous avons trouvé que les fonctionnalités réseau de Java sont à la fois fiables et d'utilisation aisée. Toute personne ayant essayé de faire de la programmation pour Internet avec un autre langage se réjouira de la simplicité de Java lorsqu'il s'agit de mettre en oeuvre des tâches lourdes, comme l'ouverture d'une connexion avec un socket. De plus, Java rend plus facile l'élaboration des scripts CGI (Common Gateway Interface), et un mécanisme élégant, nommé servlet, augmente considérablement l'efficacité du traitement côté serveur, assuré par Java. De nombreux serveurs Web, parmi les plus courants, supportent les servlets. Le mécanisme d'invocation de méthode à distance (RMI) autorise la communication entre objets distribués. »*

- ***Fiabilité :*** *Java a été conçu pour que les programmes qui l'utilisent soient fiables sous différents aspects. Sa conception encourage le programmeur à traquer préventivement les éventuels problèmes, à lancer des vérifications dynamiques en cours d'exécution et à éliminer les situations génératrices d'erreurs... La seule et unique grosse différence entre C++ et Java réside dans le fait que ce dernier intègre un modèle de pointeur qui écarte les risques d'écrasement de la mémoire et d'endommagement des données.*

- ***Orienté objet :*** *Pour rester simples, disons que la conception orientée objet est une technique de programmation qui se concentre sur les données (les objets) et sur les interfaces avec ces objets. Pour faire une analogie avec la menuiserie, on pourrait dire qu'un menuisier "orienté objet " s'intéresse essentiellement à la chaise l'objet qu'il fabrique et non à sa conception (le "comment"). Par opposition, le menuisier "non orienté objet " penserait d'abord au "comment "...*

- ***Simple :*** *L'objectif était de créer un système qui puisse être programmé simplement sans nécessiter un apprentissage ésotérique, et qui tire parti de l'expérience standard actuelle. En conséquence, même si nous pensions que C++ ne convenait pas, Java a été conçu de façon relativement proche de ce langage dans le dessein de faciliter la compréhension du système. De nombreuses fonctions compliquées, mal comprises, rarement utilisées de C++, qui nous semblaient par expérience apporter plus d'inconvénients que d'avantages, ont été supprimées de Java.*

- **Sécurité :** *Java a été conçu pour être exploité dans des environnements serveur et distribués. Dans ce but, la sécurité n'a pas été négligée. Java permet la construction de systèmes inaltérables et sans virus.*

- **Architecture neutre :** *Le compilateur génère un format de fichier objet dont l'architecture est neutre – le code compilé est exécutable sur de nombreux processeurs, à partir du moment où le système d'exécution de Java est présent. Pour ce faire, le compilateur Java génère des instructions en bytecode qui n'ont de lien avec aucune architecture particulière. Au contraire, ces instructions ont été conçues pour être à la fois faciles à interpréter et faciles à traduire en code natif.*

- **Portable :** *A la différence du C/C++, on ne trouve pas les aspects de dépendance de la mise ne œuvre dans la spécification. Les tailles des types de données primaires sont spécifiées, ainsi que le comportement arithmétique qui leur est applicable.*
- **Interprété :** *L'interpréteur Java peut exécuter les bytecode directement sur n'importe quelle machine sur laquelle il a été porté. Dans la mesure où la liaison est un processus plus incrémentiel et léger, le processus de développement peut se révéler plus rapide et exploratoire.*
- **Performances élevées :** *En général, les performances des bytecodes interprétés sont tout à fait suffisantes, il existe toutefois des situations dans lesquelles des performances plus élevées sont nécessaires. Les bytecodes peuvent être traduits à la volée en code machine pour l'unité centrale destinée à accueillir l'application.*
- **Multithread :** *Les avantages du multithread sont une meilleure interréactivité et un meilleur comportement en temps réel.*

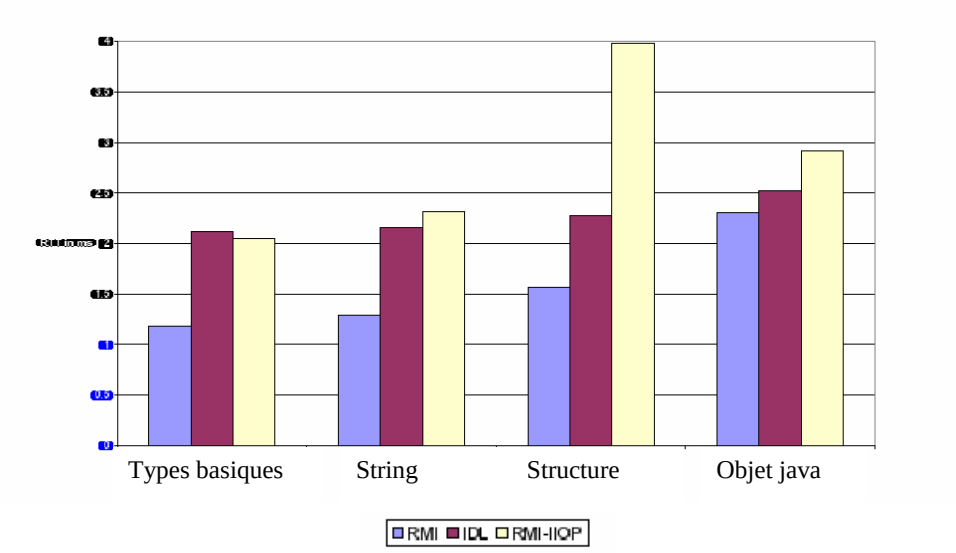
VI.5. La distribution

Nous avons travaillé sur deux standards qui sont RMI et CORBA. Il en existe d'autres très (plus ?) intéressants, simples et rapides comme HORB [Hirano, 1997] mais les contraintes de maintenance et d'évolutivité nous imposent de nous orienter sur un produit qui sera suivi dans le temps. Nous ne voulions pas non plus nous orienter sur une solution totalement propriétaire comme DCOM de Microsoft malgré son efficacité. Des tests de performances entre HORB, RMI, CORBA et DCOM sont réalisés dans [Hirano & al., 1998].

La première version de Phoenix fut programmée en RMI. Cependant, nous avons envisagé très tôt la migration de la plateforme Phoenix de JAVA vers un autre langage plus rapide. Par conséquent, il nous fallait remplacer RMI par une technologie identique mais adaptée sur plusieurs langages.

Une étude comparative[45], réalisée par *Matjaz B. Juric et Ivan Rozman,* met l'accent sur la comparaison des performances des trois technologies fréquemment rencontrées dans la programmation distribuée : RMI-JAVA, RMI*over*IIOIP-JAVA et IDL-CORBA. L'évaluation des performances se fait à l'aide d'une méthode appelée « Round Trip Time » (RTT). RTT représente le temps total que prend l'invocation d'une méthode. Chaque méthode utilisée pour l'évaluation transfert des paramètres et retourne simplement des valeurs (i.e aucune méthode n'exécute de processus). La nature des types influençant les résultats, le test a été réalisé suivant onze types : les types basiques (Moyenne de huit types primitifs), le type String, un type définit par l'utilisateur (Structure) et un objet Java défini par l'utilisateur.

[45] http://www.javareport.com



RTT avec un seul client pour11 types (8 Primitifs, String, 2 types Utilisateur)

On voit sur ce graphique qu'en majorité, CORBA-IDL est plus performant que RMI over IIOP. Nous allons présenter brièvement ci-dessous ces différentes technologies.

VI.5.1. RMI

L'intérêt immédiat de RMI est sa simplicité de mise en œuvre. Son principal inconvénient demeure sa restriction au langage JAVA.

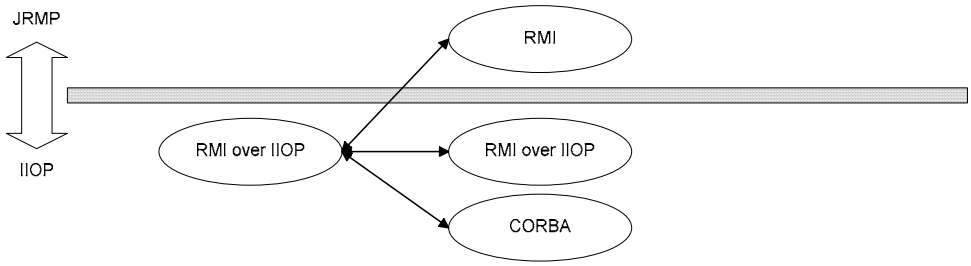
Le principe de fonctionnement consiste à écrire une interface java définissant l'ensemble des méthodes de l'objet qui pourront être invoquées à travers le réseau. Cette interface va dériver de l'interface java : *Remote*, et sera implémentée par l'objet héritant lui même de la classe java : *UnicastRemoteServer*. L'objet aura alors deux parties, la partie distribuée provenant de l'interface et le reste qui sera sa partie locale.

Une fois l'objet distant codé, il est compilé avec le compilateur rmic, qui va créer deux classes : une classe *stub* du coté de la machine appelante et une autre nommée *skeleton* pour la machine appelée. Une fois cette étape terminée, l'objet est prêt à être distribué. Le serveur RMI (le *rmiregistry*) doit être lancé et un programme java ira inscrire une instance de notre objet nouvellement créé auprès de ce serveur. Dès cet instant, l'objet est distribué et accessible à travers le réseau.

VI.5.2. RMI over IIOP

La principale des restrictions imposées par RMI est le fait que ce dernier n'est utilisable que sous java. Par exemple, si un client est écrit dans un autre langage, il ne pourra pas se connecter au serveur afin de pouvoir manipuler l'objet à distance. Pour palier à ce problème, IBM, en collaboration avec Sun, propose une nouvelle extension, actuellement en distribution, les « *RMI over IIOP* ». IIOP est un protocole proposé par CORBA afin de permettre à des objets programmés dans des langages différents de communiquer entre eux.

Pour permettre à des objets interfacés RMI de communiquer avec des objets interfacés IDL sous CORBA, la solution consiste à recompiler les anciennes interfaces RMI avec l'option -iiop et de déployer ainsi une interface IDL. Le *rmiregistry* permet de se connecter à un bus IIOP. Les deux ainsi réunis, donnent la possibilité à un objet distribué sous RMI de communiquer avec un objet distribué sous CORBA.



Bien sûr, il est toujours nécessaire d'implémenter les IDL obtenues dans d'autres langages pour pouvoir manipuler les objets.

La facilité du mécanisme de RMI lié par RMI over IIOP à l'ouverture qu'offre CORBA est très intéressante. Néanmoins, ce mécanisme possède des contraintes non négligeables. Les IDL produites par RMI over IIOP doivent être implémentées, ce qui peut être pire que de créer sa propre IDL. Par exemple, prenons un objet dont une des méthodes distribuées est : *Vector getIndice()*. Dans le Vector sont stockés des Integer de java. Imaginons un client de cet objet qui utilise une seule méthode du Vector pour récupérer les objets Integer, et une méthode d'Integer pour obtenir le nombre entier qu'il contient. En passant par CORBA, on souhaite avoir un client programmé en C++ pour pouvoir attaquer l'objet à travers le réseau. RMI over IIOP produira alors l'IDL pour notre objet distribué, mais aussi celle du Vector avec les 42 méthodes qu'il contient qui devront être implémentées et il ne faudra pas oublier de déployer (créer) l'IDL de l'Objet Integer. De même, si le Vector peut contenir des objets java de différents types, cela complexifie encore l'opération !

En outre, la sécurité n'est pas encore prévue entre une application RMI over IIOP et CORBA. Ce problème qui sera résolu dans le futur, suffit à lui seul à ne pas choisir cette solution.

VI.5.3. CORBA

Finalement, parmi les trois technologies distribuées, notre choix s'et porté sur CORBA 2.3. Nous allons présenter CORBA [Geib & al., 1999] dans les paragraphes suivants :

CORBA est un produit de l'OMG, organisme fondé en 1989 par huit sociétés : 3Com, Américan Airlines, Canon, Data Général, Hewlett-Packard, Philips Telecomunication, Sun Microsystem et Unisys. Le nombre d'adhérents n'a cessé de croître pour atteindre plus de 800 membres en 2000. On trouve au sein de cet organisme l'ensemble des acteurs de la scène informatique, c'est-à-dire des constructeurs comme IBM ou Compaq, des éditeurs comme Inprise ou Microsoft, des universitaires ainsi que des utilisateurs comme Air France ou France Telecom. Cet organisme est un groupe de travail ayant pour but de promouvoir et de standardiser l'architecture d'objets distribués. Son but est de définir les bases de l'architecture distribuée universelle appelée OMA (Object Management Architecture).

Le but d'une architecture distribuée est de concevoir des applications morcelées qui seront réparties sur plusieurs ordinateurs. Un des points clés de ce type d'architecture est d'obtenir une interopérabilité aussi bien du point de vue de la plate-forme matérielle, du système d'exploitation et du langage de conception. Ce point est important pour Phœnix puisque la migration de son système de communication vise (si possible) l'interopérabilité entre différents langages de conception.

Le rôle de l'OMA est d'identifier, de spécifier et de normaliser les composants essentiels à une architecture distribuée. CORBA (Common Object Request Broker Architecture) est une norme qui spécifie et normalise le modèle d'architecture distribuée afin de permet l'interopérabilité entre les OMA. Cette architecture favorise la conception

d'application découpée en modules d'origines diverses. Le modèle de conception qui sert de fondation à la réalisation de CORBA s'appuie sur la programmation orientée objet. C'est le modèle de conception logicielle qui offre une méthodologie pour réaliser de grandes applications, plus facilement implémentables et évolutives.

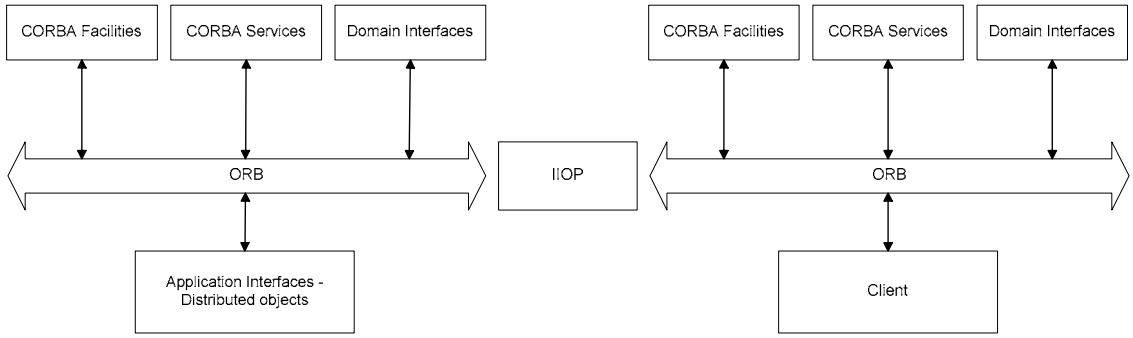
VI.5.3.1. IIOP

Le protocole de communication entre les différentes OMA est IIOP (Internet Inter-Operable Protocol), qui est basé sur le protocole TCP/IP. L'IIOP est une spécialisation du General Inter-Operable Protocol (GIOP) et sont tous les deux des protocoles de transport. IIOP définit la norme utilisée pour le format des messages lors de communications entre objets et applications sur Internet. C'est une surcouche du protocole de transport TCP/IP, dont le but est de transporter les octets, et qui est compris par l'ensemble des systèmes actuellement en service. Au dessus de la couche IIOP de CORBA, on trouve le bus ORB.

VI.5.3.2. ORB

L'ORB (Object Request Broker) est le cœur de l'OMA. C'est lui qui gère la relation entre les différents éléments de l'architecture ainsi que la communication entre ces éléments. Il est aussi appelé bus logiciel. Ses deux principaux objectifs sont de permettre l'interpolation de CORBA et de rendre transparent le dialogue entre ses différents constituants.

Du point de vue conception sous CORBA, l'ORB est l'agent de négociation qui fera la liaison entre un client et l'objet distribué demandé par ce dernier. Il permettra d'insérer aisément l'utilisation des services (servicies) et des utilitaires communs (facilities) proposés par CORBA.



VI.5.3.3. IOR

Comme on a déjà pu le constater, l'architecture CORBA est prévue afin de distribuer un programme à travers un réseau, voire Internet. Afin de pouvoir identifier un objet précisément, l'OMG a normalisé une représentation unique de la référence d'un objet appelé IOR (Interpolable Object Reference). Cette référence est constituée du nom de l'ordinateur où se situe l'objet, le port TCP utilisé, l'identifiant de l'objet sur cette machine etc.

VI.5.3.4. IDL

L'IDL (Interface Definition Language) est un langage descriptif chargé de déclarer la partie distribuée d'un objet sous CORBA. Cette déclaration se présente sous la forme d'une interface qui présentera un prototype de l'objet, déclarera ses caractéristiques et ses opérations. C'est par cette interface que l'objet distribué pourra être manipulé par un client.

Les IDL sont découpés en modules et sous modules qui contiennent les interfaces. Les interfaces sont les seuls mots clés qui permettent de définir un objet CORBA. Les interfaces contiennent des attributs qui représenteront les champs de l'objet ainsi que des opérations qui seront les méthodes permettant de manipuler l'objet à distance.

L'interface permet l'encapsulation telle que le définit la programmation objet. Une interface peut hériter d'une autre, mais contrairement à la programmation objet - l'IDL n'étant qu'un langage de définition - l'héritage ne concernera que la déclaration des attributs d'un objet. De même pour le polymorphisme, il ne concerne que l'implémentation multiple d'une unique interface.

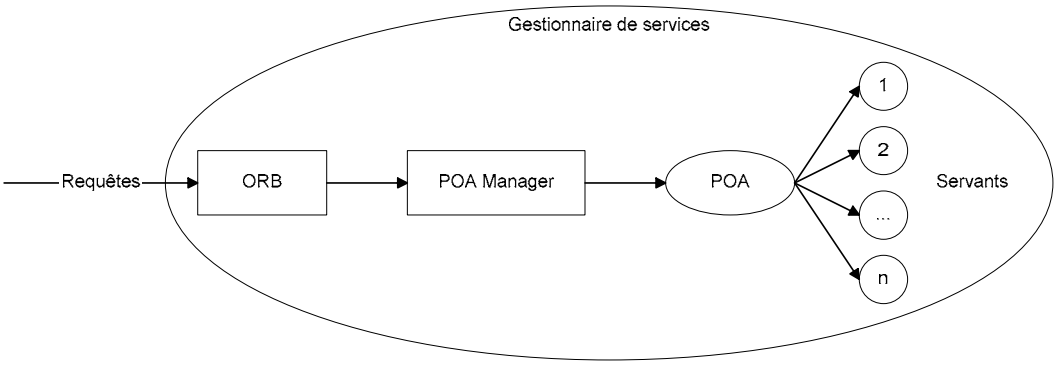
Une fois une interface définie dans CORBA, il nous faut la déployer dans le langage cible de l'implémentation. Si l'on souhaite utiliser l'interface dans deux langages différents, il nous faudra donc déployer deux fois la même interface.

VI.5.3.5. L'adaptateur d'objet

Dans l'OMA, on trouve les adaptateurs d'objet (Object Adaptator) du côté serveur, ils prennent place entre la couche de l'objet et celle de l'ORB (Object Request Broker). Ils servent de liaison entre ces deux composants et sont chargés d'assurer l'activation des objets, la transmission de la demande d'un client vers un objet, la gestion de l'appel de méthodes standard pour la création et destruction d'Objet, la détermination de l'état d'un objet et la détermination des politiques de gestion des objets. CORBA proposait autrefois le BOA, qui est une version basique de l'adaptateur d'objet. La notion d'héritage de la programmation objet permet par la suite de dériver ce BOA afin d'obtenir des adaptateurs d'objets plus spécifiques : Citons pour exemple l'OODA (Object Oriented Database Adapter) qui permet de gérer les objets provenant de base de données. Le manque de spécification des OA dans l'OMA a produit un effet pervers. Dans l'implémentation de CORBA par les éditeurs de logiciels, on a vu apparaître des fonctionnalités au BOA propres à chacun de ces éditeurs. Le résultat fut bien sûr une incompatibilité de codage d'un serveur lorsque ce dernier devait migrer d'un ORB à un autre. Afin de pallier à ce problème, l'OMG définit entièrement dans la norme du CORBA 2.2 le POA (Portable Object Adapter) et demande à ce que le POA soit utilisé à la place du BOA pour tous nouveaux projets CORBA. Le POA est un Object Adaptater indépendant du BOA, cependant le BOA, pour des raisons évidentes de compatibilité avec les anciens systèmes, n'a pas disparu, et cohabite donc avec le POA.

Dans les réflexions initiales du projet, Phœnix disposait de plusieurs types d'objets à distribuer comme les Agents, le Market… La hiérarchisation des POA nous aurait été utile. Comme nous le verrons plus loin, cette remarque ne nous a pas servi car lors de la révision du partage de Phœnix, il ne restait plus qu'une seule méthode distribuée.

Le POA gère donc la liaison entre la référence d'un objet et son servant. De ce fait, le POA permet aux programmeurs, à partir d'une référence d'un objet, un servant ou de l'Identifiant d'un objet, de récupérer les deux autres. Cette gestion se fait avec le POAManager. C'est la partie qui contrôle le POA et principalement la gestion de l'état du POA en matière de traitement des requêtes. Un POAManager peut gérer un ou plusieurs POA, par exemple un POA père et ses fils.



VI.5.3.6. Les Services CORBA

CORBA propose plusieurs services intégrés à sa norme. Ce sont des outils normalisés et centralisés, que l'on retrouve souvent dans les applications distribuées. Par exemple, la plupart des applications sur Internet ont besoin de protection afin d'éviter le piratage, dans ce cas, CORBA propose le service Security. C'est alors l'ORB qui va gérer les liaisons entre les services et les autres parties de la plate-forme et, entre autre, largement faciliter l'implémentation de ces services dans nos applications. Parmi les différents services, on peut citer le service de concurrence d'accès, le service d'événements, le service d'externationalisation, le service de gestion des licences, le service de cycle de vie, le service de nommage, de persistance, de propriétés, de sécurité, de synchronisation, d'interrogation etc.

VI.5.3.7. Les CORBA facilities

Ce sont des utilitaires spécifiés par la norme CORBA. Ils proposent aux programmeurs des outils afin d'intégrer rapidement des fonctionnalités communes à des applications. Citons par exemple les gestionnaires d'impression…

VI.5.3.8. Domain facilities

Les domaines facilities sont des outils particuliers à un domaine précis. Ce sont des objets utilitaires destinés à un secteur d'activité donné. Par exemple dans le domaine de la finance, dans la gestion des comptes, la balance est identique dans un même pays. Pour éviter au concepteur de devoir redéfinir à chaque fois un objet qui gère cette fonctionnalité, il faut vérifier que CORBA ne le propose pas déjà.

VI.5.3.9. Choix d'un ORB CORBA

Il existe un grand nombre d'ORB sur le marché, chacun respectant la norme CORBA, mais à des degrés différents. Nous avons évalué trois ORB : JAVA ORB, ORBACUS, et VISIBROKER.

Java ORB	ORBACUS	VISIBROKER
Gratuit	Gratuit pour usage non commercial	Gratuit pour usage non commercial
Compatible CORBA 2.3 (jdk1.1 à 1.3)	Compatible CORBA 2.3	Compatible CORBA 2.3, supporte POA et RMI over IIOP
Utilisable pour les produits commerciaux	Plateforme supportées : Unix, Windows, …	Plateforme supportées : Unix, Windows
Seulement compatible Java	Très documenté	Le plus connu des ORB
	Licence chère pour utilisation commerciale	Développé par BORLAND (tout comme JBuilder 6 que nous utilisons)
	Compatible C++	Très documenté
	Installation simple	Compatible C++

Installation simple
Les services ne sont pas toujours simples à utiliser
Payant pour un usage commercial
L'un de premiers ORB CORBA

Notre choix s'est porté sur VisiBroker pour différentes raisons. En 1987, l'entreprise Borland, a acquis la société Visigenic SoftWare. Cette dernière était spécialisée dans l'implémentation de CORBA pour son utilisation en tant que développement d'application commerciale. Visigenic proposait alors Visibroker qui était le principal OBR du marché, puis est devenu un standard. Afin de pouvoir utiliser du CORBA dans le programme java, il nous a fallu choisir un fournisseur CORBA. Nous souhaitions utiliser CORBA pour des raisons de facilité de migration d'un langage à un autre et éventuellement pour faire communiquer une plate-forme écrite en java avec une plate-forme écrite en C++. Comme de plus, nous travaillons sous JBuilder et que Borland - Inprise propose Visibroker pour intégrer du CORBA aussi bien dans du java que du C++ tout en étant adapté pour une conception sous JBuilder, notre choix s'est donc porté sur ce produit.

Visibroker est un produit qui a fait ses preuves et qui est fortement appuyé dans le monde de l'informatique. Afin de garantir la longévité de son produit, Inprise propose :

- Le meilleur respect possible des spécifications de l'OMG et des normes CORBA.
- Un serveur d'application Inprise.

Visibroker est portable sur les plates-formes Windows, Solaris HP-UX, RedHat LINUX, Digital UNIX, AIX, IRX, OS390, Vx Works. Les langages pour lesquels Visibroker est prévu sont Java, C++ et Delphi. Il permet bien sûr par l'interopérabilité de CORBA la communication avec d'autres langages comme Smaltalk, Fortran, Lisp, Prolog…

Ce produit offre en plus des fonctionnalités imposées par la norme CORBA, des fonctionnalités ou outils propres à Visibroker. Par exemple pour le service de nom, Visibroker offre des fonctionnalités supplémentaires comme le choix entre contenir l'arborescence en mémoire ou dans une base de données, annuaire LDAP… pour favoriser la persistance des données de ce service. Dans le cadre de la sécurité, Visibroker propose aussi SPKM ou Kerberos…

Attention, l'utilisation de tout produit extérieur à la norme CORBA, peut poser des problèmes lors d'implémentation sous un ORB différent de celui choisi à l'origine. Il est préférable, tant que possible, de se cantonner aux spécifications de la norme.

VI.6. Les JAVA Beans

L'intérêt des JavaBeans est de définir un modèle de composant logiciel. Cela permet d'écrire des composants réutilisables et de les exécuter n'importe où en bénéficiant de l'indépendance de JAVA vis à vis de la plateforme : « write once, run anywhere ». Un bean peut être manipulé dans un IDE sans écrire une ligne de code. Un composant JavaBean est donc un morceau de code compilé et réutilisable qui peut être combiné avec d'autres composants JavaBeans. Les GUI Swing sont un exemple de JavaBeans. Bien souvent les Beans sont utilisés coté client mais ils peuvent l'être aussi coté serveur avec des servlets ou des JSP. Il y a trois aspects fondamentaux des Beans :

- Les propriétés : elles décrivent l'apparence du bean.
- Les méthodes : elles définissent les services fournis par le bean.
- Les événements : ils permettent au bean de communiquer.

Un bean doit, en outre, proposer quelques caractéristiques comme le support pour l'introspection, le support de la customization et le support de la persistance qui permet au bean d'être sauvegardé puis restauré par l'IDE. Les beans sont des objets JAVA comme les autres mais ils possèdent quelques options supplémentaires qui les rendent plus maniables. Les Beans seront utilisés notamment pour la représentation graphique de nos agents.

Le langage JAVA est dynamique et permet, grâce au mécanisme de la réflexion, de découvrir au moment de l'exécution de nombreuses informations sur les objets telles que les méthodes (nom, liste de paramètres, type de retour), ses constructeurs, ses classes parentes… Le fait de découvrir les caractéristiques d'un objet est appelé *introspection*. Cette notion sera utilisée dans notre middleware pour le chargement dynamique des compétences de nos agents

Il est possible d'utiliser un *customizer* pour personnaliser un Bean graphiquement à l'aide d'une boite de dialogue. C'est un composant de l'AWT qui sera placé directement dans une boite de dialogue pour contrôler ses valeurs. Il devra dériver de la classe *component*, implémenter l'interface *customizer* et fournir un constructeur sans paramètre.

Un Bean[46] peut communiquer avec des objets distants via CORBA ou RMI, sans se soucier de leur langage et plate-forme, contrairement à ActiveX de Microsoft qui, par sa limitation au DCOM ne peut accéder aux applications CORBA.

Comparaison JAVA-ActiveX	
JavaBeans	ActiveX
JAVA	C, C++, VB, J++
CORBA, RMI	DCOM
Tout système	Win32
Tout matériel	Matériel basé sur Intel

VI.7. Les Entreprise JavaBeans

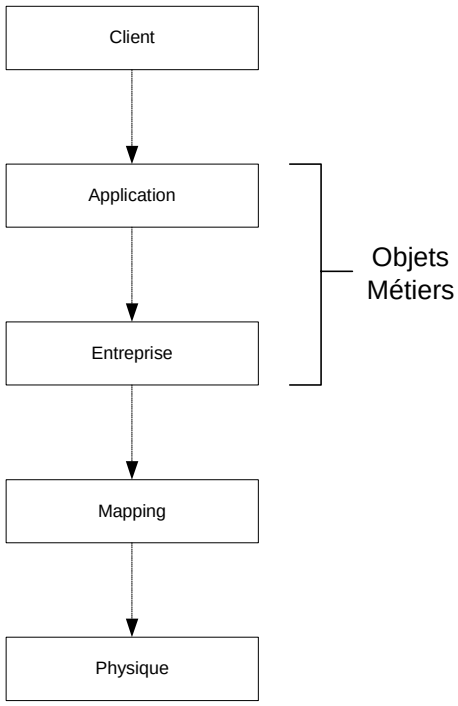
Un modèle de composant coté serveur définit une architecture pour le développement d'objets métier distribués. Ils combinent l'accessibilité des systèmes à objets distribués avec la fluidité de la logique métier utilisant les objets. Les modèles de composants cotés serveur sont utilisés sur les serveurs d'applications dans le tiers intermédiaire qui gère l'exécution des composants et les rendent disponibles aux clients distants. Ils fournissent les fonctionnalités facilitant le développement d'objets métier distribués et les assemblent en solutions d'entreprise.

Ces composants sont exploités sur des serveurs d'applications. Un serveur d'applications est souvent composé de différentes technologies, y compris des serveurs web, des ORB, des MOM (message oriented middleware), des bases de données etc. Un serveur d'applications peut aussi traiter d'objets distribués en technologies CORBA, RMI ou DCOM. Début 1999, Anne thomas du Patricia Seybold Group a inventé le terme Component Transaction Monitor (CTM) pour décrire les serveurs d'applications à base d'objets distribués plus sophistiqués. Les CTM ont évolué sous forme d'un hybride composé des moniteurs TP traditionnels et des technologies ORB. Les EJB se définissent comme un modèle de composants standard côté serveur pour les CTM.

[46] Le site web officiel pour tout savoir sur les beans est http://java.sun.com/beans. Il est intéressant de consulter aussi les newsgroups sur news://comp.lang.java.beans.

Java est une bonne solution pour le développement de solutions d'entreprise particulièrement pour le développement d'applications distribuées du coté serveur. JDBC permet par exemple de fournir une interface JAVA permettant l'accès aux bases de données relationnelles SQL indépendamment du fournisseur. Les abstractions Java pour les technologies des entreprises se sont étendues jusqu'à inclure JNDI (Java Naming and Directory Interface) pour les services d'annuaire, JMX (Java Management Extensions) pour les accès réseau aux périphériques d'un ordinateur, JMS (Java Messaging Service) pour l'accès aux différents produits middleware orientés messages etc.

Les Enterprise JavaBeans sont la dernière abstraction de la famille Java et sans doute la plus ambitieuse. Les EJB simplifient le développement d'applications distribuées en gérant automatiquement les aspects les plus complexes de l'informatique d'entreprise comme le courtage d'objets, la gestion de transactions, la sécurité, le partage des ressources, la connexion aux bases de données, la persistance et la concurrence… Java permet de rendre le programme indépendant de la plate-forme mais EJB va plus loin en proposant une solution pour être en plus indépendant de l'implémentation. L'API JDBC peut s'exécuter sur une machine Unix ou Windows mais on peut aussi accéder à une base de données de n'importe quel fournisseur si elle dispose d'un pilote JDBC. Il n'est pas nécessaire de programmer de façon spécifique l'implémentation à la base de données. Il suffit de changer le pilote pour changer de base de données sans retoucher au code Java. Le principe est le même avec les EJB. Un bean métier peut s'exécuter sur n'importe quel serveur d'applications qui implémente la spécification des EJB. Cela signifie que vous pouvez développer et déployer votre système d'entreprise EJB sur un serveur comme WebSphere d'IBM puis le passer plus tard sur un serveur EJB différent comme WebLogic de BEA.



Le génie logiciel propose de découpler les applications en modules différents ce qui permet une bonne réutilisabilité et une meilleure maintenance. Il faut alors choisir le nombre de couches adéquat pour le développement du projet. Chaque couche présente l'architecture de l'application à un niveau macroscopique, cette notion correspond au « Design Pattern Layer ». (Le Design pattern ou patron de conception, définit une modélisation typique répondant à un problème récurrent dans la conception de logiciels à objets. Les patrons permettent d'anticiper les problèmes d'ouverture et facilitent la compréhension des décisions de conception). Il y a une forte notion de découplage entre les couches ce qui garanti un critère de qualité dans le développement à objet. Une architecture à trois ou quatre couches (couche objets métiers pour couche application + couche entreprise) ne suffit pas forcément car l'inconvénient majeur est que l'architecture à objets métier de base mêle la logique fonctionnelle et les aspects transactionnels. Ces couches sont un regroupement de composants proposant un type de comportement identique. Les avantages de ce modèle s'expriment en terme d'évolutivité, de maintenabilité et de réutilisabilité. Cette décomposition en couche permet d'élever le niveau d'abstraction depuis le format de stockage des données jusqu'à l'interface utilisateur.

Pratiquement, les Enterprise JavaBeans sont donc des composants architecturaux pour les applications Java business orientées vers l'objet de construction distribuée. Un EJB est un

composant (bean) Java prêt à l'emploi qui peut être assemblé afin de former une application distribuée. Cela présente un modèle de programmation simple permettant au développeur de se concentrer sur l'aspect métier. Un serveur EJB convertit un composant en un objet distribué et prend en charge les services cités plus haut. Les EJB sont donc un code compilé réutilisable conçu pour être installé dans une application serveur compatible avec les JAVA2 Enterprise Edition de Sun.

Les EJB sont une véritable révolution pour le modèle client-serveur. Un EJB effectue la logique d'application business comme un composant de serveur qui s'exécute à l'intérieur d'un système d'exécution de composants EJB. Ils prennent en charge toute la logique réseau des autres CORBA, DCOM ou RMI. Les EJB s'inscrivent dans le cadre du développement d'application multi-tiers.

Nous souhaitions développer une plateforme qui réponde à un certain nombre de critères tels que la maintenance et la modularité. Les EJB auraient pu être une bonne solution pour séparer conceptuellement les classes techniques des classes métiers. Ces dernières auraient contenu, « l'expertise » des agents. Cette technologie est particulièrement adaptée du coté serveur et non du coté client. Nous souhaitions cependant que l'agent puisse être à la fois client et serveur pour répondre à une architecture peer-to-peer. De plus, la lourdeur inhérente au développement des EJB et à la mise en œuvre nous a définitivement écartés de l'idée de les employer dans le développement de notre middleware. Nous conserverons l'idée forte des EJB concernant le découplage de l'aspect technique et de l'aspect métier dans la conception du middleware multi-agents. Nous verrons dans le prochain chapitre quelle solution nous proposons pour ce découplage.

VI.8. XML

HTML (Hypertext Markup Language) a été qualifié de langage fondateur du World Wide Web. II propose une manière standardisée de créer des pages d'informations formatées que l'on peut délivrer à une audience sans cesse croissante par le biais de l'Internet. De manière très concrète, HTML - combiné avec HTTP (Hypertext Transport Protocol) - a révolutionné la manière dont les gens envoient et reçoivent de l'information. Mais HTML a été conçu primitivement pour l'affichage. En conséquence, HTML se concentre presque entièrement sur l'apparence de l'information, non sur sa nature ou sa structure. C'est ici qu'intervient XML (Extensible Markup Language : Langage de marquage étendu).

XML est un langage ouvert en mode texte, fournissant une information structurelle et sémantique relative aux données. Ces « données sur des données », ou métadonnées, fournissent une information contextuelle à l'application utilisant les données et elles autorisent un degré plus sophistiqué de gestion et de manipulation des informations basées sur le Web. XML, un dérivé du langage SGML (Standard Generalized Markup Language : Langage de marquage généralisé et standardisé), a été optimisé pour le Web. Ceci contribue à faire de XML un complément puissant et standardisé de HTML, qui pourrait être aussi important pour la diffusion future de l'information sur le Web que l'a été HTML à ses débuts.

XML est destiné aux créateurs de contenus aussi bien qu'aux programmeurs. Comme XML est en mode texte, il peut être lu et utilisé aisément dans les situations non techniques, mais sa capacité à organiser, décrire et structurer des données en fait un outil idéal en association avec des applications hautement techniques. XML fournit ainsi une base commune pour la création de données structurées, préparant et facilitant leur manipulation et leur affichage.

XML est souvent qualifié de langage ; en fait, c'est au sens strict un métalangage. Ceci signifie que XML peut être utilisé pour créer d'autres langages. Ainsi, les vocabulaires, qui sont des langages d'application de XML, peuvent être créés pour satisfaire des besoins ciblés ou résoudre des problèmes. Voici quelques exemples de vocabulaires XML :

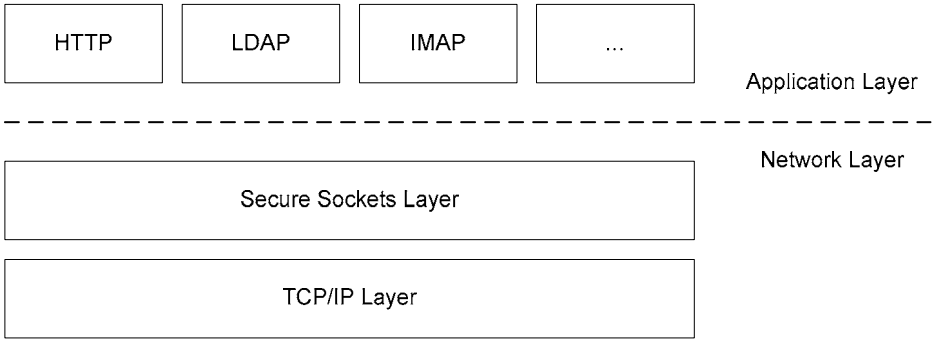
- XSL, un langage de formatage à base de feuilles de style pouvant mettre en forme des données XML et produire une vaste palette de résultats.
- XLL, un ensemble de spécifications qui portent à un haut niveau le mécanisme d'hyperliens de HTML.
- SMIL, un langage de création multimédia standardisé et hautement sophistiqué pour le web.
- XSL Patterns, un langage de requête XML offrant des fonctionnalités de recherches avancées sur les données XML.

LE XML est optimisé pour l'utilisation sur le Web. XML met sur la table un certain nombre de bénéfices que n'apporte pas SGML. XML a la capacité de travailler en HTML pour l'affichage des données et la présentation. XML offre plus d'avantage que SGML dans le cas précis de l'utilisation sur le web. Il est plus compact (les spécifications de XML tiennent en outre sur 35 pages au lieu de 155 pour SGML), XML comporte des spécifications relatives au système d'hyperlien (XLL) et propose même des liens étendus ce que ne fait pas HTML. XML inclut aussi le XSL pour les feuilles de style. Si on utilise la règle classique des 80-20, on peut dire que XML fournit 80% des caractéristiques et des fonctionnalités de SGML avec seulement 20% de sa complexité.

Nous allons utiliser XML pour la définition de fichier de configuration et pour la transmission de données via le protocole SOAP des WebServices, qui sera employée notamment pour le transfert de compétences entre agents.

VI.9. Sécurisation des communications

A l'origine développé par Netscape, le SSL (Secure Socket Layer) a été admis par le World Wide Web pour l'authentification et le cryptage des communications entre clients et serveurs. Mis au point pour sécuriser un autre protocole, le fameux TCP/IP, SSL est utilisé pour chiffrer et authentifier les serveurs. Il entre dans la catégorie des méthodes de chiffrement à clef publique. L'objectif est d'assurer une communication confidentielle et fiable entre un client et un serveur, par exemple, en étant capable de vérifier l'identité du serveur et/ou du client. SSL nécessite un protocole de transport, comme TCP, pour la transmission et la réception des données. SSL est un protocole qui vient se placer au dessus de la couche TCP/IP. Il permet aux couches encore supérieures du modèle OSI (HTTP, LDAP, IMAP, SMTP…) de profiter d'un mode d'accès sécurisé.



VI.9.1. Encodage SSL

SSL peut prendre en charge différents algorithmes cryptographiques pour authentifier et envoyer des certificats, et pour établir des clés. Le sous protocole de négociation a entre autres pour fonction de définir les paramètres du chiffrement : sera choisi celui qui répond le mieux aux besoins et aux outils des deux communicants. Les algorithmes généralement utilisés sont les suivants : DES (Data Encryption Standard), MD5 (Message Digest), RC2 & RC4 (Rivest encryption Ciphers), SHA-1 (Secure Hash Algorithm), SKIPJACK (algorithme de clef symétrique mis en place par Fortezza). Les algorithmes d'échanges de clef, comme RSA (Rivest, Shamir et Adleman) et KEA (Key Exchange Algorithm), sont utilisés par la plupart des chiffrements SSL. Ils déterminent quelle clef symétrique sera utilisée pour l'échange. SSL 2.0 et 3.0 sont des protocoles supportant un certain nombre de chiffrements. Pendant la négociation entre le client et le serveur, ils permettent d'identifier le chiffrement le plus puissant qu'ils possèdent en commun. Ces chiffrements peuvent être activés ou désactivés par un administrateur. Les participants à la négociation choisissent la palette de chiffrements qu'ils vont mutuellement se proposer. Utiliser tous les chiffrements existants assure l'universalité de l'échange. A l'inverse, sélectionner uniquement les chiffrements les plus performants empêche d'être sollicité pour des échanges à risques.

Il y a bien entendu différents algorithmes, fonctionnant avec RSA, du plus sécurisé au moins sécurisés. L'algorithme offrant les meilleures garanties est le 3DES (DES appliqué 3 fois) supportant un chiffrement à 168 bits, couplé avec SHA-1 pour l'authentification des messages. La taille de la clef dans ce cas là est si importante qu'il existe un nombre gigantesque de possibilités de clefs (environ 3.7*10^50). Il est moins rapide mais plus sûr que RC4. Il est supporté par SSL 2.0 et 3.0. Le deuxième algorithme le plus sûr est le RC4 avec un chiffrement 128 bits, couplé à MD5 pour l'authentification des messages - lui aussi supporté par SSL 2.0 et 3.0. Le nombre de clefs possibles est de 3.4*10^38, et RC4 est le plus rapide des modes de chiffrement. Ensuite vient le RC2 avec un chiffrement 128 bits couplé à MD5 pour l'authentification des messages. Le nombre de clefs possibles est le même qu'avec le RC4, mais RC2 est plus lent, et cette méthode n'est supporté que par SSL 2.0. Le DES, par ailleurs, permet un chiffrement à 56 bits couplé avec SHA-1 pour l'authentification des messages. Les 56 bits permettent une probabilité de 7.2*10^16 clefs différentes. Moins performant que RC2, il est cependant supporté par SSL 2.0 (avec MD5 à la place de SHA-1) et 3.0. Enfin les deux algorithmes suivants offrent des garanties moindres, mais restent malgré tout utilisés. Première méthode, le RC4 avec un chiffrement 40 bits, couplé avec MD5 pour l'authentification des messages, est supporté par SSL 2.0 et 3.0. Le nombre de clefs possibles est alors de 1.1*10^12. Seconde méthode, idem sauf que RC2 est utilisé à la place de RC4. Les caractéristiques sont inchangées, si ce n'est que RC2 est moins rapide que RC4. La dernière proposition n'offre que de faibles garanties. L'algorithme n'utilise pas de modèle de chiffrement, les données ne sont pas protégées. En revanche, MD5 permet de détecter les attaques et donc de garantir l'intégrité des données. D'autres algorithmes sont utilisés avec KEA de Fortezza. Ils sont supportés par SSL 3.0 uniquement sur les produits Netscape. Moins nombreux, on en distingue trois différents. Seuls les deux premiers cités offrent des garanties de sécurité acceptables. Le premier est un RC4 avec un chiffrement de 128 bits et couplé avec SHA-1 pour l'authentification des messages. Il est comparable au RC4/128 bits/MD5 cité plus haut. C'est l'un des plus forts après 3DES. Il peut générer 3.4*10^38 clefs probables. Le deuxième est un RC4 à chiffrement SKIPJACK 80 bits couplé avec SHA-1 pour l'authentification des messages. Le dernier, en parallèle au dernier algorithme RSA présenté précédemment, n'offre que peu de garanties. Sans systèmes de chiffrement, il utilise SHA-1 pour l'authentification des messages. Seule l'intégrité des données est garantie.

VI.9.2. Négociation SSL

Les trois fonctionnalités principales de SSL sont l'authentification du serveur, l'authentification du client et le chiffrement des données. Le protocole se compose de deux couches : le SSL Record Protocol concerne l'encodage (envoi des données) et le SSL Handshake protocol la négociation. Parmi les algorithmes de codage, le SSL possède la particularité de combiner l'utilisation de clefs symétriques et clefs asymétriques. Les clefs asymétriques sont mieux adaptées à l'authentification. Le protocole de négociation SSL est en mesure, à partir des clefs publiques et privées du client et du serveur, d'établir une communication entre les deux parties avec une clef « secrète » publique, dont la taille (128 bits) est bien inférieure à celle des clefs publiques traditionnelles (1024 bits). Le processus de négociation SSL comprend huit étapes, avec les messages que s'envoient le client et le serveur qui construisent la communication sécurisée grâce à SSL. Nous n'allons pas les détailler ici mais le lecteur pourra se reporter à un document de conception interne du middleware qui est plus explicite.

VI.9.3. Principales vulnérabilités du SSL

Théoriquement, SSL est vulnérable aux attaques par force brute (tentatives de déchiffrement direct d'un message crypté) si les clefs utilisées sont de 40 bits. C'est pourquoi, il est plus sûr d'utiliser des clefs de 128 bits pour se prémunir des attaques. Les caractéristiques temporelles et financières de ce type d'attaque sont facilement calculables. Des renseignements sur le type de SSL utilisé indiquent le matériel et le temps nécessaires à l'attaque. Le chiffrement à 40 bits est de moins en moins cher et long à attaquer, même pour un pirate seul. Tant qu'il n'existe pas d'algorithmes cryptographiques capables d'attaquer directement le chiffrement 128 bits, ce dernier pourra être considéré comme sûr.

VI.9.4. Cryptage des données

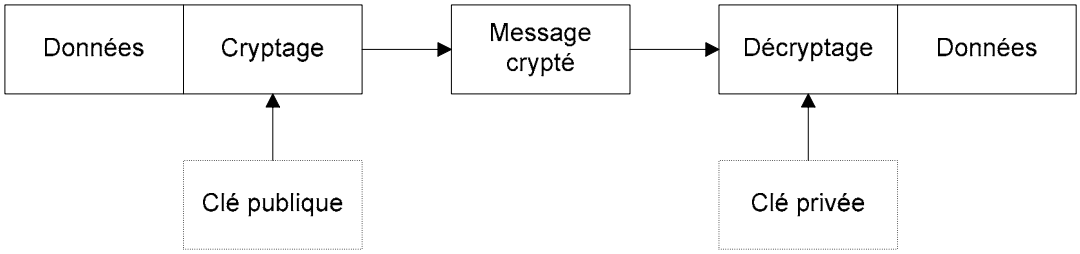
Le premier besoin en matière de sécurité informatique est bien sur le cryptage des données. Pour ce faire, SSL utilise l'algorithme RSA de la société RSA Data Security. Pour la réalisation de cet algorithme, SSL demande l'apport de deux clés extérieures :

- Une clé publique qui est utilisée lors du cryptage des données.
- Une clé privée qui est utilisée lors du décryptage de ces données.

La clé publique sera distribuée à tous les clients autorisés à la consultation du message, quant à la clé privée, elle ne sera détenue que par le serveur. L'algorithme de RSA garantit l'impossibilité de retrouver la clé privée à partir de la clé publique. Le protocole HTTPS (Secured HTTP) est utilisé pour l'encodage et le protocole HTTP s'occupe de l'envoi.

Le protocole HTTPS est supporté par Apache, Netscape, Microsoft IIS, Lotus Domino, Sun Java Web Server. Visibroker propose le serveur GateKeeper pour accepter une connexion SSL activée par HTTPS. Lors d'une communication entre un client et un serveur :

- Le serveur possède les clés publiques et privées.
- Le serveur communique la clé publique au client.
- Le client peut crypter ses messages avec sa clé publique afin de les envoyer
- Le serveur peut décrypter les messages du client avec sa clé privée.



VI.9.5. L'authentification des protagonistes

Le deuxième besoin en matière de sécurité informatique se situe au niveau de la reconnaissance de l'interlocuteur. Le serveur ou le client n'étant pas forcément celui qu'il prétend être. Afin de répondre à cette question, SSL propose un mécanisme d'identification qui s'appuie sur la norme ISO X509. Elle propose l'identification par l'intermédiaire de certificat. Un certificat est constitué de la clé publique à employer, ainsi que d'une signature digitale générée à l'aide d'une clé privée. Un certificat peut être généré par un CA (Certificate Autority) qui joue le rôle de notaire informatique. Par exemple lors d'une tentative de mise en communication entre un client et un serveur :

- le serveur envoie son certificat au client
- le client vérifie le certificat et choisit alors d'accepter la communication ou pas.

Pour une plus grande sécurité, le client peut se faire aider d'un processus tiers qui, lui seul, peut authentifier le serveur, par exemple le CA. Attention le contrôle d'un CA extérieur n'est pas gratuit.

Il est aussi possible d'enchaîner les certificats entre eux. Un certificat peut utiliser sa clé privée pour signer un autre certificat. Les certificats sont ainsi liés entre eux. Pour pirater un certificat, il faut donc pirater l'ensemble des certificats.

VI.10. Les Web Services

Phoenix doit permettre à des agents de se sous-traiter des tâches entre eux et de s'échanger des compétences. Il est possible pour cela d'implémenter un simple système de sockets établissant une communication binaire d'un point à un autre du réseau. C'est globalement le principe du FTP (File Transfert Protocole). Cette solution est simple et facile à mettre en œuvre.

Néanmoins Phoenix doit savoir s'ouvrir vers d'autres systèmes informatiques afin de récupérer des données. Phoenix peut par exemple être amené à collecter des données d'un logiciel de CRM pour effectuer des tâches statistiques sur les clients, comme un profiling. Autrefois on utilisait un format d'échange assez standard appelé communément l'EDI pour Echange de Données Informatiques. Aujourd'hui, les logiciels sont de plus en plus ouverts sur le Net pour faciliter une migration des données. Les éditeurs de logiciels ont ainsi « Webifié » leurs applications en les dotant d'une interface de communication XML. Le protocole du Web étant le HTTP, une norme fut créée pour permettre la communication de fichiers XML via ce protocole HTTP. Cette norme s'appelle SOAP et c'est celle utilisée dans les WebServices.

Les Web Services sont donc accessibles via le port 80 du serveur web. Nous avons décidé d'adopter cette norme dans Phoenix pour lui permettre :

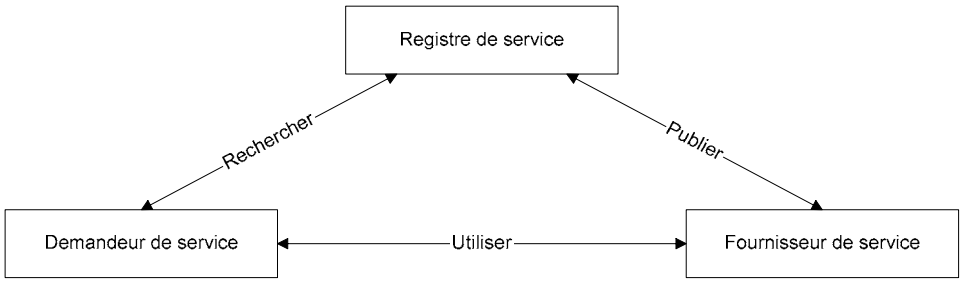
- communiquer avec d'autres logiciels d'entreprise
- échanger des compétences entre agents
- Permettre l'exécution d'un service à distance via le protocole Web

En outre, il y a un autre intérêt à utiliser les *web services* dans Phoenix qui est l'interopérabilité. Phoenix doit pouvoir être porté sur un autre environnement que JAVA, par exemple le MS .NET. Microsoft utilise intensivement les *web services* (même les appels locaux en .Net sont des *web services*).

Le *web service* est une interface standardisée décrivant des opérations accessibles en réseau par XML standard. Cette interface doit cacher l'implémentation de telle façon qu'on soit capable d'utiliser les services proposés sans se préoccuper du langage de programmation ou du matériel. Il doit être muni de 3 composants importants :

- Un service est invocable (normalement par SOAP)
- Un service est décrit (normalement par WSDL)
- Un service est déclaré / enregistré auprès d'un registre (normalement UDDI)

L'architecture des *web services* en général est la suivante :

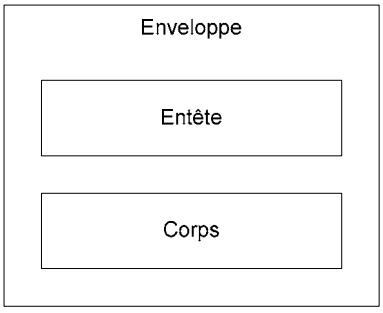


Architecture de *web services*

Le fournisseur de services a la responsabilité de décrire ses services et de les publier aux registres de services. On peut considérer le fournisseur comme une sorte de « serveur » dans l'architecture client-serveur. Le demandeur de service, quant à lui, a la responsabilité de chercher la description de service dans le registre et de l'utiliser (invoquer ces services). C'est la partie « client » de notre architecture. Le registre de services publie les descriptions des services. Dès que le client trouve le service dont il a besoin, le rôle du registre n'est plus demandé. Nous allons à présent décrire les trois protocoles utilisés dans les Web Services.

VI.10.1. SOAP

SOAP (*Simple Object Access Protocol*) est une technologie permettant le RPC (*Remote Procedure Call*) en utilisant XML et des protocoles internet standard (http, smtp etc. Il faut noter que le http est le plus souvent utilisé). Un RPC permet d'invoquer des méthodes dans un objet à distance. Le grand avantage de SOAP est sa simplicité car c'est un RPC en mode texte (contrairement au CORBA, par exemple, qui est en mode binaire). Le fait qu'il puisse travailler en utilisant le http lui permet de traverser les *firewalls* sans problème.



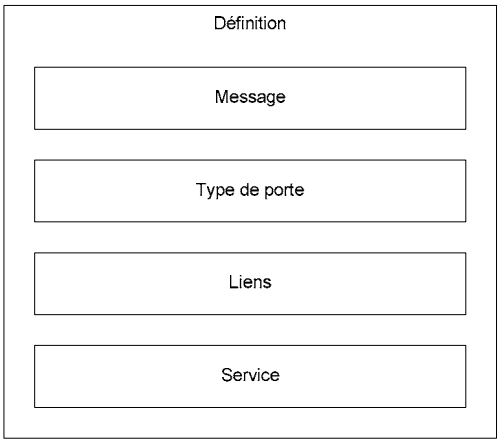
L'inconvénient de SOAP est sa sécurité. La façon standard de sécuriser une communication SOAP n'existe pas. La simplicité de SOAP entraîne aussi quelques limites de fonctionnalités (il n'y a pas de passage par référence etc.).De plus, la nature textuelle de SOAP gonfle les requêtes et ralentit la communication. La requête SOAP est composée de deux éléments : Le corps et l'entête. Dans le corps, on peut retrouver les éléments de la requête locale (le nom de la

méthode, le type de paramètre, le nom du paramètre etc.). L'entête donne des liens vers l'encodage de type. Par exemple, le type *vector*<…> en C++ et *Vector* en Java pourrait être codé de la même façon malgré la différence de nom et d'implémentation. Cette information pourrait être mise dans l'entête. La réponse de cette requête sera de la même forme mais avec une balise XML indiquant que c'est une réponse. Elle contient le retour de la méthode. (Un exemple de requête SOAP et la réponse se trouvent dans l'annexe Exemple de SOAP, WSDL)

VI.10.2. WSDL

WSDL (*Web Services Definition Language*) est un document XML permettant de décrire une interface SOAP. La description est faite en trois parties :

- Que fait le service ? – le nom de la méthode.
- Comment accède-t-on au service ? – le type de données, protocoles etc.
- Où se trouve le service ? – une adresse IP ou un URL



Un fichier WSDL possède une structure telle que décrite sur la figure ci-contre. La partie « définition » donne le nom du *web service* ainsi que quelques autres informations (codage de type comme dans SOAP, etc.). La partie « message » donne l'information spécifique sur le service : le nom de la méthode, les paramètres, le type de retour etc. Le « type de porte » va nous donner le type de service, soit une requête/réponse, un « aller-simple » où il n'y a pas de réponse etc. Dans le « liens » nous allons décrire comment se connecter au service : les protocoles utilisés. Le « service » va nous donner où il se trouve. D'habitude, le fichier WSDL est mis dans un registre; en interrogeant ce registre, on va savoir où se trouve réellement le service (Un exemple de fichier WSDL est fournit dans l'annexe Exemple de SOAP, WSDL).

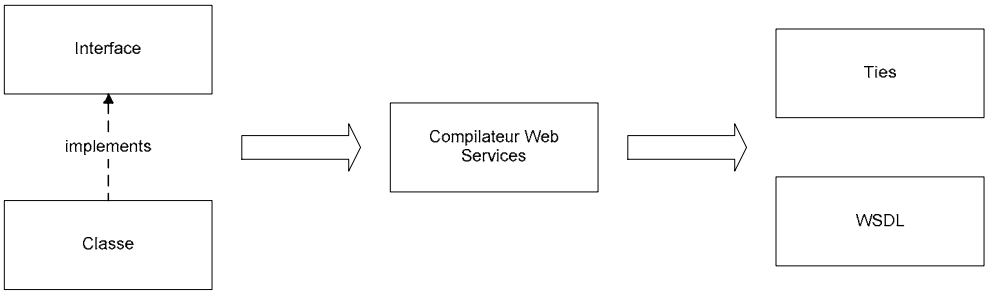
VI.10.3. UDDI

UDDI[47] (*Universal Description Discovery and Integration standard*) est une norme décrivant un registre de services. Le contenu de ce registre consiste en des descriptions de services (par exemple, des fichiers WSDL). Il intègre aussi une fonction de recherche. Pour enregistrer un nouveau service auprès d'un registre UDDI, il suffit normalement de remplir un formulaire html et en suite d'envoyer le ficher WSDL au registre. On utilise SOAP pour interroger ce registre. Les protocoles utilisés dans ce registre sont assez compliqués et ne seront pas détaillés ici.

VI.10.4. Création d'un service (côté serveur)

L'action d'exposer un objet en tant que *web service* s'appelle « le déploiement ». La plupart des outils de déploiement des *web services* vont créer à la fois le service et le fichier WSDL décrivant le service. Pour déployer un objet, on a besoin de sa classe ainsi que de son interface. On va compiler (avec un compilateur spécialisé) le code source de la classe pour créer un fichier WSDL ainsi que des *ties* (nous verrons par la suite l'utilité de ces *ties)*. La figure suivante résume la création de fichier WSDL et les *ties.*

[47] Plus d'information sur http://www.uddi.org et http://uddi.microsoft.com/default.aspx



Le fichier WSDL créé va être distribué dans les registres UDDIs ou donné directement aux clients.

VI.10.5. Utilisation d'un service (côté client)

Pour utiliser les services, le client doit d'abord créer des *stubs* (nous verrons ensuite l'utilité du *stub*). Pour cela, le client doit récupérer le fichier WSDL (soit directement du serveur, soit à partir d'un registre UDDI) et le compiler avec un compilateur spécialisé.



Après la création des *stubs*, le client peut les utiliser pour interroger les *web services*. Le serveur va alors utiliser des *ties* pour recevoir les requêtes venant du client et pour y répondre. Les requêtes/réponses se font à travers les APIs (bibliothèques) des *web services*. Les APIs côté serveur doivent comporter un serveur web équipé soit avec un conteneur de servlet pour des serveurs JAVAs soit par des CGIs (*Common Gateway Interface* : Interface commune vers les programmes situés dans le serveur web) pour d'autres type de serveurs (Un exemple de client *web service* est mis dans l'annexe Exemple de SOAP, WSDL).

VI.10.6. Sécurité des web services

Pour que Phoenix puisse communiquer de façon sécurisée, nous devons implémenter un modèle de sécurité dans la partie *web services.* Notre choix s'est porté sur celui utilisé dans la plupart des transactions web : le SSL. La transaction est encryptée avec un algorithme RSA plus la possibilité d'authentification.

Le cryptage va nous garantir que la transaction n'est pas lisible par les parties non concernées. L'authentification nous garantit que notre interlocuteur est celui qu'il prétend être et n'est pas un intrus. L'authentification est faite en utilisant un certificat fourni par une tierce partie (un « *Certificate Authority* » ou CA. Exemple : Verisign).

VI.10.7. Choix d'une architecture de serveur web

Nous cherchions une solution à faible couplage entre Phoenix et le serveur Web bien qu'un WebService étant dépendant d'un serveur web. Nous avons envisagé trois alternatives d'architecture pour la mise en oeuvre d'un serveur web intégrant des *web services* dans Phœnix. La première consiste en un serveur isolé, la deuxième en un serveur intégré et la troisième en un serveur développé « maison ». La première avait le désavantage d'être non transparente pour l'utilisateur et nécessitait un administrateur système pour déployer les services utilisés par Phoenix. L'architecture présentant un serveur intégré est conçue de façon à ce que le serveur soit géré par la plate-forme Phœnix. Le démarrage/arrêt et le déploiement des services font donc partie des fonctionnalités de Phœnix. Cette solution est transparente pour l'utilisateur mais impose davantage de développement pour notre plate-forme. La dernière nous impose le développement du serveur web. Cela revient à intégrer le serveur web

directement dans Phoenix. Le démarrage/arrêt du serveur et le déploiement des services sont fournis dans Phœnix d'une façon transparente pour l'extérieur. L'avantage de cette solution est sa totale transparence car on peut même « cacher » le serveur web dans Phœnix et on possède la maîtrise totale du serveur web. Cette solution est la moins chère pour l'utilisateur final mais impose un développement supplémentaire pour les concepteurs de Phoenix. En outre, aucune intervention humaine n'est nécessaire. L'inconvénient est la complexité du développement d'un serveur web gérant le protocole SOAP. Cela revient à créer notre propre système RPC. Il faut aussi s'intéresser de près aux problèmes de sécurités potentiels engendrés et de la rapidité d'exécution…

Nous avons évalués quelques serveurs web[48] en fonction de certains critères comme la simplicité pour les développeurs et les utilisateurs, le SOAP avec attachement, la sécurité, le coût, la fiabilité, la taille et l'éditeur. Nous avons évalués WASP de Systinet, Apache SOAP et Tomcat, GLUE de "The Mind Electric".

Critères	**WASP**	**Apache SOAP**	**GLUE**
SOAP avec attachement	Il existe un bug mais on arrive à le contourner	Problème d'implémentation mais on arrive à le contourner	Fonctionne correctement
Sécurité	Supporte SSL en édition *Advance*	Ne supporte pas SSL dans notre implémentation (« serveur intégré »). Supporte le SSL dans l'architecture « serveur à part »	Fonctionne correctement
Coût	Gratuit pour l'édition Lite. Payant pour l'édition *Advance*	Gratuit / open source (licence Apache)	Gratuit pour l'édition standard. Payant pour l'édition *Advance*.
Bugs	Bug sur l'attachement	Aucun bug n'est trouvé.	Aucun bug n'est trouvé.
Taille	8 M	6 M	2 M
Simplicité pour les développeurs de Phoenix	Assez facile	Difficile	Facile
Simplicité pour les utilisateurs finaux	N'est pas transparent	Transparent pour la solution « serveur intégré »	Transparent
Entreprise	Très établie	Groupe open source	Etablie
Points forts	**L'entreprise, plug-in JBuilder**	**Open source**	**Facilité d'utilisation, plug-in JBuilder, coût, transparence, taille,**
Points faibles	**Des bugs**	**Difficile à utiliser**	**Aucun trouvé**

VI.11. La communication et le partage de ressources

Les grands réseaux locaux offrent aujourd'hui la disponibilité de nombreuses ressources de calcul et de stockage. Internet est lui-même un réseau de communication

[48] Vous pourrez trouver davantage de documentation sur les sites des éditeurs : http://www.themindelectric.com/glue/index.html (GLUE), http://xml.apache.org/soap/ (APACHE-Soap) et http://www.systinet.com/products/wasp_lite/ (WASP)

suffisamment efficace et stable pour véhiculer de grands volumes de données. Cela a donné naissance au concept de SDGE (Systèmes Distribués à Grande Echelle). Les projets et les infrastructures émanent de différentes communautés et peuvent être classées en trois catégories : les systèmes GRID (Globalisation des Ressources Informatiques et des Données), les systèmes de calcul global et les systèmes de partage de ressources entre pairs (peer-to-peer) [Cappello, 2002]

Le métacomputing [Smarr & Catlett, 1992] et les systèmes de Grid [Foster & Kesselman, 1998] étaient à l'origine développés pour des besoins de calcul scientifique (projet EuroGrid, basé sur Unicore) ou dans un objectif de stockage massif (projet DataGrid basé sur Globus). L'ambition est de définir des infrastructures permettant de faire interopérer des ressources de calcul, des grands instruments de mesure, des bases de données et des centres de visualisation et de réalité virtuelle. A cela s'ajoute la facilité d'accès à ces ressources et son organisation. Au même moment, le système Globus était développé pour relier ensemble des supercalculateurs américains et construire ainsi un calculateur parallèle encore plus puissant. Globus s'adresse essentiellement aux couches basses, voire systèmes, des logiciels et des réseaux.

Les systèmes de Global Computing (ou calcul global) impliquent la participation de volontaires particuliers trouvant un intérêt social dans une expérience spécifique comme le généthon, la lutte contre le cancer, ou la recherche de signaux extra-terrestres (projet SETI). Le modèle de calcul est un parallélisme massif nécessitant peu de communication entre les sites.

Les systèmes de partage pair à pair concernaient jusqu'aujourd'hui principalement les internautes cherchant à échanger des fichiers multimédia. Dans de tels systèmes, toutes les machines remplissent à la fois le rôle de client et de serveur, d'où le terme de ServEnt dans Gnutella [Kan, 2001]. Les ressources partagées peuvent être de différentes natures, espace disque, puissance CPU, données…

Aujourd'hui on a tendance à confondre ces différentes appellations et on constate bien souvent dans la littérature que les systèmes de Calcul Global sont assimilés à des systèmes de Pair à Pair [Barkai, 2001]. Un véritable système pair à pair fonctionne à la fois comme client et serveur. Les systèmes de calcul global comme SETI@Home, Entropia, UnitedDevice ne respectent pas cette propriété, contrairement à XtremWeb[49], ActiveCluster de Platform et à tous les systèmes d'échange de fichiers pair à pair connus [Capello, 2002]. Une des caractéristiques des systèmes P2P est, qu'une fois l'étape de mise en relation effectuée, les données sont transmises sans intermédiaire entre les pairs. Ce n'est par exemple pas le cas pour FreeNet [Langley, 2001], où la transmission de l'information est réalisée par l'infrastructure de mise en relation. Cette propriété est aussi absente dans les systèmes de Calcul Global.

Certains systèmes comme SETI, Napster et XtremWeb utilisent donc une architecture centralisée. Napster propose en effet un répertoire centralisé pour permettre aux clients d'identifier des serveurs potentiels. Pour SETI et XtremWeb, l'ordonnanceur de calcul et le collecteur de résultats sont aussi centralisés. A l'inverse, les architectures P2P comme Gnutella, Freenet et fastrack utilise une infrastructure distribuée (backbone) pour permettre la mise en relation des clients et des serveurs.

Ces systèmes sont en quelque sorte auto-organisés car on trouve la présence d'une reconformation dynamique et totalement distribuée. Cela est attribué au principe de communauté dynamique des systèmes de Calcul Global ou de P2P. Cette propriété fait

[49] XTremWeb site : www.xtremweb.net

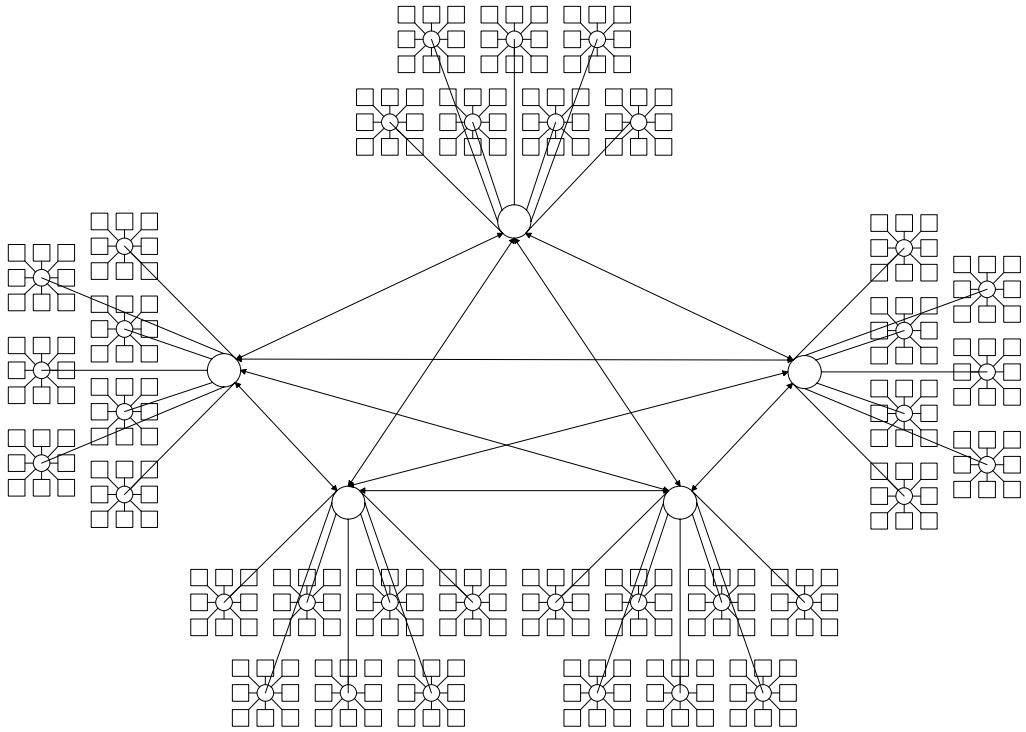
référence à la possibilité pour tous les nœuds de rejoindre ou de quitter à tout moment le système.

VI.12. Le grid computing

GRID, "Globalisation des ressources informatiques et des données", est le nom d'un programme d'action scientifique (action concertée incitative ou ACI) mis en place en février 2001 par le ministre de la Recherche, Roger-Gérard Schwartzenberg. Son objectif est de mettre en réseau sur Internet la puissance de calcul et de stockage des ordinateurs installés. Ce "super web" vise à répondre aux besoins des scientifiques en capacité de stockage et de calcul. L'action GRID constitue un véritable moteur d'expérimentation qui pousse la recherche dans les environnements de programmation distribués, dans les réseaux à très haut débit comme dans les nouvelles interfaces d'accès en langage naturel.

Le projet GRID du ministère de la Recherche a pour objectif le développement des activités de recherche dans ce domaine pour inventer et mettre au point de nouveaux modèles. D'autres projets sont eux aussi soutenus par le réseau national de recherche en télécommunications (RNRT) (par exemple VTHD, qui développe une infrastructure de réseaux à très haut débit en vue d'une utilisation en metacomputing) et le réseau national des technologies logicielles (RNTL) (e-Toile, développement de technologie Application Service Provider, par exemple)

L'objectif est de contribuer à la création des outils logiciels permettant la mise en œuvre d'un "réseau ambiant", rendant l'accès totalement transparent à des ressources informatiques potentiellement illimitées. Les infrastructures de type grille de calcul vont permettre de nouveaux usages comme par exemple simuler de façon précise des objets physiques complexes nécessitant des calculs lourds et visualiser leur comportement en interactions. La programmation de ces applications sur ce type d'infrastructure est difficile puisque les ressources informatiques sont distribuées (géographiquement) et sont aussi parallèles (supercalculateurs). Le projet Globus permet par exemple de masquer la nature distribuée de ces ressources en considérant la grille comme un supercalculateur parallèle virtuel dont la programmation est assurée à l'aide de bibliothèques de communication par échange de messages. Cependant, Globus ne permet pas de traiter facilement toutes les applications qui pourraient utiliser une grille. Une approche, qui sera développée dans le cadre de l'ACI GRID, est de combiner les modèles du calcul parallèle (à base de tâches et de messages) et les modèles du calcul distribué (à base d'objets distribués Java et de composants logiciels Corba). Dans cette approche, une application pour la grille est un ensemble d'objets ou de composants logiciels en interaction, ces objets pouvant être eux-mêmes des programmes parallèles.



La figure ci-dessus représente une vue logique d’une grille. Les liaisons entre les cinq nœuds centraux se font physiquement par un réseau comme Internet en passant par des routeurs etc.

Une approche de plus haut niveau se base sur le concept d'ASP (Application Service Provider) dans lequel des clients accèdent à des ressources logicielles (et non plus matérielles comme dans les précédentes) à distance. Il s'agit de déployer un ensemble de serveurs de calcul sur les ordinateurs d'une grille de calcul et d'en offrir l'accès à des clients répartis sur le réseau. La localisation de ressources et l'équilibrage de charge entre les serveurs se font grâce à un ensemble d'agents eux-mêmes disposés à des emplacements stratégiques sur la grille. L'objectif est de permettre l'accès à des machines à fortes puissances de calcul et à grande capacité mémoire à distance sans forcément être un expert du parallélisme, sans avoir à installer sur sa machine des logiciels compliqués aux interfaces hermétiques, et en pouvant utiliser des codes et des données confidentiels sur les serveurs de ceux qui les développent, ceci de manière sécurisée.

Pour gérer ces grilles, il faut des systèmes permettant d'accéder à de multiples sources de données préexistantes, autonomes et potentiellement hétérogènes. Offrir un accès transparent à un ensemble de ressources passe par la définition d'une infrastructure qui rend la complexité de l'architecture sous-jacente transparente à l'utilisateur en offrant des facilités de conception, d'intégration, d'interrogation et d'administration d'une manière fiable et efficace entre les applications.

Les applications des grilles de calcul et de données se retrouvent dans tous les domaines scientifiques : physique des particules, climatologie, astronomie, sciences du langage, biologie et génomique, chimie, mécanique, géologie et géophysique, cryptologie, optimisation combinatoire, planification, etc.

C'est aussi un des buts de l'ACI GRID que de constituer des consortiums pluridisciplinaires permettant de résoudre des problèmes nouveaux ou d'une taille considérable.

Nous allons reprendre le concept du Grid dans notre middleware, puisqu'un agent aura la possibilité de sous-traiter à un autre agent plus performant dans le domaine. Nous envisageons également de créer une interface permettant à Phoenix de s'appuyer sur des logiciels de Grid existants. Pour cela, nous proposons de respecter des protocoles standards du Grid qui sont les standards OGSA (Open Grid Services Architecture). Chaque élément de la grille fournit un ensemble de ressources, chaque ressource est considérée comme un service de la grille (GridService). Un GridService est définit comme un web service qui fournit un ensemble d'interfaces bien définies et qui suit des conventions spécifiques. Les services principaux sont : les factorys (usines) responsables de la création dynamique des services, le registre qui est une sorte de « pages jaunes » contenant les descriptions des services, y compris les factorys, et le mapper qui fait correspondre à chaque description de service un identifiant (handle) retourné par un factory.

- **GRAM** : Les protocoles GRAM (Grid Resource Allocation and Management) permettent la création et la gestion sûre, fiable des jobs à distance.
- **MDS :** Les protocoles MDS (Monitoring Discovery Service) permettent la découverte et la gestion de l'information sur les ressources impliquées dans la grille.
- **GSI :** Les mécanismes du GSI sont utilisés pour l'authentification, l'autorisation et la délégation pour le calcul distant.

Nous verrons dans le chapitre présentant l'architecture de Phoenix, quel modèle nous proposons pour nous interfacer avec les systèmes de Grid, bien qu'en soit, Phoenix ait nativement des fonctionnalités de Grid.

VI.13. SMP

SMP est l'abréviation de "*Symmetric Multiprocessing*". Il s'agit d'une architecture informatique qui fournit de très bonnes performances en utilisant plusieurs microprocesseurs pour exécuter des processus individuels complets. Contrairement au traitement asymétrique, tout processeur inactif peut se voir attribuer des tâches et les microprocesseurs additionnels peuvent être ajoutés pour améliorer les performances. Le SMP est rendu possible que lorsque le code utilise le multithreading (développement de l'application avec des processus légers).

Le SMP utilise un seul OS (Unix, Windows NT…) et partage une mémoire commune et les ressources disques (I/O). A noter que tous les OS ne supporte pas le SMP.

> *Attention à ne pas confondre le « Symmetric MultiProcessing » avec le « Simple Management Protocol », qui est un autre acronyme pour le SNMP2, lui même étant une version améliorée du SNMP (Simple Network Management Protocol) [NDLR]*

Le logiciel Phoenix reprendra simplement une idée de ce concept puisque les agents peuvent utiliser un système de délégation de traitement pour effectuer leurs tâches.

VI.14. L'Autonomic Computing

L'Autonomic Computing est un nouveau concept proposé par quelques grands éditeurs ou constructeurs informatiques. Le terme "autonomic nervous system », tiré de la biologie, est traduit en français par le système « sympathique ». Il gère la plupart des organes et des muscles du corps humain. Son fonctionnement est totalement involontaire et dépend d'un certain nombre de variables comme la température, le stress, le taux de sucre… Il y a

cependant une corrélation étroite entre les différentes parties de l'ensemble et des boucles de rétroaction.

Pour ce qui nous concerne, cela consiste à traiter les problèmes de complexité d'intégration des nouvelles technologies pour permettre à un intégrateur de réduite ses coûts et ses délais. C'est dans la continuité des systèmes de composants métiers tels que les EJB. Le principe global est l'auto-gestion d'un système informatique. Les laboratoires d'IBM travaillent sur ce sujet depuis 2001 [ACIBM, 2003]. Cette approche d'une informatique massivement réparties semble être vouée à un bel avenir.

Un système d'Autonomic Computing est auto-configurable, auto-optimisable, auto-protégeable et auto-réparable. Il possède les caractéristiques suivantes :

1. Il a besoin de se connaître lui-même. Ses composants doivent aussi posséder une identité. Il doit connaître leur état instantané, leur capacité ultime et toutes les connexions avec le reste du monde. Il doit connaître les limites des ressources sous son contrôle, celles qu'il peut prêter et emprunter, celles qui peuvent être partagées ou doivent être isolées
2. Il doit se configurer et se reconfigurer automatiquement lui-même dans des conditions variables et imprévisibles
3. Il ne s'installe jamais dans le statu quo - il cherche toujours à optimiser son travail en surveillant ses composants et en affinant le flux de travaux pour remplir ses missions
4. Il doit être capable de se "guérir" et se sortir de situations de crise prévisibles ou non. Il doit découvrir les problèmes réels ou potentiels et en isoler les causes par reconfiguration dynamique
5. Il doit être expert dans l'auto protection, en détectant, identifiant les menaces et en s'en protégeant pour maintenir sécurité et intégrité
6. Il doit connaître son environnement ainsi que le contexte de son activité et agir en conséquence. Il doit trouver et générer les règles optimisant son interaction avec les systèmes voisins
7. Il ne peut exister dans un environnement hermétique : il fonctionne selon des standards ouverts dans un monde hétérogène
8. Il doit anticiper les besoins en ressources optimisées tout en masquant sa complexité à l'utilisateur qui ne doit connaître que ses besoins en matière de gestion.

De part sa nature, notre plate-forme s'inscrit comme un des premiers systèmes d'Autonomic Computing. Elle s'adapte dynamiquement au changement et propose ses services sur réseau via des systèmes standards et ouverts. Le développeur de « skills » ne se préoccupe plus de l'infrastructure matérielle ou logicielle, il développe simplement des composants métiers adaptatifs.

VI.15. Le P2P

VI.15.1. Une nouvelle philosophie d'architecture

Le P2P est l'acronyme de Peer To Peer ou « pair à pair » en français. Il s'agit d'un modèle d'architecture positionnée comme le pendant de la célèbre architecture client-serveur pour laquelle certains ordinateurs sont dédiés pour servir les autres. L'encyclopédie online

Webopedia[50] définit le P2P comme un type de réseau dans lequel chaque machine a des capacités et des responsabilités sensiblement équivalentes.

Dans un modèle P2P, chaque machine est en effet à la fois cliente et serveur. Par extension, il s'agit d'un réseau d'échange qui permet la mise en relation de milliers d'ordinateurs pouvant ainsi partager leurs données. Il rend possible la récupération directe sur le disque dur d'un utilisateur connecté du ou des fichiers sélectionnés.

Les programmes les plus connus qui ont contribués à populariser le P2P sont ICQ (www.icq.com), SETI (www.seti.org) et les programmes d'échange de fichiers comme Napster, WinMx, LimeWire, KaZaa et Grokster… Ces derniers sont utilisés comme moteurs de recherche. Ils proposent une liste de toutes les liaisons correspondant à la demande formulée. Il ne reste qu'à télécharger le fichier choisi. Napster s'était originellement spécialisé dans le partage de fichiers de musique MP3.

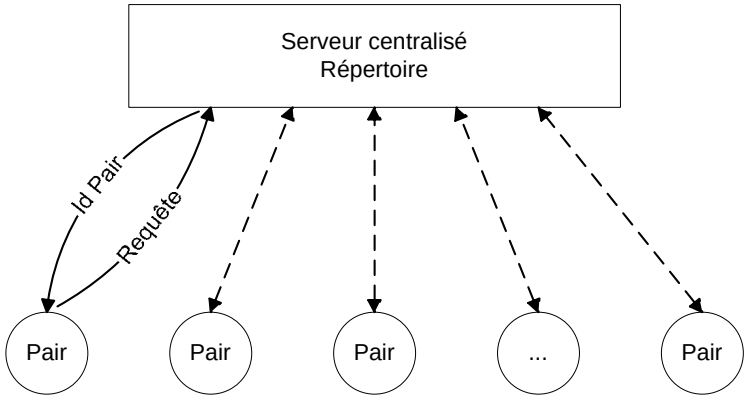
Cet usage ne permet néanmoins pas de rendre compte de toutes les potentialités du modèle. En effet, le modèle P2P permet de répondre à certaines insuffisances des modèles traditionnels. Entre autres problèmes, on peut citer : le partage des traitements, la montée en charge dans les serveurs centraux, l'utilisation plus efficace de la bande passante… le P2P offre les fonctionnalités nécessaires à la réalisation des applications totalement ou partiellement distribuées.

Le vrai changement se situe dans la nature des éléments qui constituent les réseaux peer-to-peer. Dans le passé, les centaines de millions d'ordinateurs connectés à Internet de manière intermittente ne faisaient pas partie du réseau. Ils n'avaient pas d'adresse IP fixe sur le réseau connue dans les DNS. Peer-to-peer désigne donc une classe d'applications qui tire partie des ressources matérielles ou humaines qui sont disponibles au bord de l'Internet. Comme ces ressources ont une connectivité instable ou des adresses IP variables, elles fonctionnent de manière autonome, indépendamment de systèmes centraux comme les DNS. Ce qui a rendu Napster et des systèmes similaires populaires, c'est le fait de tirer partie de ressources qui étaient auparavant inutilisées en tolérant et même en tirant partie d'une connectivité aléatoire. Un vrai système peer-to-peer se reconnaît donc par la faculté de chaque pair de se connecter de manière intermittente avec des adresses IP variables et d'âtre dotée d'une autonomie significative ?

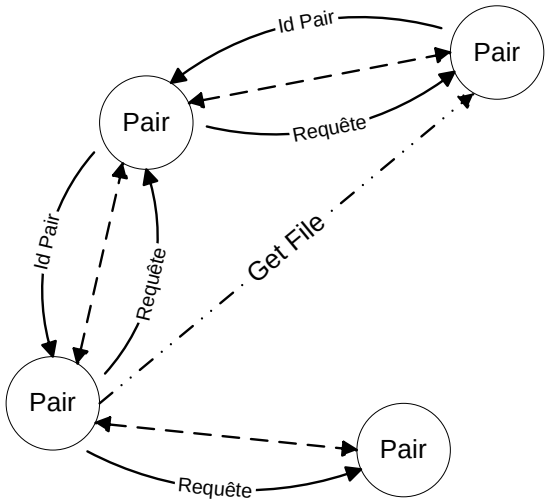
Une autre manière de distinguer un système peer-to-peer est de raisonner en terme de "propriété". Il faut remplacer la question "Est-ce que les pairs peuvent échanger des informations ?" par la question "Qui possède les ressources qui font tourner le système ?". Dans un système comme Yahoo!, l'essentiel des ressources est possédé par Yahoo! tandis que dans un système comme Napster, l'essentiel des ressources est possédé par les utilisateurs de Napster. Dans les vrais systèmes peer-to-peer, on ne se contente pas de décentraliser les fonctions mais aussi les coûts et les charges d'administration.

Voici une figure schématisant le fonctionnement d'une recherche avec un système comme Napster :

[50] Encyclopédie Webopedia : www.webopedia.com



Dans Napster, le système de recherche de ressources est centralisé. Il repose sur un répertoire stockant des relations « ressource-pair ». Le client envoie une requête au répertoire. Le répertoire recherche la ou les adresses IP des machines correspondant à la ressource demandée. Enfin le répertoire renvoie l'IP au client. Voici maintenant comment fonctionne un système du type Gnutella :



Le système de recherche de Gnutella est différent. Il fonctionne de façon purement distribuée. Un pair est connecté à un ensemble limité de pairs voisins logiques identifiés par une adresse IP. Lorqu'un pair recherche un document, il diffuse une requête à ses voisins logiques. Les voisins propagent eux-mêmes à leurs voisins. Le protocole agit donc par une inondation des nœuds (pairs) présents dans le système. Lorsqu'un pair reçoit la requête et possède le document recherché, il ne propage plus la requête et envoie une réponse à l'émetteur de la requête. La réponse suit le parcours inverse de la requête, en se propageant de pair en pair jusqu'au destinataire. Il y a bien entendu un mécanisme qui permet de limiter l'inondation : chaque requête possède une durée de vie (un compteur de saut) qui est décrémenté à chaque fois qu'un pair propage la requête. Un autre mécanisme permet aussi d'éviter les boucles dans le protocole d'inondation.

Freenet[51] possède un système similaire mais toute ressource est cryptée et identifiée avec une clé unique. Le système de propagation de requête ne transmet la requête au voisin possédant une clé proche de la clé recherchée. Une clé peut représenter le cryptage du contenu de la ressource entière ou d'un ensemble de mots clés identifiant la ressource.

Pour un calcul très simplifié, considérons 100 millions de PC connectés à Internet avec une puissance de 100Mhz et 100Mo de disque partagé. Ils représentent une puissance de dix milliards de Mhz et dix mille téra-octets de stockage. C'est de ces ressources dont les programmes comme Napster ou SETI essaient de tirer parti.

VI.15.2. Avantages et inconvénients

Les avantages immédiats du P2P sont :

[51] Site web du projet Freenet : http://freenet.project.free.fr

- réduction des coûts : inutile de bâtir une architecture client / serveur, chaque poste joue simultanément ces rôles
- indépendance vis-à-vis d'une architecture centrale et des machines
- mise ne oeuvre instantanée : il suffit simplement d'installer un programme
- pas de problème de stockage : les informations sont sur les disques durs des autres ordinateurs du réseau
- plus rapide : le traitement est réalisé en local
- mise à disposition de l'information directement à sa source

Il présente néanmoins quelques contraintes :

- dépendant de la machine de l'internaute résolument altruiste
- nécessite une ligne à haut débit, une bonne bande passante pour une utilisation confortable

VI.15.3. Vers des protocoles P2P standard ?

Il y a plusieurs façons de construire un protocole peer-to-peer. Il est bien entendu recommandé d'adopter des technologies standards. Les bases sont les protocoles réseau TCP et HTTP. Certains systèmes ont même été construits par-dessus le protocole de messagerie SMTP. Le mode d'appel par procédure distante a rapidement été utilisé. XtremWeb, par exemple, fut programmé au dessus de Java RMI ou JINI. Depuis, d'autres protocoles furent proposés, notamment SOAP qui encapsule des RPC (Remote Procedure Call) encodées au format XML dans des messages HTTP. Napster, l'un des pionniers du P2P utilisait un protocole réseau propriétaire qui n'était néanmoins pas basé sur XML.

Il existe aujourd'hui un consortium soutenu par Intel, le Peer-to-Peer Working Group, qui propose de développer des technologies de base pour faciliter le développement des systèmes P2P. Ils proposent pour cela des solutions pour le NAT (translation d'adresses IP), pour les firewall etc. Il existe déjà une bibliothèque de sécurisation des communications fondées sur OpenSSL.

Un modèle de développement P2P en JAVA

Il existe un toolkit P2P développé en Java par la société Sun : JXTA[52]. Il propose de structurer un système autour de quatre éléments :

- Les « peer pipes », mécanismes de connexion d'un pair à un autre et de partage d'information sur le réseau, de manière distribuée
- Les « peer groups », qui permettent la création de groupe de façon dynamique et le regroupement logique et cohérent de contenu
- Le « peer monitoring », qui offre la possibilité de surveiller et de mesurer les interactions et de définir des politiques de contrôle entre pairs
- Des mécanismes de sécurités permettant de garantir la confidentialité, l'identité et l'accès contrôlé aux services.

L'une des questions fondamentales actuellement du point de vue scientifique est la résistance aux défaillances des systèmes. Historiquement le réseau Internet, dérivé de MilNet, proposait justement une architecture qui permettait de résister partiellement aux pannes, puisqu'en cas de défaillance d'une partie du réseau, le reste peut encore fonctionner. De son coté, le système P2P permet d'éliminer des serveurs informatiques coûteux et surchargés et

[52] Le site web de www.jxta.org permettra de se documenter davantage. A noter enfin que la société Activecluster annonce la sortie de son logiciel « Plateform » qui devrait aussi contribuer à renforcer l'intérêt des industriels envers le P2P.

donc minimise le risque de perdre toutes les données en cas de défaillance du serveur. Il permet aussi de mieux équilibrer la bande passante. Si le P2P veut connaître ses jours de gloire, il doit cependant être en mesure de résister aux défaillances et aux attaques des pirates. C'est la raison pour laquelle, certains scientifiques entreprennent de modéliser et de simuler les systèmes P2P afin de mieux comprendre les phénomènes inhérents à leur fonctionnement. Nous avons fait le choix du modèle P2P pour notre architecture d'agents distribuée en raison de sa légèreté et de la facilité de sa mise en œuvre.

VI.16. Mobilité et Persistance des Threads

VI.16.1. Mobilité

Un problème assez délicat en JAVA est la mobilité des processus et threads. Dans le cadre d'une application multi-agents, les agents mobiles doivent par définition migrer d'une plateforme à une autre. Une solution assez brutale consiste à tuer le processus, à copier les fichiers sur une autre plateforme et à relancer le processus sur cette nouvelle plateforme. Il faut néanmoins pouvoir sauvegarder « l'état mental » de l'agent, c'est-à-dire l'état de ses variables internes. JAVA propose des solutions pour des applications mobiles mais pas pour rendre les threads mobiles. Cela pose de nombreux problèmes lorsque des tâches étaient en cours d'exécution.

VI.16.2. Persistance

Un thread est persistant si son exécution peut survivre à une terminaison anormale d'une application. La persistance consiste à sauver l'état du thread sur le disque et à le restaurer ; cela est notamment utilisé dans le cadre du déboguage d'applications ou pour des programmes de tolérances de panne. Java ne fournit pas de solution pour la persistance. Certains middleware comme CORBA proposent cependant quelques notions de persistance.

VI.16.3. Une solution élégante

Pour apporter une réponse à ces deux problèmes sus-cités, une solution élégante a été mise au point à l'INRIA par Sara Bouchenak [Bouchenak, 2001] et [Bruneton, 2002] qui consiste en la re-ingénierie de quelques packages de la JVM JAVA concernant la gestion des processus. Ce toolkit qui est présenté comme une extension de JAVA, propose une solution pour la mobilité et pour la persistance des threads et est donc naturellement et particulièrement adaptée à notre projet PhoenixNet.

Avec ce système, il devient possible de transférer d'une machine à une autre les threads d'une application et reprendre l'exécution exactement là où on était dans la machine d'origine[53].

Ce procédé est néanmoins assez contraignant dans la mesure où il faut mettre à jour le développement de ce package à chaque nouvelle version de JAVA livrée par SUN, ce qui, d'après une discussion avec S.Bouchenak, n'est pas prévu. Un des objectifs du projet Phoenix étant la maintenabilité du code, nous ne pouvons donc pas nous orienter vers cette solution.

A noter qu'il existe une solution similaire en cours de développement par Mobile Agent Technologies et International Knowledge Systems (www.iks.com) dans le cadre de la

[53] Ce toolkit (java.lang.thread) est disponible sur le site du projet SARDES (System Architecture for Reflective Distributed EnvironmentS), digne successeur des projets SIRAC et GUIDE de l'INRIA Rhône-Alpes : http://sardes.inrialpes.fr/research/JavaThread/

plateforme AgentX qui se nomme RTM pour Realistic Thread Migration. Nous en parlons dans notre état de l'art sur les plateformes SMA.

VI.17. Les techniques d'apprentissage

Le modèle YAMAM propose deux types d'agents : l'agent réactif et l'agent cognitif. Ce dernier requiert des capacités d'apprentissages complexes. Nous allons décrire les mécanismes de décision que nous voulons implémenter dans un premier temps. Mais notre étude bibliographique devra permettre d'implémenter un processus de décision plus complexe dans le futur. Nous avons choisi de nous intéresser à quatre principaux types de raisonnements "intelligents" qui sont les raisonnements à base de règles, les réseaux Bayésiens [Jensen, 2001], les réseaux neuronaux et le Q-Learning. Ces « raisonnements » requièrent des systèmes d'apprentissages qui peuvent être supervisés, non supervisés ou par renforcement. Nous nous sommes intéressés aux systèmes de retro-propagation du gradient et aux cartes de Kohonen ainsi qu'aux arbres de décisions. On évoquera les systèmes de classification qui ajoutent l'apprentissage aux systèmes à base de règles en utilisant les algorithmes génétiques.

Un élément central des comportements intelligents est la faculté de s'adapter ou d'apprendre par l'expérience. Les agents ne peuvent connaître a priori toutes les situations qu'ils seront amenés à rencontrer. L'apprentissage devient alors un atout considérable pour ces agents. L'agent cognitif qui pourra apprendre aura donc un avantage par rapport à celui qui ne le peut pas. L'agent pourra par exemple apprendre à quel agent faire confiance ou non. Un agent qui apprend peut reconnaître des situations déjà rencontrées et améliorer ses performances en fonction de son expérience. Beaucoup de travaux de recherche ont porté sur des algorithmes d'apprentissage. Certains se sont appliqués à étendre le cadre d'application de ces algorithmes d'apprentissage par renforcement à des problèmes qui peuvent se modéliser sous la forme de problèmes de décision markoviens partiellement observés. Il est ainsi possible d'apprendre des comportements plus complexes, dans des situations plus plausibles où la perception n'est pas toujours suffisante pour choisir une action optimale [Dutech, 1999]. Il existe plusieurs formes d'apprentissage :

- Une forme d'apprentissage simple est l'apprentissage par cœur (rote learning). On fait apprendre des nouvelles situations à l'agent pendant un certain temps. Plus il y passe de temps, plus il se perfectionne.
- Une autre forme d'apprentissage est l'ajustement de paramètres (poids). Il faut connaître a priori quels facteurs sont importants pour les prises de décision. Afin que le système nous fournisse une sortie correcte, on ajuste les poids du réseau et on réitère l'opération tant que l'on n'est pas satisfait.
- L'induction est une forme d'apprentissage par l'exemple ou on essaie d'extraire les caractéristiques importantes d'un problème ce qui permet ainsi de généraliser de nouvelles situations ou des entrées. Les arbres de décision et les réseaux neuronaux utilisent tous les deux l'induction et peuvent être utilisés pour les problèmes de classification ou de régression.
- Un autre type encore est appelé le clustering ou l'abstraction de connaissance. C'est la faculté de détecter des formes communes et de généraliser des nouvelles situations. Chaque échantillon est affecté à un cluster avec d'autres exemples déterminés comme similaires. Cela est souvent utilisé pour le marketing pour classer des acheteurs selon leurs centres d'intérêts ou selon leurs « stratégies » d'achat ou selon d'autres critères qui n'apparaissent pas clairement en regard des données brutes.

Toutes ces techniques d'apprentissage sont utilisées dans les outils de Data Mining. Le Data Mining se définissant comme le processus d'extraction d'informations non évidentes et précieuses dans une grande collection de données [Bigus, 1996]. Le principal intérêt du Data Mining est de trouver des formes dont on ignorait l'existence, donc de découvrir de nouvelles informations ou de nouvelles connaissances. Ce principe peut être repris par un agent pour découvrir son environnement ou pour traiter les données qu'il reçoit des autres agents.

Il y a essentiellement trois techniques d'apprentissage : l'apprentissage supervisé, l'apprentissage non supervisé et l'apprentissage par renforcement. Certains chercheurs combinent ces différentes approches. L'apprentissage supervisé, encore appelé apprentissage par l'exemple, est la technique la plus connue. Le système fait une prédiction basée sur les entrées et si la sortie diffère ou la sortie désirée, le système s'adapte pour produire une sortie correcte. Ce processus est répété jusqu'à ce que le système soit capable de faire des classifications précises. Il y a deux algorithmes célèbres : la retro-propagation dans un réseau de neurones et les arbres de décision. L'apprentissage non supervisé est utilisé lorsque le système a besoin de reconnaître des similarités entre les entrées ou identifier certaines caractéristiques dans les données d'entrées. Une fois les données présentées en entrée, le système effectue un partitionnement des données en groupes. Le processus de segmentation continue jusqu'à ce que le système place les mêmes données dans les mêmes groupes sur les différents passes successives. L'algorithme de la carte de Kohonen est le plus connu. L'apprentissage par renforcement est un type d'apprentissage supervisé. Ce type d'apprentissage est plus long que l'apprentissage supervisé et est moins efficace. Il est utilisé lorsque le système n'a pas l'information sur la sortie désirée. Dans certain cas, ce type d'apprentissage est le plus réaliste. Nous présenterons en annexe une formalisation des réseaux de neurones et quelques exemples de code montrant notre implémentation de ces systèmes d'apprentissage.

VI.17.1. Raisonnement à base de règles (rule-based reasoning)

Les systèmes experts à base de règles ont connu un succès indéniable dans de nombreux domaines à partir des années 1960. Un tel système inclut un moteur d'inférence, une base de faits ainsi qu'une base de règles, dans laquelle la connaissance a été introduite par avance par des spécialistes du domaine. Il fonctionne en enchaînant les règles de la base de règles en fonction d'une requête soumise initialement. Certains de ces systèmes experts à base de règles disposent de modules additionnels permettant de fournir des explications au résultat obtenu ou d'acquérir la connaissance dans la base de règles. Le succès de cette approche est dû à la séparation entre la base de règles et le moteur d'inférence, augmentant ainsi la flexibilité de manière significative. Cette approche présente toutefois plusieurs inconvénients. Cette information, lorsqu'elle est connue par des spécialistes du domaine, est encore difficile à obtenir et à formaliser. Du point de vue des performances, la base de règles est susceptible de devoir être modifiée de manière très importante à chaque changement des entités gérées et le système de raisonnement lui-même devient de plus en plus lent au fur et à mesure de l'extension de la base de règles.

VI.17.2. Réseaux Bayésiens

Un réseau bayésien est constitué de nœuds représentants des assertions incertaines et d'arcs orientés représentants les causes et les effets entre les nœuds. Les arcs supportent une distribution de probabilité pour chaque nœud effet, basée sur la probabilité de ces nœuds cause. Grâce à leur capacité à traiter les incertitudes et leur représentation spécifique des connaissances, les réseaux bayésiens sont particulièrement intéressants dans les domaines où

les connaissances sont incomplètes. Cependant, les faiblesses de cette approche sont proches de celles énoncées pour le raisonnement à base de règles.

VI.17.3. Réseaux de neurones

Les réseaux neuronaux ont déjà trouvé de très nombreuses applications dans le domaine du diagnostic, du traitement d'image, de la classification ou de la reconnaissance de formes. Ils peuvent être très utiles grâce à leur capacité à reconnaître efficacement des modèles donnés, à faire une approximation d'une fonction quelconque à partir d'un nombre suffisant de neurones et d'apprendre à l'aide de paires entrées /sorties en ajustant automatiquement les poids sans réellement comprendre de manière approfondie le domaine. Plus important, les réseaux neuronaux, tout comme les réseaux bayésiens, peuvent fonctionner avec des données incomplètes, ambiguës ou imparfaites. L'un des problèmes de cette approche est que le réseau doit être entraîné au départ avec un grand nombre d'exemples de données consistant en des paires entrées /sorties, ces données pouvant être indisponibles dans certains domaines. Finalement, l'impossibilité, une fois une solution fournie par le réseau neuronal, d'avoir l'explication de ce résultat, présente aussi un autre défaut. On restreindra donc leur utilisation à des domaines très particuliers. Le fonctionnement des réseaux de neurones est décrit en annexe.

VI.17.4. Q-Learning

Le Q-learning est un modèle d'apprentissage par renforcement. Il permet aux agents avec une capacité d'apprentissage d'agir de façon optimale dans leur environnement. Le *Q-learning* est une forme primitive d'apprentissage où l'agent teste une action à un état particulier et évalue ses conséquences en termes de récompenses immédiates ou de pénalités qu'il reçoit de l'environnement. En d'autre terme la Q-valeur est la récompense escomptée pour l'action "a" à l'état "x" et en suivant une stratégie p. L'objectif du *Q-learning* est d'estimer les *Q-valeurs* pour une stratégie optimale :

$$Q(x(t), a(t)) = \sum_{j=0}^{\infty} \gamma^j r(t+j) \qquad (1)$$

où t représente le temps à l'instant t, $0 < \gamma < 1$ est le facteur d'ajustement de l'importance des actions dans le temps et r(t) le renforcement reçu au temps t. Nous pouvons également estimer un état, comme la meilleure valeur de l'action dans cet état (qui correspond à l'utilité) :

$$U(x) = \max_x Q(x, a) \qquad (2)$$

L'algorithme d'apprentissage, s'exécute en créant un tableau bi-dimensionnel des Q-valeurs indexées par les actions et les inputs et en ajustant dans le tableau les valeurs basées sur les actions prises et les renforcements reçus.

VI.18. Conclusion

La conception de notre plate-forme nous a conduit à étudier différentes technologies et à faire des choix d'architecture. Nous avons fait le choix assez radical de ne pas utiliser de serveur central. Au contraire, une architecture Point à Point a été retenue.

Nous avons orienté notre choix sur le langage JAVA pour son ouverture, sa fiabilité, sa sécurité et surtout sa portabilité. Ce langage nous permettait de travailler sur plusieurs systèmes d'objets distribués et nous avons opté pour CORBA qui offre de nombreux services

additionnels et une possibilité d'interfaçage avec d'autres langages comme le C++. Nous souhaitions pouvoir communiquer d'agent à agent avec un protocole sécurisé et avons choisi SSL, standard du marché, qui offre un cryptage de l'information grâce à différents algorithmes. Toujours pour pérenniser notre développement, nous avons retenu les web-services, qui sont de plus en plus utilisés dans l'industrie, pour communiquer avec des systèmes distants hétérogènes. Ces Services Web sont aussi utilisés pour la mobilité des agents, notamment le transfert des compétences. La persistance des données reposera sur XML.

Nous n'avons pas retenu de solutions qui ne garantissaient pas de pérennité. Notre préoccupation est de concevoir un logiciel que l'on puisse maintenir dans le temps et dont on puisse étendre à loisir les fonctionnalités.

Chapitre VII. Conception du middleware Phoenix

« Il semble que la perfection soit atteinte, non quand il n'y a plus rien à ajouter, mais quand il n'y a plus rien à retrancher » - Antoine de Saint Exupéry.

Résumé : Ce chapitre concerne la phase de conception de notre plate-forme SMA, présentée sous forme de diagrammes UML. Chaque sous-module est explicité en détail : les agents, les messages, le market, la migration, la persistance, l'accès distant aux compétences, la répartition des rôles et des tâches, le système décisionnel et la sécurité des communications.

VII.1. Introduction

Phœnix est le nom de code donné à notre plate-forme multi-agents conçue sur le modèle YAMAM. Le choix du nom, qui fait référence à l'oiseau mythique[54], s'explique en partie par les propriétés de « résurrection » des agents. Un agent temporairement inapte ou sous-exploité est voué à mourir pour ne pas surcharger la plate-forme. Il peut cependant reprendre vie lorsque une nouvelle mission justifie sa présence.

Nous détaillons dans cette section les principales fonctionnalités de cette plate-forme et présentons les concepts les plus intéressants de la conception. Phoenix est donc l'implémentation JAVA du modèle YAMAM. Nous avons organisé en packages les quelques 200 classes composant Phœnix. Chacun de ces package sera présenté en détail dans les prochains paragraphes.

VII.2. Fonctionnalités globales du Middleware

De nature très distribuée, la plate-forme construite sur le modèle FIPA, présente des caractéristiques d'allocation dynamique de processus et de délégation de rôle et de tâches. Ces caractéristiques permettent indirectement de bénéficier d'un équilibrage de charge. Un langage de script, lu par un parser, permet d'ordonnancer les tâches séquentiellement ou parallèlement.

Un agent peut être de nature réactive ou cognitive. Dans ce dernier cas, il dispose d'un système décisionnel qui lui permet de sélectionner une action lorsqu'une alternative se présente. Ce système peut être couplé à une mémoire et à des algorithmes d'apprentissage.

Une plate-forme peut aussi adopter un comportement proactif, ce qui permet par exemple de découvrir automatiquement de nouvelles plates-formes et d'intégrer de nouveaux

[54] Le Phénix est un oiseau imaginaire qu'on retrouve dans plusieurs civilisations : grecque (Phoenix), égyptienne (Benu), indienne (Garuda), chinoise (Feng), juive (Milcham) ou japonaise (Karura). Il est la personnification de la création et de la force vitale. C'est dans la civilisation égyptienne qu'il est fait mention pour la première fois de cet oiseau, le Benu, ressemblant à un héron rouge/doré. Il serait un oiseau sacré de la cité du Soleil Héliopolis et lié aux dieux Ra (soleil) et Osiris (dieu des enfers). C'est ce dernier qui lui aurait conféré l'immortalité. Sa principale caractéristique est de renaître de ses cendres.

agents. Les agents migrent physiquement d'un hôte à un autre en emportant éventuellement leurs compétences. Les compétences des agents, constituant leur savoir et leur savoir-faire, sont présentées sous forme de composants métiers. Elles sont accessibles par d'autres agents et d'autres programmes via les WebServices et sont référencées dans un systèmes de « pages jaunes » pour faciliter leur identification et leur localisation.

L'architecture globale repose sur un système point à point. Les communications peuvent être sécurisées, ce qui permet d'utiliser la plate-forme dans des domaines critiques comme l'e-business. Les agents communiquent entre eux de façon synchrone ou asynchrone via des protocoles standards comme XML, ACL ou SL0.

Enfin, la mise en œuvre globale d'un système de sous-traitance de tâches et de localisation de compétences est proche du concept de Grid Computing. Une passerelle de compatibilité avec un système Grid plus standard comme Globus est proposée.

Le fonctionnement auto-adaptatif, basé sur le modèle YAMAM, propose une dynamique basée sur un échange d'énergie entre les agents et les plates-formes. Ce mode de fonctionnement permet de classer la plate-forme parmi les premiers systèmes logiciels d'Autonomic Computing[55] [ACIBM, 2003].

VII.3. Positionnement de la plate-forme

Le groupe ASA [ASA, 2002] a défini un certain nombre de catégories permettant de classifier les plates-formes SMA. Phoenix se positionne comme suit :

VII.3.1. Caractéristiques générales :

Nom de la plate-forme : Phœnix
Auteurs : Marc Savall (PSI/Araknyde) et l'équipe d'Araknyde
Nature de la plate-forme : simulation et internet/intranet - SMA générique -- Environnement de développement de SMA
Etat de la plate-forme : prototype opérationnel
Logiciel libre/commercial : commercial
Applications développées : petits exemples – application de simulation
Type d'application privilégié : Simulation - Résolution de problème – Gestion de réseau – Autonomic Computing
Nombre d'utilisateurs : ?
Références bibliographiques sur la plate-forme et les modèles qu'elle utilise : 7 articles de conférence et 1 revue.

VII.3.2. Configuration logicielle et matérielle :

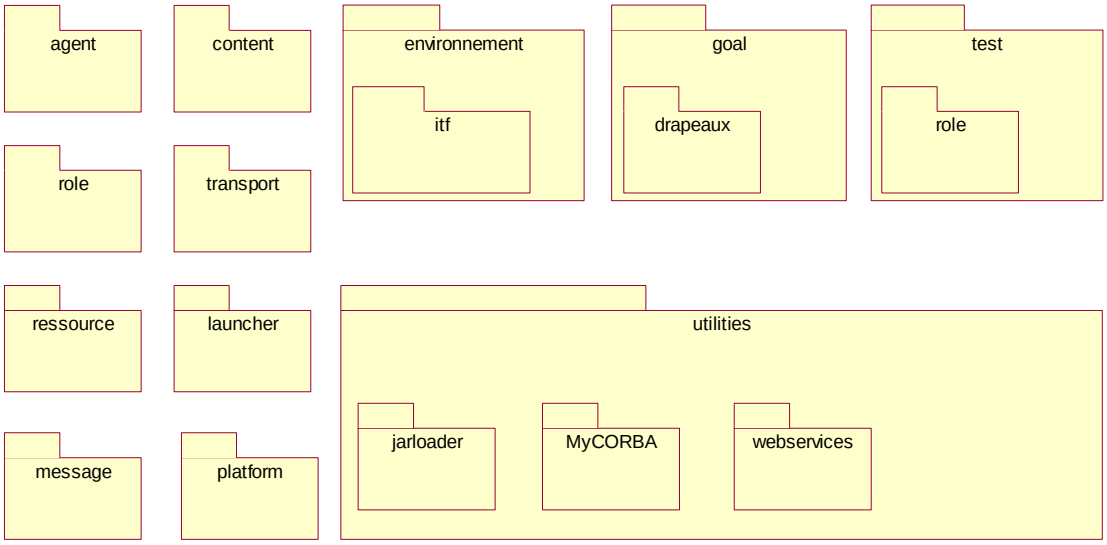
Configuration logicielle nécessaire : JDK 1.4 / CORBA 2.3
Langage utilisé : JAVA2
Machines cibles : toute machine supportant Java
Systèmes d'exploitation : tout OS supportant Java

[55] Les caractéristiques d'un système d'Autonomic Computing sont présentées dans le chapitre dédié aux Choix Technologiques de notre plate-forme.

VII.4. Conception de la plateforme

VII.4.1. Vue d'ensemble des packages

Pour faciliter la réutilisation du code, Java permet de regrouper plusieurs définitions de classes dans un groupe logique appelé paquet (ou package). Le langage Java est fourni avec de nombreux paquets prédéfinis. Par exemple, le paquet *java.applet* contient des classes permettant de travailler avec les applets Java. De même, nous avons regroupé les classes du middleware Phoenix dans des packages.



VII.4.2. Modélisation de la plateforme

VII.4.2.1. Respect du standard FIPA

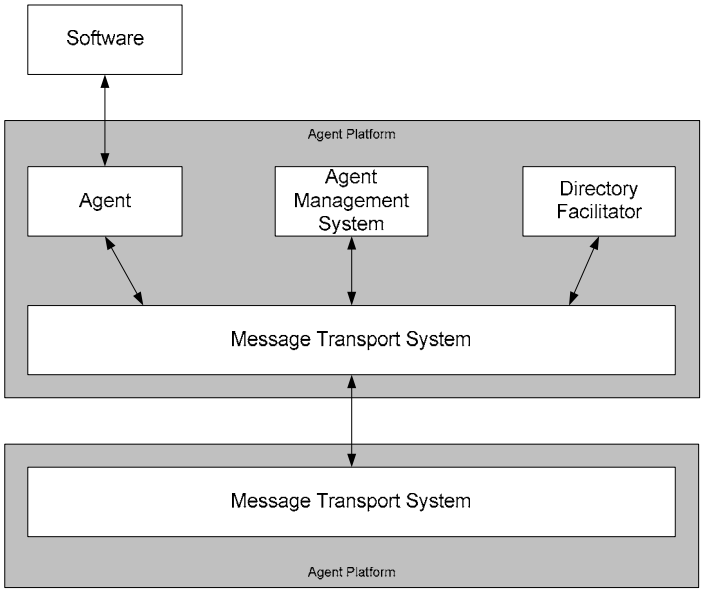
Tout comme April Agent Platform de Fujitsu Labs, Grasshopper de IKV++, JADE [Bellifemine & al., 2001] ou Zeus de British Telecom [Nader & Thompson, 2000], Phoenix est conçu sur les spécifications de FIPA [FIPA, 2000]. Il reprend donc l'architecture de l'Agent Management Reference Model proposé par FIPA. Un agent particulier L'AgentPlatform (AP) fonctionnant comme un noyau, prend en charge plusieurs rôles assimilés à des services. Les services de bases proposés par le noyau sont le Directory Facilitator (DF), l'Agent Management System (AMS) et le Message Transport Service (MTS).

Le DF est un composant qui fait office d'annuaire. C'est un service de "pages jaunes" qui permet de mettre en relation les agents avec leurs compétences. Ainsi, un agent peut enregistrer ses compétences dans le DF ou interroger celui-ci pour connaître les agents possédant une compétence précise. Selon les normes FIPA, il est possible d'avoir plusieurs DF sur une même plate forme mais dans notre implémentation, chaque plate-forme Phoenix dispose d'un unique DF. Le DF est un des éléments centraux lors de la délégation de tâches : il permet aux agents de savoir qui est capable de réaliser la tâche qu'ils désirent déléguer : le DF rassemble dans une table associative (java.util.hashtable) les agents par compétences possédées. Cette table est remise à jour par envoi de message des agents lorsque ceux-ci apprennent de nouvelles compétences (ou à leur création bien entendu). Cependant les informations contenues dans le DF sont non fiables, c'est-à-dire que des agents décédés peuvent être encore référencés dans le DF (ou encore des agents ayant migré sur une autre

plate-forme sans en avoir informé le DF). Les agents ont la possibilité de contacter des DF distants afin de se renseigner sur les agents de cette plate-forme (les DF ne se contactent pas entre eux). Afin de gérer les requêtes au DF, on utilise le mécanisme de messages standards.

L'AMS est un autre composant important car il assure la cohérence des entités de la plate-forme et maintient un répertoire contenant les adresses de transport des agents de la plate forme. Ce service est de type "page blanche" car il fait la correspondance entre un agent et son identificateur unique, appelé AID, que nous étudierons ultérieurement. Dans le modèle théorique YAMAM, nous souhaitions tout « agentifier » de la même manière. Le noyau de la plate-forme était en fait un agent disposant de compétences particulières lui permettant de tenir ce rôle particulier. Au vu des difficultés imposées par ce modèle, on a fait le choix d'écrire l'AMS sans le faire dériver d'*AbstractAgent*, ce qui n'a que peu d'importance finalement, car grâce au système de communication par messages, l'AMS peut facilement se faire passer pour un agent offrant les services de gestion de la plate-forme aux yeux des autres agents. D'autre part, l'utilisation de messages introduit de l'asynchronisme, ce qui évite l'utilisation trop massive du mot clef *synchronized* : l'utilisation de blocs synchronisés provoque un effondrement des performances lorsque les agents sont nombreux à attendre après une même méthode (ce qui est très fréquent pour les méthodes offertes par l'AMS, le DF, l'environnement ou le Market). La classe AMS rassemble donc les méthodes utilitaires permettant de gérer la création de ressources, la création d'agents, la destruction de ressources ou d'agents (avec remise à jour du registre RMI), la recherche d'un ACC pour pouvoir envoyer des messages, et la coopération avec d'autres plates-formes Phoenix. L'AMS décide aussi des coûts d'imposition infligés aux agents. Vu le grand nombre de tâches qu'a à effectuer l'AMS en simultané, nous avons eu à utiliser plusieurs threads, afin de gérer la lecture de la boite aux lettres, ou encore l'imposition des agents. Afin de garantir une qualité de service constante (en terme de temps de réponse particulièrement), l'AMS peut créer plusieurs threads supplémentaires pour servir les agents (un peu comme le serveur HTTP apache crée plusieurs copies de lui même afin de pouvoir servir plusieurs clients simultanément, en gardant toujours quelques serveurs d'avance afin de pouvoir répondre à un pic de requêtes). Cependant afin de limiter la prolifération de threads, un nombre maximal de threads pour l'AMS peut être spécifié dans le fichier de configuration de la plate-forme.

Le MTS fournit un mécanisme pour le transfert de messages entre agents. Les agents concernés peuvent être locaux sur un unique AP ou sur différents AP. Sur un unique AP, le MTS est fourni par un Agent Communication Channel (ACC). L'Agent Communication Channel (ACC) propose un service directement aux agents de l'AP. L'ACC peut accéder aux informations des autres services de l'AP (comme l'AMS ou le DF) pour permettre d'effectuer ses tâches de transport de messages. L'ACC opère l'envoi physique de messages grâce à 2 Message Transfert Protocol (MTP). L'Internal MTP est un protocole de transport de messages permettant à toute entité de la plate forme de contacter l'ACC locale. L'External MTP est un protocole de transport de messages permettant à l'ACC de la plate forme de contacter l'ACC d'une plate forme distante.



Un agent possède un identifieur unique sur la plate forme appelé AgentIDentifier (AID). Il comprend différentes parties :

- Un nom de type name@HAP où HAP (Home Agent Platform) correspond au nom de la plate-forme suivi de l'adresse IP de la machine hôte (ex: AgentA@PlatformA.194.254.17.91). Pour passer des routeurs avec translation d'adresse (NAT), nous proposons d'ajouter à cette clé composée, un numéro de port IP permettant de faire de la redirection de port pour utiliser plusieurs plateformes phoenix avec une seule IP publique.
- Une adresse de transport facultative qui correspond à l'adresse où l'agent peut être contacté. Si l'agent peut recevoir lui-même des messages, il s'agit de l'adresse de l'agent, sinon, il s'agit de l'adresse de l'ACC dont dépend l'agent.
- Une adresse de service de résolution de nom qui correspond à l'adresse où l'on peut faire correspondre une adresse de transport au nom d'un agent. Seul l'AMS possède un service de résolution de nom dont l'adresse correspond à l'adresse de transport de l'AMS.

VII.4.2.2. Liste des packages les plus significatifs

Les classes ont été organisées suivant divers packages, ce découpage étant motivé par des considérations de maintenabilité du code. Les divers packages sont issus des groupements mis en relief lors des spécifications. Voici la liste des package et les types de classes qu'ils rassemblent :

- Agent : toutes les classes concernant l'agent, la délégation, l'accointance, les identificateurs d'agents, les mémoires, les variables de rôles, etc ...
- Content : toutes les classes concernant la création de messages aux formats ACL et SL0, ainsi que les parsers permettant de les décoder.
- Environnement : sont rassemblées ici les classes permettant la gestion des divers environnements.
- Goal : il ne s'agit pas d'un package à proprement parler, mais afin de pouvoir séparer les fichiers spécifiques à chaque problème (comme les implémentations de rôles, ou les environnements spécifiques), toutes les classes concrètes ont été rassemblées dans une sous-arborescence identique à celle-ci, mais placée dans un package qui porte le nom du programme (par exemple l'environnement du toy-problem de recherche des

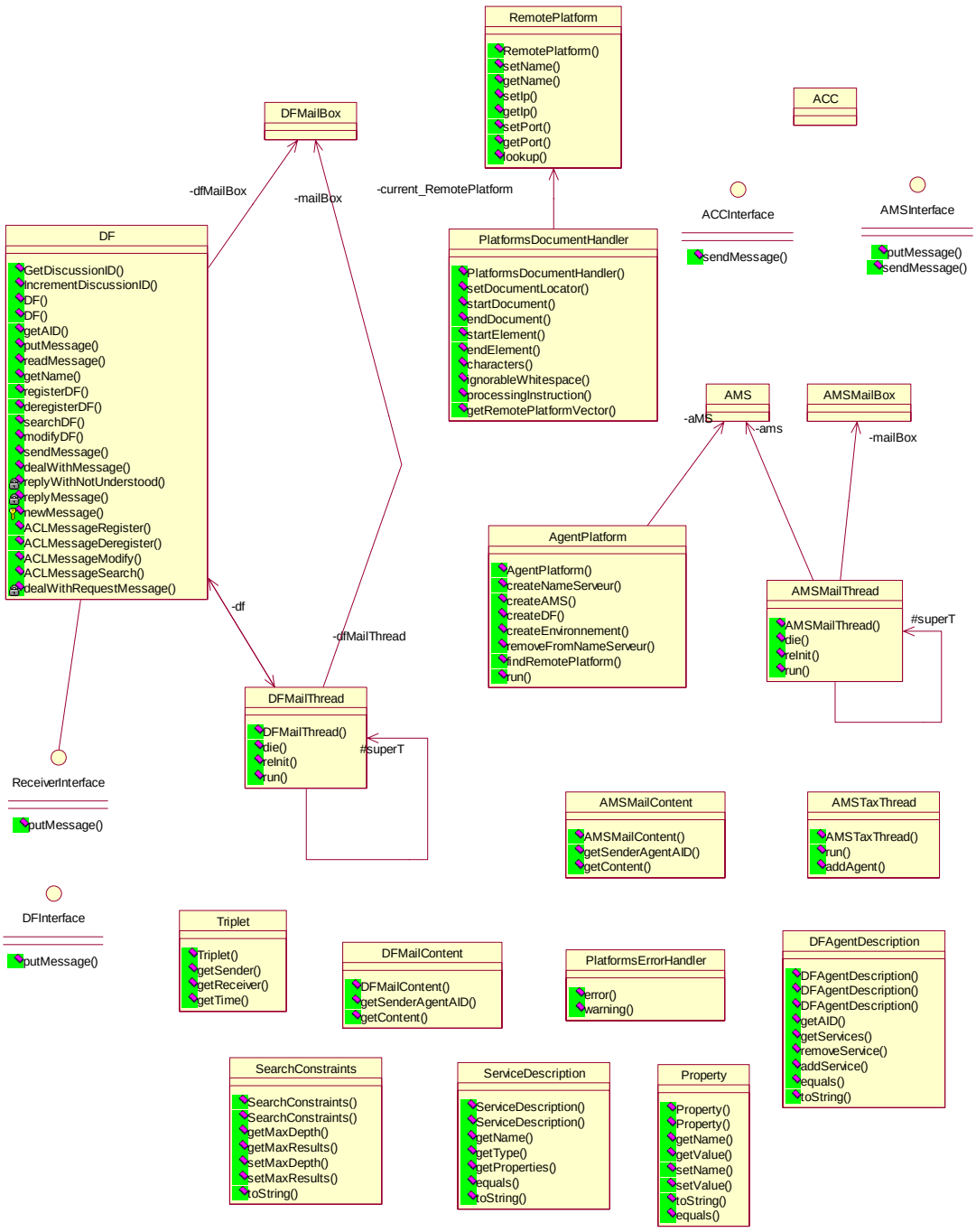
drapeaux est implémenté en dérivant la classe nommée *phoenix.goal.drapeaux.environnement.environnement2*d qui hérite de la classe abstraite *phoenix.environnement.AbstractEnvironnement*).

- Launcher : toutes les classes permettant l'initialisation de la plateforme et le lancement des programmes contenus dans des fichiers de type Jar (en fait ces fichiers Jar contiennent chacun un des programmes définis dans un sous package de goal).
- Message : toutes les classes permettant le transport de messages, l'interprétation ou la création de messages n'est pas gérée ici. Pour la plupart, ces classes sont utilisées par la classe *phoenix.platform.ACC*.
- Platform : les classes concernant la plate-forme elle même, on trouve ici en particulier les implémentations des agents spéciaux comme l'AMS (l'agent noyau), mais aussi les diverses fonctionnalités offertes par une plate-forme FIPA : le DF, l'ACC et le NameServer, et enfin les services utilitaires supplémentaires comme le market ... D'autres classes ont été ajoutées ici : comme les classes permettant la gestion de l'énergie ou d'autres classes utilitaires non reliées à un autre package.
- Ressource : toutes les classes concernant la gestion des ressources, en particulier le système d'identification des ressources, ou la gestion des lots de ressources.
- Role : ce package rassemble tout à la fois la gestion des rôles, tâches et compétences, en particulier les divers interpréteurs, la gestion du parallélisme et des accès concurrents aux variables, les mécanismes d'identification des rôles, tâches ou compétence, et le système de sauvegarde de l'état d'un rôle pour une reprise ultérieure.

VII.4.2.3. Configuration de la plate-forme

Afin de pouvoir passer des options de configuration à la plate-forme Phoenix, nous avons inclut un système de fichier de configuration. Ce fichier est écrit au format XML, et est lu grâce au parser fourni par la librairie *xml-tr2*. Ce fichier permet de changer la valeur des variables statiques servant à contrôler le comportement de la plate-forme (comme le nombre maximal de threads pour l'AMS ou l'environnement, ou encore la ou les plage(s) de plates-formes distantes avec qui coopérer...

VII.4.2.4. Diagramme d'héritage de la plate-forme



VII.4.3. Modélisation des agents

Un agent est une unité autonome et communicante. Son autonomie nécessite donc l'utilisation d'un *thread* au moins. La contrainte majeure est posée par la possibilité pour un agent de recevoir et de traiter des messages asynchrones. Deux directions s'en suivent :

- la représentation d'un agent par deux threads, l'un gérant l'exécution du rôle, et l'autre traitant ses messages asynchrones.

- la représentation d'un agent par un thread uniquement, devant régulièrement vérifier qu'aucun message n'a été reçu.

La seconde méthode, bien que plus coûteuse que la première a priori (il s'agit d'une attente semi-active) est en fait largement préférable : vu le grand nombre d'agents sur la plate-forme, le surcoût en mémoire engendré par la multiplication par deux du nombre de *threads* grèverait par trop les performances de Phoenix. A noter que nous n'avons pas implémenté un système de Sonde comme dans MadKit [Gutknecht & Ferber, 2000]. Cette solution est pourtant alléchante pour minimiser le nombre de threads lors de la simulation d'un système biologique par exemple.

VII.4.3.1. Les agents réactifs

L'agent réactif sera représenté par la classe *ReactiveAgent* dont les principales fonctionnalités sont :

- Lors de sa création, l'agent noyau lui assignera un rôle qu'il exécutera en boucle.
- Seul l'agent noyau pourra décider de changer le rôle d'un agent.
- L'agent réactif ne pourra déléguer une tâche que s'il y est autorisé.
- Un agent réactif ne peut décider de déléguer une tâche que s'il n'a pas la compétence ou s'il n'a pas les ressources nécessaires.
- De la même façon, si un agent réactif est autorisé à exécuter des tâches déléguées, il les fera sans se poser de questions.

L'agent réactif ne pourra déléguer une tâche que s'il y est autorisé. Une variable sera initialisée pour le dire. Un agent réactif ne délègue une tache que s'il n'a pas la compétence ou s'il n'a pas les ressources nécessaires à la réalisation de celle-ci. A noter que s'il n'y a aucun agent à qui déléguer une tâche sur la plate-forme locale alors le *Directory Facilitator* en recherchera un parmi les agents des plates-formes distantes. Si aucun agent du SMA ne possède la compétence recherchée alors la plate-forme locale va décider de créer un agent cognitif qui pourra former l'agent demandeur (s'il a la compétence d'apprendre une compétence). Lorsqu'une tâche est déléguée à un agent réactif autorisé à recevoir des tâches et possédant les compétences nécessaires, il ne peut la refuser et l'exécute automatiquement.

VII.4.3.2. Les agents cognitifs

L'agent cognitif sera représenté par la classe *CognitiveAgent* qui hérite indirectement de la classe *AbstractAgent*. Cette classe peut être liée à la classe *CognitiveSkill* pour adjoindre des capacités d'apprentissage à l'agent cognitif. Ces dernières implémentent les algorithmes d'apprentissage décrits dans le chapitre des Choix Technologiques. La plupart de ces fonctions furent récupérées en freeware ou sur des dites dédiés à la programmation OpenSource. Nous avons implémenté un réseau de neurones en rétro-propagation. Nous avons aussi proposé dans le modèle, un système d'état mental qui peut être attaché à l'agent cognitif. Cet « état mental » est découplé volontairement de l'agent cognitif et peut aussi utiliser ces compétences d'apprentissage. Nous n'imposons aucune technologie en particulier pour l'apprentissage. Le développeur utilisera à sa guise l'une ou l'autre des classes que nous proposons en fonction du contexte et de ses besoins spécifiques. Ces choix sont à réaliser lors de l'attachement des compétences (skills). En revanche nous avons imposé à l'agent cognitif d'utiliser le système d'échange d'énergie, car cela est lié au système décisionnel.

Lorsqu'un agent a le choix entre plusieurs alternatives, le système décisionnel intervient. Il peut être programmé sous forme de règles stipulant le meilleurs choix à faire

dans une situation particulière. Dans tous les cas, l'agent doit chercher à maximiser son niveau d'énergie.

VII.4.3.3. L'agent noyau

L'agent noyau, aussi appelée AMS, est un agent très particulier. Il implémente des fonctionnalités que les autres agents ne possèdent pas :

- Un système de jetons a été implémenté : pour chaque rôle sur une plate-forme un nombre de jetons est associé qui représentera le nombre d'agent pouvant jouer un rôle. Ce système permettra de simuler l'offre et la demande au niveau des rôles.
- L'AMS peut renvoyer la liste des rôles avec le nombre de jetons restants.
- Un agent cognitif peut demander à l'AMS de jouer un rôle.
- L'agent rend le jeton d'un rôle quand il l'a achevé.

Son principe de fonctionnement est aussi différent des autres agents. Lors de la création d'un rôle sur la plate-forme il est possible d'indiquer le nombre de jetons disponibles simultanément pour ce rôle. Ce nombre indiquera combien d'agents peuvent exécuter ce rôle au même moment. Par exemple, admettons qu'un rôle consiste à scruter l'arrivée de mail sur une plate-forme au sein d'un SMA consistant à optimiser les performances d'un réseau. Alors ce rôle n'aura peut être pas grand intérêt à être effectué simultanément par deux agents différents. On peut donc instaurer un unique jeton pour ce rôle.

La possibilité est aussi offerte d'instaurer un nombre infini de jetons ce qui permet à un rôle d'être exécuté sans limite autant de fois que souhaité.

De plus, un agent peut connaître le nombre de rôles théoriquement disponibles simultanément sur la plate-forme et le nombre de rôles disponibles actuellement. Il lui suffit d'interroger l'AMS par l'intermédiaire d'un message.

Ces fonctionnalités permettent de « contrôler » le bon fonctionnement d'une plate-forme et ainsi de veiller à ce qu'elle effectue les fonctionnalités souhaitées.

VII.4.4. Les messages

Un système multi-agents nécessite une communication entre agents. Grâce à elle, émergent entre les agents des comportements d'interaction : collaboration, coordination, compétition. Les messages représentent cette interaction entre l'agent et ce qui l'entoure (son environnement, les autres agents, ...). Il existe plusieurs grandes classes de messages, regroupant tous les messages ayant la même composante illocutoire (la nature du message) :

- messages affirmatifs
- requêtes
- demandes de faire quelque chose
- ...

On utilise un verbe comme mot-clé pour indiquer la classe d'un message donné. Ce verbe est appelé performative. La sémantique d'un message est donc très liée à la performative. De plus, certains messages nécessitent une réponse et d'autres non. Pour caractériser pleinement la communication dans un SMA, il faut :

- définir l'ensemble des performatives
- pour chaque performative, définir la syntaxe et la sémantique des différents messages de cette performative

- préciser pour chaque message s'il y a une réponse, et l'ensemble des types possibles qu'elle peut prendre.

Nous avons choisi d'implémenter uniquement ACL, mais un développeur peut sans aucun problème programmer une nouvelle compétence de façon à ce que les agents utilisent MASIF ou KQML. Phoenix doit être vu comme un environnement de développement fourni avec quelques fonctionnalités de base mais il est possible et souhaitable d'en étendre les fonctionnalités. Si l'on utilise l'exemple des messages ACL, on trouve les performatives :

- inform
- query-ref
- inform-ref
- agree
- refuse
- propose
- not-understood

Ce qui permet de représenter tous les messages dont a besoin Phoenix. Il est cependant possible de spécialiser la classe ACL pour étendre ces performatives ou surcharger les fonctions d'interprétation.

VII.4.4.1. Structure des messages

FIPA utilise le langage de communication FIPA-ACL [FIPA, 2000]. Celui-ci comprend un ensemble d'éléments dont la *performative*, le *sender*, le *receiver* et le *content* sont obligatoires. Si un agent ne comprend pas ou n'est pas capable de traiter un message il peut répondre avec un message approprié de type *not-understood*.

Elements	Description
performative	Décrit le type de communication
sender	Définit l'identité de l'expéditeur du message
receiver	Définit l'identité du destinataire du message
reply-to	Indique que les messages découlant de la conversation entamée devront être envoyés à l'agent défini par l'élément *reply-to* et non pas par l'agent défini par l'élément *sender*.
content	Définit le contenu du message c'est à dire l'objet de l'action à effectuer.
language	Définit le langage dans lequel content est défini.
encoding	Définit l'encodage utilisé pour le *content.*
ontology	Définit la (les) ontologie(s) utilisée(s) pour donner un sens au *content*.
protocol	Protocole
conversation-id	Définit une expression qui identifie les séquences de communication formant une conversation.
reply-with	Définit une expression utilisée par l'agent receveur pour identifier le message.
in-reply-to	Définit une expression qui référence une action préalable dont le message est une réponse. exemple : Si l'agent i envoie à l'agent j un message contenant : *reply-with <expr>* L'agent j répondra un message contenant : *in-reply-to <expr>*
reply-by	Définit une heure et (ou) une date indiquant la date avant laquelle l'agent expéditeur aimerait recevoir une réponse.

VII.4.4.2. Identification de l'expéditeur et du destinataire

On est renseigné sur l'expéditeur et le destinataire par les éléments *sender* et *receiver*. Ceux-ci contiennent un élément *agent-identifier* identifiant l'agent. Ainsi, on a dans le message :

(inform
: receiver (set (agent-identifier

```
: name agentName@platformName.hostName
: addresses (sequence
        (//hostName/platformName/ACCName) )
: resolvers (sequence
        (agent-identifier
        : name AMSName@@platformName.hostName
        : addresses (sequence
                (//hostName/platformName/AMSName) )
))
//suite du message
```

Agent-identifier contient le nom de l'agent, une ou plusieurs adresses de transport et un ou plusieurs agents permettant la résolution de noms. L'élément *name* est obligatoire. L'élément resolvers est obligatoire si l'élément addresses n'est pas renseigné et celui-ci est donc facultatif. Les éléments addresses et resolvers peuvent avoir respectivement plusieurs adresses de transport et plusieurs agents resolveurs.

VII.4.4.3. Champ lexical et grammaire

Pour se conformer à la spécification de FIPA sur le format ACL, nous avons respecté la grammaire exacte spécifiée par la spécification n° XC00070F. Le champ lexical est celui-ci :

Word	= [~ "\0x00" – "\0x20", "(", ")", "#", "0" – "9", "-", "@"] [~ "\0x00" – "\0x20", "(", ")"]*.
String	= StringLiteral | ByteLengthEncodedString.
StringLiteral	= "\"" ([~ "\""] | "\\\"")* "\"".
ByteLengthEncodedString	= "#" Digit+ "\"" <byte sequence>.
Number	= Integer | Float.
URL	= See [RFC2396]
DateTimeToken	= "+" ? Year Month Day "T" Hour Minute Second MilliSecond TypeDesignator ?).
Year	= Digit Digit Digit Digit.
Month	= Digit Digit.
Day	= Digit Digit.
Hour	= Digit Digit.
Minute	= Digit Digit.
Second	= Digit Digit.
MilliSecond	= Digit Digit Digit.
TypeDesignator	= AlphaCharacter.
AlphaCharacter	= ["a" – "z"] | ["A" – "Z"].
Digit	= ["0" – "9"].
Sign	= ["+" , "-"].
Integer	= Sign? Digit+.
Dot	= ["."].
Float	= Sign? FloatMantissa FloatExponent? | Sign? Digit+FloatExponent
FloatMantissa	= Digit+ Dot Digit* | Digit* Dot Digit+

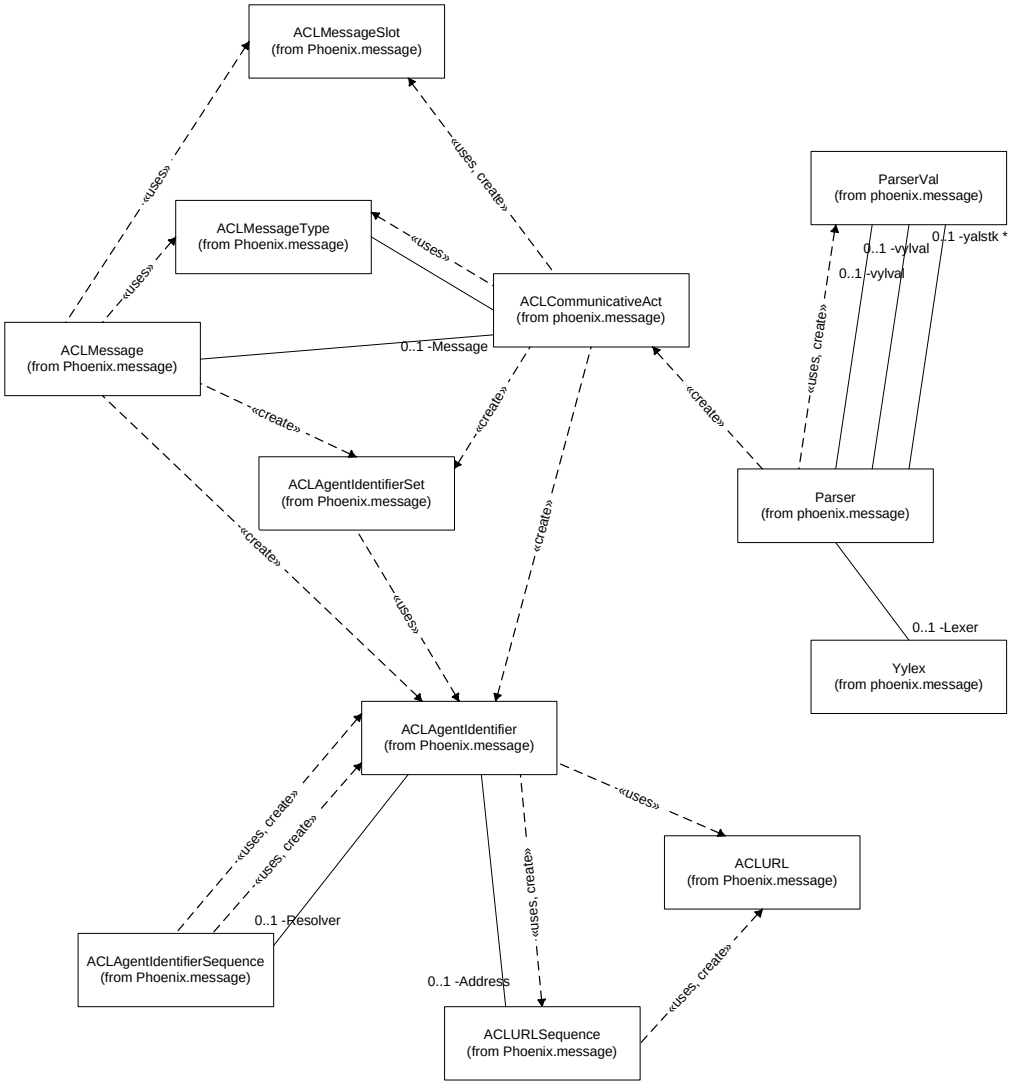
La grammaire est la suivante :

ACLCommunicativeAct	= Message.
Message	= "(" MessageType MessageSlot* ")".
MessageType	= See [FIPA00037]
MessageSlot	= ":sender" AgentIdentifier | ":receiver" AgentIdentifierSet | ":content" String

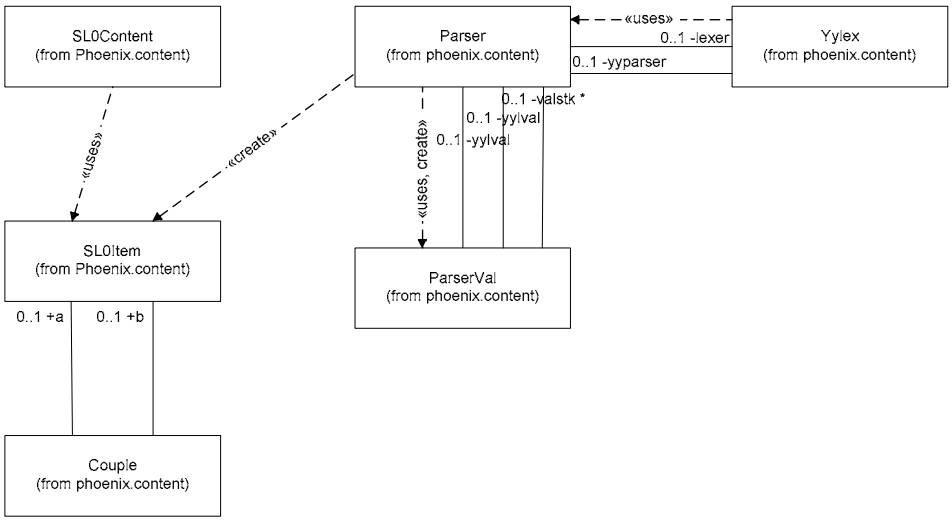
	| ":reply-with" Expression | ":reply-by" DateTime | ":in-reply-to" Expression | ":reply-to" AgentIdentifierSet | ":language" Expression | ":encoding" Expression | ":ontology" Expression | ":protocol" Word | ":conversation-id" Expression | UserDefinedSlot Expression.
UserDefinedSlot	= Word
Expression	= Word | String | Number | DateTime | "(" Expression* ")".
AgentIdentifier	= "(" "agent-identifier" ":name" word [":addresses" URLSequence] [":resolvers" AgentIdentifierSequence] (UserDefinedSlot Expression)* ")".
AgentIdentifierSequence	= "(" "sequence" AgentIdentifier* ")".
AgentIdentifierSet	= "(" "set" AgentIdentifier* ")".
URLSequence	= "(" "sequence" URL* ")".
DateTime	= DateTimeToken.

Pour générer les interpréteurs de messages ACL et SL0, nous avons essayé en premier lieu le générateur de compilateur javacc mais il s'est avéré être un générateur d'analyseurs LL(n) donc incapable de gérer les règles récursives à gauche. Nous avons donc utilisé deux outils dérivés de Lex et Yacc : JFlex et BYacc/J. Ces logiciels ont étés réécrits de manière à générer du code Java. Ainsi, à partir du champ lexical, JFlex découpe le message en morceaux, et à partir de la grammaire, BYacc/J analyse ses morceaux pour les interpréter. Pour générer des messages ACL, on respecte la grammaire FIPA, ainsi, pour chaque non-terminal, on a une classe correspondante. Ces classes ont deux utilisations principales, la première est de générer un élément de message syntaxiquement correct. La deuxième est de représenter informatiquement chaque élément des messages à décoder. Le respect de la totalité de la grammaire ACL permettra plus tard à PHOENIX d'échanger de l'information avec d'autres plates-formes FIPA. On peut envisager d'étendre la grammaire de façon à comprendre les langages KIF, FIPA-SL, ou FIPA-CL.

VII.4.4.4. Modélisation du système de transport de message



VII.4.4.5. Modélisation du système de décodage des messages



VII.4.5. La boite aux lettres

Lorsque l'agent reçoit un message (il vient de finir une tâche), il regarde si ce message est urgent ou non. S'il l'est, il va bloquer le rôle et appeler une méthode réservée. Une fois la méthode de traitement terminée, il va demander la récupération du rôle, ce qui n'est pas toujours possible. En pratique, toutes les variables sont sauvegardées, et un système de balises est mis en place : dépasser une balise permet la récupération au moins jusqu'à cette balise. Ceci évite un problème d'incohérence du déroulement d'un rôle. Cependant, il faut noter que les messages urgents sont très rares. La plupart du temps, lors de la réception du message, le rôle n'est pas interrompu mais le message passe au travers d'un filtre. Deux situations peuvent se produire :

- le message était urgent : la méthode garde le sémaphore et traite le message. Il peut s'agir d'un message de gain d'énergie, de mort de l'agent, ... Alors le message est traité entièrement. Ceci a deux avantages, d'une part de masquer aux rôles (et donc aux développeurs qui les écrivent) ce genre de messages, afin d'éviter des erreurs (intentionnelles ou non) créant des agents immortels, d'autre part, cela permet de bloquer toutes les méthodes critiques (pour éviter qu'un agent mort ne puisse recevoir encore quelques messages).
- le message n'était pas urgent : le sémaphore est libéré et le message est traité, sans bloquer l'appel à des méthodes critiques. Les messages de ce type sont souvent des messages liés au rôle, comme des requêtes à l'environnement, des messages d'acquittement, des transactions, ...

VII.4.6. Le market

VII.4.6.1. Principe de fonctionnement

Le Market sert à enregistrer les vendeurs ou les acheteurs pour chaque ressource présente sur la plate-forme. Cette entité est unique sur chaque plate-forme. C’est en quelque sorte un lieu d’échanges de ressources où les acheteurs / vendeurs peuvent s’inscrire pour une ressource souhaitée. Il s’est avéré que la base du Market s’adaptait bien à l’achat de compétences. En effet, les mécanismes d’achat / vente de ressources sont très proches des mécanismes d’acquisition / formation des compétences.

Un agent pouvant enseigner des compétences consultera s’il le souhaite le Market pour proposer ses connaissances à un agent inscrit en tant qu’acheteur et s’il n’y a pas d’acheteur il aura la possibilité de s’inscrire dans le Market pour les compétences qu’il peut enseigner au meilleur prix selon lui.

Un agent ayant la capacité d’apprentissage de compétences pourra rechercher sur le Market si un agent propose cette compétence, sinon il pourra s’inscrire en tant qu’acheteur dans celui-ci.

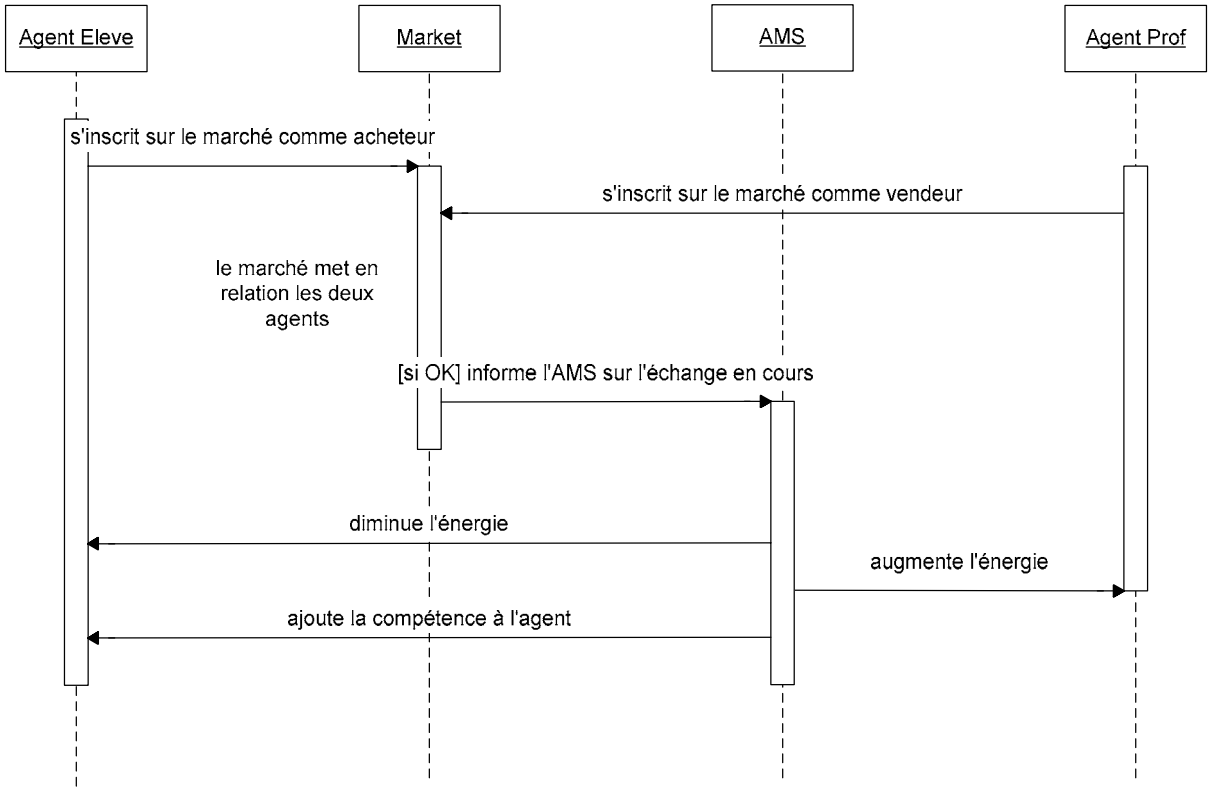
Son principe de fonctionnement est le suivant : pour acheter une compétence, l’agent va s’inscrire sur le marché en tant qu’acheteur de la compétence X à un prix P_X. Lorsqu’un agent désire acheter (respectivement vendre) une ressource, il indique le prix d’achat maximal (respectivement prix de vente minimal) de la ressource, et le type de la ressource voulue. Le type d’une ressource permet de les répartir en fonction de leur utilité.

Si un vendeur existe au prix indiqué, le marché va se charger de les mettre en relation, sinon dès qu’un vendeur se présentera sur le marché l’agent en sera notifié. Dans le cas contraire, si un agent ne s’est pas présenté pour acheter (respectivement vendre), l’agent impatient se désinscrit du marché.

Une fois que les agents se seront mis d'accord, le marché informera l'AMS qui sera chargé d'effectuer la transaction. En effet, il devra retirer le prix de cette compétence au taux d'énergie de l'agent acheteur (et inversement pour le vendeur) et transmettre la compétence à l'agent acheteur.

Ainsi cet investissement qui peut être assez lourd lui permettra à long terme d'effectuer des tâches qu'il ne pouvait faire jusqu'à présent. Possédant plus de compétences, il sera donc capable d'effectuer un plus grand nombre de rôle et ainsi rentabiliser cet achat.

Détaillons le mécanisme d'acquisition de compétences :



Acquisition de compétence par l'intermédiaire du Market

VII.4.6.2. Implémentation

Le Market est implémenté sous forme de Hashtable. Les clés sont des String et les valeurs associées des Vector de MarketInformation. La clé est constituée du type de la ressource, concaténée à la chaîne ".buy" si la ressource à être vendue, ou à la chaîne ".sell" dans le cas contraire. Une MarketInformation est constituée :

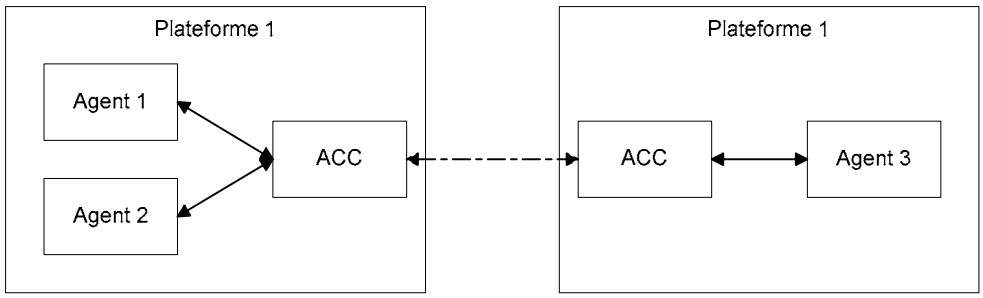
- l'AID de l'agent acheteur (resp vendeur)
- le RID de la ressource vendue, ou le type de la ressource voulant être achetée
- le prix à laquelle la ressource est achetée (ou vendue)
- la valeur du champ in-reply-to

Prenons comme exemple le cas où un agent désire acheter une ressource. Soit il n'y a aucun vendeur, dans ce cas l'agent est placé dans une file d'attente, soit il y a plusieurs vendeurs, tous placés dans la liste d'attente de vente. L'acheteur cherche alors celui qui vend au prix le moins cher, et la transaction s'effectue. Le vendeur est retiré de la liste d'attente.

La notion d'accointance peut cependant intervenir aussi dans le Market. Les agents appartenant à la liste d'accointance de l'agent sont favorisés dans la mesure ou l'agent possède déjà des connaissances sur eux et un coefficient de confiance.

VII.4.7. Communication inter-plates-formes

Dans l'optique de faire passer les messages d'une plate-forme à une autre, seule la méthode d'envoi de messages a besoin d'être distribuée sous CORBA. Nous allons donc distribuer la méthode void sendMessage(String message) de l'ACC, objet de Phœnix chargé du transfert de message. On obtient donc l'architecture suivante pour Phœnix sous CORBA :



La flèche en pointillés représente les communications inter-plateformes et les autres représentent les communications internes (intra-plateforme). Pour l'envoi interne de messages, cette méthode ne sera pas utilisée par l'intermédiaire de CORBA afin de ne pas alourdir la plate-forme. Seul le cas de messages envoyés entre deux plate-formes fera appel à la méthode de façon distribuée.

La programmation objet est prévue afin de concevoir des applications particulièrement importantes en taille, mais aussi facilement évolutives. Pour cette partie, nous avons décidé de séparer l'implémentation de la partie CORBA du reste de la plate-forme. Sous Phœnix, vous trouverez dans le package myCorba, la classe Gestionnaire qui centralise le mécanisme CORBA pour la plate-forme. Ainsi, si dans le futur, il est décidé de mettre l'ACC de Phœnix en distribution par l'intermédiaire du Serveur de nom de CORBA, il n'y aura que dans cette classe que les modifications devront être effectuées. De même, CORBA devrait évoluer au cours du temps, et si des modifications fondamentales doivent être effectuées, seule, cette classe devrait être à modifier.

VII.4.8. Migration d'agents

CORBA propose le système de « Fabrique ». Ce système est un service CORBA qui contrairement aux autres services, est une méthode pour permettre la création ou destruction d'objets à distance ou dans le cas que nous allons implémenter ici, une migration d'objets. CORBA ne propose pas seulement la méthode dans ce service, mais aussi un mécanisme transparent qui gérera les fabriques une fois celles-ci conçues et enregistrées auprès de CORBA.

La migration est une copie sur la plateforme destination suivie d'un effacement sur la plateforme d'origine. C'est un problème qu'on pourrait imaginer assez simple. En réalité, elle soulève beaucoup de problèmes dans le cas d'objets complexes.

En ce qui concerne la migration d'objets simples, cela ne pose en effet pas de problèmes particuliers. Considérons un objet quelconque que l'on appellera le Migrateur mais composé entièrement de type de base (int, char, float…). Dans ce cas, le mécanisme est assez simple. On va utiliser une méthode distribuée, qui prendra en paramètre d'entrée l'objet Migrateur et retournera un autre objet, copie du Migrateur. On appelle la méthode à travers le

réseau, la copie du Migrateur étant construite dans la méthode appelée, on obtient donc un nouvel objet qui se trouve lui même de l'autre coté du réseau. Finalement la méthode va retourner une référence sur cet objet distant. Dans le cas d'une migration, contrairement à une copie, il ne restera plus qu'à détruire l'original.

En revanche, lorsqu'il s'agit d'objets plus complexes, la migration n'est pas triviale. Dans un modèle orienté objet, la majorité des objets sont composés ou liés à d'autres objets. Prenons, par exemple, un objet Document qui est lié à un objet Dictionnaire dans lequel il est référencé. Le document contient un objet Graphique et un objet Logo. De plus le Logo est lié au graphique. Comment se passe dans ce cas la migration de l'objet document ? Quelles entités doivent être communiquées pour la migration du document ? Une entité contenue ou liée à l'objet migrateur doit elle migrer elle aussi ou juste être référencée à travers le réseau ? Que faire dans le cas où le graphe de relations entre objets contient des cycles ? Qu'en est-il de l'évolutivité du modèle objet ?

Nous nous sommes posés ces questions lorsque nous avons entrepris la migration d'agents cognitifs avec leurs rôles, tâches, croyances, désirs, intentions etc. Les concepteurs de l'OMG se sont aussi posés ces questions lorsqu'ils ont défini les spécifications du service de cycle de vie de CORBA. Ce service est particulier par rapport aux autres services de CORBA. En effet, la migration d'un objet est différente dans chaque cas. Certains objets migrent avec l'objet, d'autres sont uniquement référencés à travers le réseau ou dupliqués sur la machine d'arrivée. Le *CosLifeCycle* propose de fournir un mécanisme général à la migration d'un objet, mais de l'autre coté, pour chaque projet, le concepteur devra implémenter une partie du service suivant certaines règles. Des interfaces sont proposées pour guider et forcer le concepteur d'application à suivre la procédure d'implémentation d'un *CosLifeCycle*. C'est par l'intermédiaire de ces interfaces que le service de CORBA pourra mettre en œuvre facilement la migration d'objets. Nous verrons ultérieurement le mécanisme de ce service.

VII.4.8.1. Eléments à migrer

Un agent dans Phœnix est défini par quatre éléments :

- L'agent : l'entité elle-même.
- Rôle : le service ou la fonction remplie par l'agent. C'est une fonctionnalité générale que devra remplir l'agent.
- Tâche : un rôle est défini par un ensemble de tâches. Pour un rôle particulier, il y aura plusieurs actions à faire afin de le réaliser. Chacune de ces actions est une tâche. Certaines sont exécutées en parallèle ou en séquentiel.
- Compétence : C'est une unité de connaissance nécessaire à la réalisation d'une tâche.

Apres étude, voici les scénarii qui définissent les éléments qui doivent migrer en plus de l'agent. Ces plans de migration sont basés sur le lien particulier qui existe entre un rôle, une tâche et une compétence. Cette partie a été réalisée lors du développement des « Web services ».

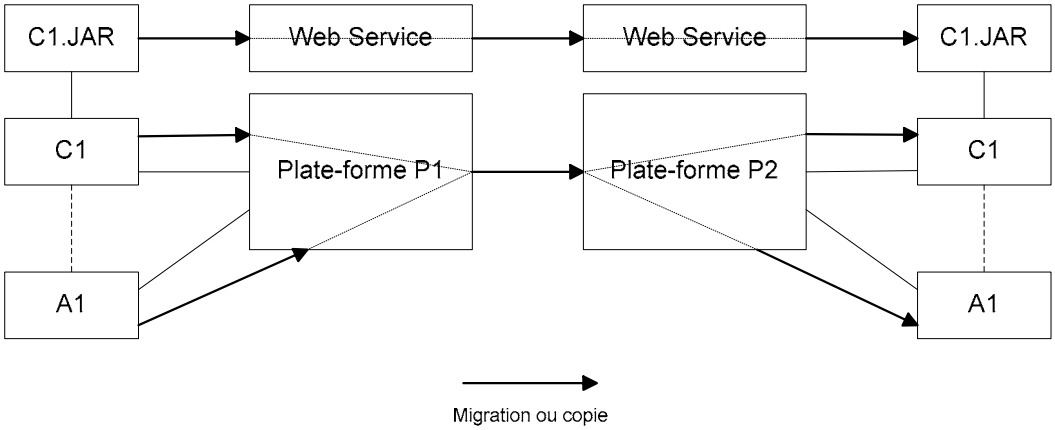
Le rôle et les tâches ne migrent pas. Seules les compétences vont migrer sous la forme de fichiers jar. L'avantage est que, seules, les compétences ont besoin de migrer, ce qui facilite le travail. Le principal problème est que pour réaliser un rôle, il faut que toutes les compétences nécessaires à ce dernier soient présentes sur la plate-forme, ce qui peut impliquer l'envoi de plusieurs compétences, une par une. Pour l'instant, c'est la solution qui a été retenue et qui servira de base pour l'implémentation de la migration d'agent dans cette version de Phoenix. Il est techniquement possible de migrer également le rôle et les tâches

mais ce n'est pas véritablement souhaitable dans la mesure où les rôles sont locaux à un environnement définit éventuellement que sur la plateforme d'origine. La migration d'un agent peut s'expliquer par des raisons de performance d'exécution.

VII.4.8.2. Scénario de migration d'un agent

Un agent A1 est lié à une compétence C1 qui lui donne accès à un fichier jar « C1.jar ». A1 et C1 sont des objets instanciés. Cet agent qui se trouve actuellement sur la plate-forme P1 souhaite migrer sur la plate-forme P2. L'algorithme de migration est donc :

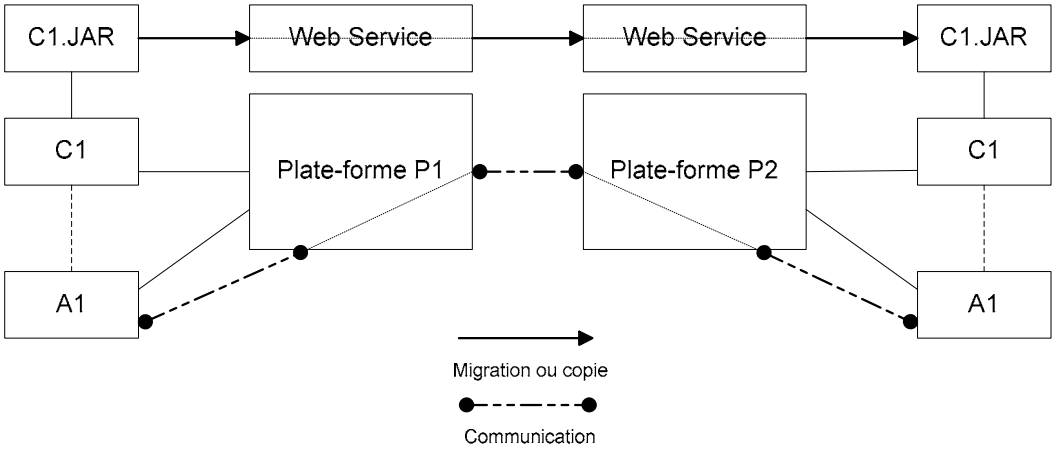
SI A1 est le seul à posséder C1 ALORS
A1 migre sur P2
C1 migre sur P2
SI C1.jar n'existe pas sur P2 ALORS
C1.jar migre sur P2
FIN SI
SINON
A1 migre sur P2
C1 est copié sur P2
SI C1.jar n'existe pas sur P2 ALORS
C1.jar est copié sur P2
FIN SI
FIN SI



VII.4.8.3. Scénario d'achat de compétences à distance

Un agent A1 se trouve sur la plate-forme P1 ainsi que la compétence C1 qui lui donne accès à un fichier jar « C1.jar ». Un agent A2 se trouve sur une plate-forme P2. A2 souhaite acheter la compétence « C1.jar » à A1. Une négociation intervient pour fixer le « prix » de la compétence. L'algorithme de migration de la compétence est le suivant :

Transaction entre A1 et A2 pour l'achat (la migration ou transfert de compétence)
C1.jar est copié sur P2
A2 créer un élément C1 sur sa plate-forme afin d'accéder à « C1.jar » de sa plate-forme
A2 paie A1 (transfert d'énergie)



VII.4.8.4. Scénario d'initialisation d'un agent

Sur la plate-forme P1 un agent A1 est créé. Cet agent à besoin de compétence donc il va les acquérir. Un agent distributeur va être créé et son rôle sera de lui fournir une compétence. Dans ce cas, A1 paye 0 à l'agent distributeur. Dans le cas où la compétence n'est pas présente sur la plate-forme, l'acquisition ne sera pas possible. Il faudra après sa création que A1 achète une compétence comme dans le scénario d'achat de compétence.

Même si ce scénario ne semble pas être un scénario de migration, le problème était de savoir ici si la compétence était récupérée à distance.

PS : la plate-forme gèrera ou pas la persistance de ses compétences sous la forme de fichier jar.

VII.4.8.5. Scénario d'achat d'une compétence que personne ne connaît

Un agent A1 est sur une plate-forme P1. Une plate-forme P2 possède une compétence C2.jar que personne ne possède. A1 souhaite acheter « C2.jar ». Un agent distributeur sera créé et A1 achètera à prix fort la compétence à cet agent. Deux cas se présentent alors :

Achat à distance

Dans ce premier cas, l'agent distributeur est créé sur la plate-forme P2 et A1 achètera la compétence à l'agent distributeur comme dans le scénario d'achat de compétence.

Ce scénario permet de récupérer l'implémentation du scénario d'achat de compétence en ce qui concerne la migration. Ceci permet un gain de temps lors de l'implémentation du scénario et d'un point de vue programmation orienté objet, une maintenance et une facilité d'évolution non négligeable. Ce scénario n'a pas été retenu pour des raisons que nous verrons ci-dessous.

Récupération de la compétence

La compétence sera d'abord récupérée sur la plate-forme P1 puis P1 créera un agent distributeur. L'achat se passera alors comme un achat de compétence normal entre A1 et l'agent distributeur. Ce scénario permet de récupérer l'implémentation d'achat de compétence au sein de la plate-forme. Le problème est que la récupération du fichier JAR devra être redéfinie.

C'est ce scénario qui est retenu étant donné que seul le fichier JAR sera à récupérer. Dans le scénario précédent, des communications supplémentaires entre les agents des plate-formes sont nécessaires.

VII.4.8.6. Scénario d'achat d'une compétence locale qui n'appartient à personne

Pour ce scénario, on retombe dans le cas des scénarii d'achat de compétence distante et d'achat de compétence que personne ne connaît. Un agent A1 se trouve sur la plate-forme P1 ainsi que la compétence « C1.jar » qui n'appartient à personne. A1 souhaite acheter la compétence « C1.jar ». L'algorithme d'achat est :

> *Un agent distributeur est créé avec un accès à C1.jar*
> *A1 achètera la compétence « C1.jar » à l'agent distributeur comme si c'était un agent normal.*

VII.4.9. Implémentation de la migration

Phoenix utilise une notion d'échange d'énergie pour auto-organiser les SMA. Cette énergie, pendant de l'économie de marché, intervient lors d'un transfert de compétence ou dans une migration d'un agent d'une plateforme à une autre. De même la notion de mémoire des agents constitue un ensemble d'objets à migrer ou non.

VII.4.9.1. Migration d'un agent réactif

L'état d'un agent est donné par deux variables :

- *Energy energy* : Représente l'énergie de l'agent. Lors de la migration, l'agent conservera son énergie. L'objet associé devra migrer avec l'agent.
- *boolean isAlive*: indique si l'agent est encore vivant. Si un agent migre, c'est qu'il est vivant. Donc la valeur n'a pas besoin de migrer et sera mise à « true » une fois l'agent migré. Si l'agent devrait mourir à l'arrivée, ce sera la nouvelle plate-forme qui se chargera de modifier cette valeur.

Par soucis de simplification, un agent devra terminer sa tâche en cours d'exécution avant de migrer et il n'acceptera plus de tâche déléguée avant sa migration. Aucun des champs suivant n'accompagnera l'agent lors de sa migration :

- *boolean performingDelegatedTask* : indique si l'agent remplit une tâche pour un autre agent. La valeur n'a pas besoin de migrer, elle sera à « false » une fois arrivée sur l'autre plate-forme.
- *Hashtable delegatedTaskList* : Contient la liste des tâches déléguées.
- *HashSet modifiedVars* : liste des variables modifiées par le roleMemory durant la délégation.

De même, l'agent devra terminer son rôle en cours avant de migrer. Le rôle ne migre donc pas avec l'agent. L'agent choisira un nouveau rôle après la migration. Aucun des champs suivants n'accompagnera l'agent :

- *AbstractRole role* : référence sur le rôle actuel de l'agent.
- *Memory roleMemory* : mémorise les informations pour le rôle que remplit l'agent.

De plus, d'autres variables ne sont pas nécessaires à conserver :

- *Memory memory* : Mémoire de l'agent, elle mémorise aussi bien des références sur des objets systèmes de la plate-forme (AMS…) que des chaînes de caractères, ou des interfaces. La mémoire de l'agent contient des références sur les éléments de la plate-forme où il est actuellement et qui ne seront pas conservées. On y retrouve l'identifiant de l'agent propre à la plate-forme. L'agent récupèrera un nouvel identifiant lorsqu'il arrivera sur la nouvelle plate-forme. Le nom de la plate-forme ne sera pas non plus conservé, puisqu'il changera. Un agent qui attend pour vendre ou acheter une ressource migrera une fois la transaction faite. Les compétences connues de l'agent sont stockées dans la *memory*. Ces dernières sont le savoir et l'expérience de l'agent. Ces informations accompagneront l'agent lors de sa migration. Afin de rendre plus rapide l'accès aux compétences (qui sont les éléments les plus utilisés dans le modèle YAMAM) et faciliter la migration, les compétences seront stockées dans une autre mémoire : la *skillMemory*. Ainsi le champ *memory* ne migrera pas avec l'agent, par contre, la *skillMemory* l'accompagnera.
- *Memory ressourceMemory* : C'est la mémoire où l'agent stocke les ressources qu'il a acquises. Cela fait partie de l'expérience personnelle de l'agent. Cette mémoire accompagnera donc l'agent lors de sa migration.
- *NameServeur ns* : mémorise une référence sur le NameServeur. Comme l'agent va changer de plate-forme, il se réfèrera à un nouveau NameServeur et donc cette valeur ne l'accompagnera pas durant sa migration.
- *Thread process*: Processus dans lequel l'agent s'exécute. Le processus ne migre pas avec l'agent. Un nouveau processus lui sera donné à son arrivée sur l'autre plate-forme. Nous avons présenté dans le chapitre dédié aux choix technologiques une version de JAVA faite à l'INRIA par Sara Bouchenak qui permettait une mobilité des processus. Cette solution n'avait néanmoins pas été retenue.

VII.4.9.2. Migration d'un agent cognitif

Un agent cognitif (CognitiveAgent) possède, par héritage, toutes les caractéristiques de la classe AbstractAgent avec quelques variables et méthodes supplémentaires dûes à sa spécialisation.

- *Vector accointance* : liste des agents ayant un lien d'amitié avec l'agent. Ce vecteur contient l'ensemble des RID des agents alliés. Comme l'agent peut entretenir des relations avec des agents appartenant à d'autres plates-formes, ce vecteur doit migrer avec l'agent.
- *Hashtable envMemory* : mémoire sur l'environnement de l'agent. Cet environnement va changer une fois arrivé sur l'autre plate-forme, il n'est donc pas nécessaire d'emmener cette mémoire lors de la migration.
- *Hashtable roleBeliefs* : mémoire des croyances sur les différents rôles. Cette Hashtable représente les croyances de l'agent acquises par expérience. Ces informations vont accompagner l'agent lors de sa migration. Elle contient une autre Hashtable, associée au nom du rôle qui possèdera trois informations : REUSSITE, ECHEC, TOTAL. Ces informations sont associées à un Integer, une valeur entière.

Finalement, l'agent emmènera donc avec lui l'état de ses objets *energy*, *skillMemory* et *ressourceMemory*. L'objet *memory* de l'agent ne l'accompagne pas, mais avait été prévu pour être un conteneur dans lequel le développeur pouvait stocker divers champs. Afin de permettre à un développeur d'affecter à un agent des informations propres à son application,

et afin que ces dernières l'accompagnent lors de sa migration, nous avons ajouté un *portableMemory* à l'agent. En complément, un *CognitiveAgent* emportera avec lui ses objets *accointance* et *roleBeliefs*.

VII.4.9.3. Mise en œuvre de la migration

La migration d'un objet entre deux programmes, nécessite un canal par lequel les informations vont transiter. Les deux programmes doivent s'entendre sur un protocole de codification de l'objet. Pour faire transiter les informations contenant l'agent qui migre entre deux plates-formes, nous avons décidé quelles seraient envoyées par l'intermédiaire de messages. Le système étant déjà mis en place sous le premier prototype Phœnix, le temps d'implémentation de la migration était ainsi réduit. De plus, cela évitait d'ouvrir un nouveau canal entre les deux plates-formes uniquement pour la migration ou de rajouter une méthode distribuée dans le partage sous CORBA.

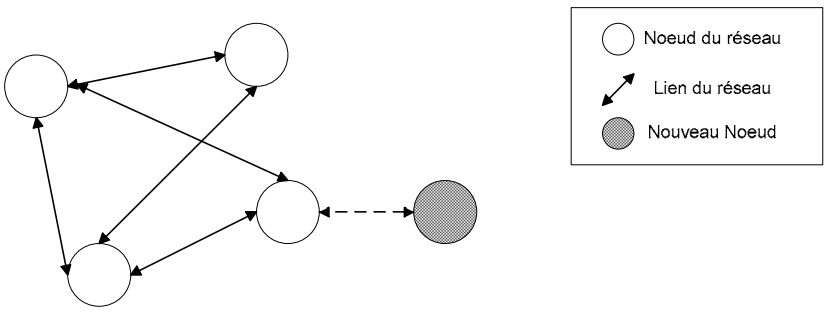
Le deuxième problème à résoudre, était la normalisation de l'objet pour envoyer les informations le concernant et le parcours du graphe d'objet. Le premier prototype implémentait un système de sérialisation java. Cette technique était intéressante dans la mesure où le langage s'occupe lui-même de la codification de l'objet. Cependant dans un souci d'interopérabilité, cette solution n'a pas été conservée. Nous avons trouvé une autre solution dans les « Web Services ». SOAP propose la codification des objets en XML avec ses règles exprimées dans une DTD. Cette solution offre la portabilité de l'encodage pour d'autres environnements de programmation comme le C++ ou tout autre langage dans lequel Phœnix sera implémenté dans le futur.

Nous travaillons cependant sur une version de Phoenix qui devrait permettre un meilleur découplage des tâches et des rôles, ainsi qu'une migration plus aisée.

VII.4.10. Une architecture P2P

L'architecture P2P est une architecture réseau où tous les nœuds du réseau possèdent les mêmes capacités et responsabilités. Nous avons décrit le fonctionnement du P2P dans le chapitre dédié aux différentes technologies utilisées dans le développement de Phoenix.

Un système multi-agents comme Phoenix est naturellement de type Peer-to-Peer où un nœud (une plate-forme Phoenix par exemple) peut envoyer et recevoir des agents ou des messages. Outre tous les avantages du P2P, il existe des problèmes inhérents à son architecture tels que la découverte des plates-formes (des nœuds) lors de l'intégration d'un nouveau noeud.



Un nouveau nœud connecté à un seul nœud doit parcourir tout le réseau pour retrouver les autres nœuds. Ce parcours peut prendre du temps. De plus, si un nœud ne fonctionne pas pendant le parcours, le nouveau nœud ne va jamais le retrouver (sauf en effectuant un nouveau parcours). La plupart des systèmes P2P sont conçus autour d'un serveur central dans lequel une nouvelle plate-forme peut s'inscrire et découvrir les autres plates-formes à partir de

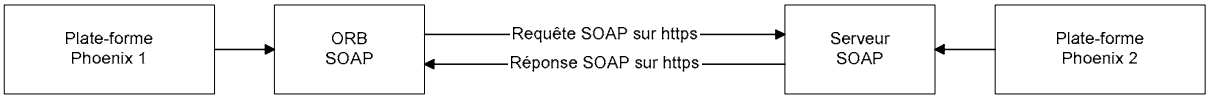
ce serveur. Le logiciel Napster utilise ce fonctionnement, néanmoins l'appellation P2P n'est pas pleinement justifiée de part la présence du serveur.

Pour des raisons de simplicité de codage, la première version de Phoenix imposait de lui spécifier manuellement la liste de tous ses voisins. Nous travaillons en ce moment sur une découverte automatique et dynamique qui trouve elle-même les voisins sans serveur central. C'est cependant un problème délicat à résoudre dans la mesure où il faut spécifier tout de même le champ de recherche. Une plateforme ne va pas aller rechercher toutes les autres plates-formes Phoenix existant sur Internet. Nous utilisons pour cela un masque IP pour restreindre la recherche à une ou plusieurs plages d'adresses déterminées. Ce paramètre sera situé dans le fichier XML contenant la configuration initiale de la plate-forme. Nous avons également développé une version de Phoenix avec un serveur central qui n'a pour unique but de référencer les plates-formes du réseau et de tester régulièrement leur état.

VII.4.11. Utilisation des WebServices

VII.4.11.1. Importation des compétences par les Web Services

A travers les *Web services*, un agent peut importer d'une autre plateforme une compétence qu'il n'aurait pas sur la plate-forme locale. Cette importation revient à invoquer une méthode à distance, `getSkill()` par exemple, utilisant SOAP. Pour ce faire, on a besoin des APIs permettant la requête d'un objet à distance utilisant SOAP. Ces APIs doivent aussi transformer la réponse SOAP en objet JAVA. Le groupement de ces APIs est une sorte d'ORB (*Object Request Broker*) pour le client SOAP. On appellera ce groupement « ORB SOAP ».



Architecture Phoenix + web services

Un exemple d'importation :

La plate-forme Phoenix_1 souhaite importer une compétence de la plate-forme Phoenix_2. Phoenix_1 va créer une requête SOAP et l'envoyer à travers https (http Sécurisé) vers le serveur SOAP. La réponse du serveur, qui est aussi en SOAP, va être envoyée du serveur à l'ORB SOAP.

Normalement, l'ORB SOAP est fourni avec le serveur web. Dans ce cas, on peut généralement dire que l'ensemble ORB SOAP et Serveur SOAP est un serveur web. On a besoin alors d'implémenter ce système d'échange de compétence et de le sécuriser.

Techniquement, importer ou exporter une compétence consiste à copier un fichier compilé par le réseau. Un fichier .class représente une `classe` Java compilée. Une compétence peut néanmoins être composée d'un package de classes java, ce qui représente plusieurs fichiers. Afin d'éviter une multitude de communications et un temps de transfert important, nous allons regrouper ces classes dans un package java compressé. Le format de fichier .JAR est le fichier archive standard proposé par JAVA. Les W*eb services* seront chargés de récupérer ou d'envoyer ces .JARs. Le nom du fichier .JAR sera le même que le nom de la compétence qu'il détient.

Nous avions deux alternatives pour importer les compétences des agents :

- Un service par compétence : en se basant sur la notion « une compétence égale un service ». Si l'agent souhaite obtenir une compétence, il va invoquer le

service correspondant à cette compétence. Autrement dit, il y a autant de services que de compétences.

- En utilisant un « service import ». Un agent qui souhaite obtenir une compétence va toujours invoquer ce même service. Le nom de la compétence voulue sera mis comme paramètre du service import. Dans ce cas, il y aura seulement un service.

Pour des raisons de modularité propre au génie logiciel et pour des raisons de rapidité de fonctionnement, nous avons retenu la deuxième solution. En outre, cette solution nous simplifiait le développement puisqu'elle nous évite le besoin de « découvrir » des services. Ceci entraîne l'élimination du protocole UDDI et les registres des *web services*.

VII.4.11.2. Choix du serveur web

Nous avons choisi la solution serveur web intégré et compte tenu du tableau comparatif présenté dans le chapitre lié au Choix Technologiques, nous nous sommes orientés sur le serveur GLUE qui est facile à programmer, gratuit et de petite taille.

Cependant, afin de respecter les concepts de modularité, nous souhaitons nous réserver la possibilité de changer de serveur web très facilement et remplacer GLUE par un autre si le besoin s'en faisait sentir. Cela requiert une certaine indépendance fonctionnelle du serveur web par rapport à Phoenix. Nous avons pour cela conçu adjoint une classe « interface » entre le serveur web et Phoenix qui sera la seule à modifier si l'on désirait changer de serveur web, SOAP étant un standard.

VII.4.11.3. Une solution au transfert de compétences entre agents

Chaque agent dans Phoenix possède son propre *thread* et tous les agents sont aussi capables d'importer des compétences. Cela implique une gestion des accès concurrents par le serveur. Les *Web services*, constituant un groupement des méthodes à exécuter, seront modélisés comme un élément de compétence. Un Web Service est en quelque sorte un cas particulier d'une compétence Phoenix. Nous allons donc faire hériter les méthodes de la classe *WebServices* de la super-classe *AbstractSkill*.

Un agent souhaitant importer des fichiers .JARs d'une autre plate-forme Phoenix va utiliser sa compétence « *ImportDistance* ». On aura alors une architecture sous la forme :

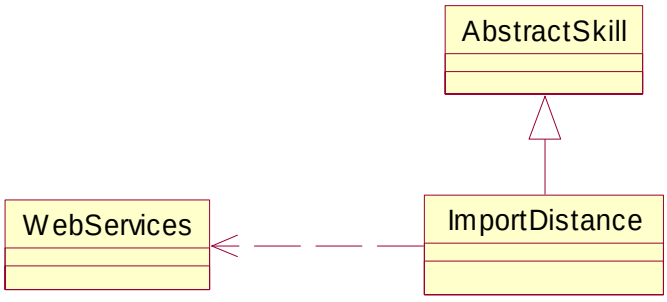
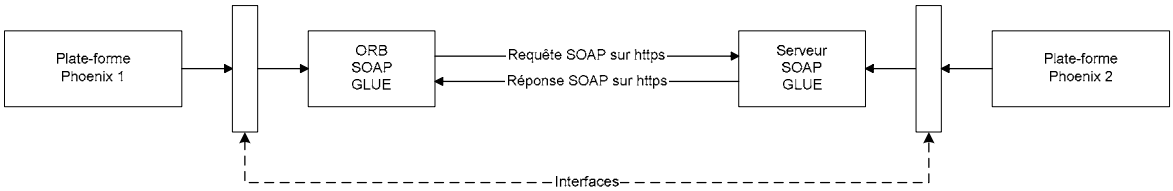


Diagramme de `classes` UML pour l'importation

ImportDistance est la compétence responsable pour importer des fichiers .JARs à travers les W*eb services*. Le nom *ImportDistance* est donné à titre d'exemple.

VII.4.11.4. Une solution au faible couplage

Afin de minimiser le couplage entre le serveur GLUE et Phoenix, nous avons imaginé de placer des classes interface entre les deux. Il devient ainsi possible de remplacer GLUE par un autre système en changeant simplement quelques méthodes de ces classes interface.

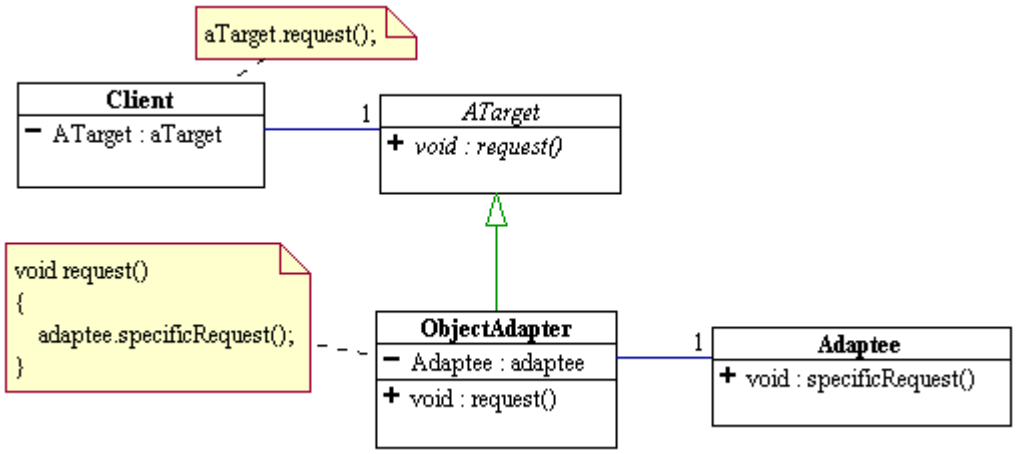


Phoenix et GLUE en accès indirect

Avec cette solution, il devient possible de remplacer facilement GLUE par un autre produit (par exemple, Apache SOAP). Le changement à effectuer est localisé dans ces interfaces et aucun changement n'est nécessaire dans le code de Phoenix.

VII.4.11.5. Un modèle d'abstraction

Nous avons utilisé le concept des design patterns pour implémenter cette interface. C'est un modèle d'architecture d'objets pour résoudre certains problèmes connus. Dans notre cas, une solution d'interface est le design pattern *adapteur*.



Le *design pattern adapter*

Si l'on souhaite par exemple découpler la classe *Adaptee* (équivalent au serveur SOAP ou ORB SOAP) et Client (équivalent à notre Phoenix), on peut mettre une classe *adapter* (ici *ObjectAdapter*) plus une classe abstraite. Le Client va pouvoir accéder à *Adaptee* au travers de l'interface *ATarget*. Dans l'hypothèse ou l'on souhaite remplacer un *Adaptee* par un autre (remplacer un serveur web par un autre), il suffit de réécrire l'*ObjectAdapte*r et il n'est pas nécessaire de reprogrammer le Client (Phoenix). Les parties serveur et client sont bien découplées.

VII.4.11.6. Implémentation de cette abstraction

Dans la figure précédente présentant le design pattern *adapter*, on constate que la classe *ObjectAdapter* hérite de la classe *ATarget*. Dans l'implémentation de ce modèle, nous allons utiliser l'implémentation d'interface au lieu d'un héritage simple : La classe *ATarget*

sera une interface. L'avantage de l'interface repose que le fait que le programmeur doit implémenter toutes les méthodes de l'interface. On aura alors deux *adapters* : une pour le serveur SOAP, l'autre pour l'ORB SOAP. Les diagrammes de classes suivants vont résumer notre implémentation.

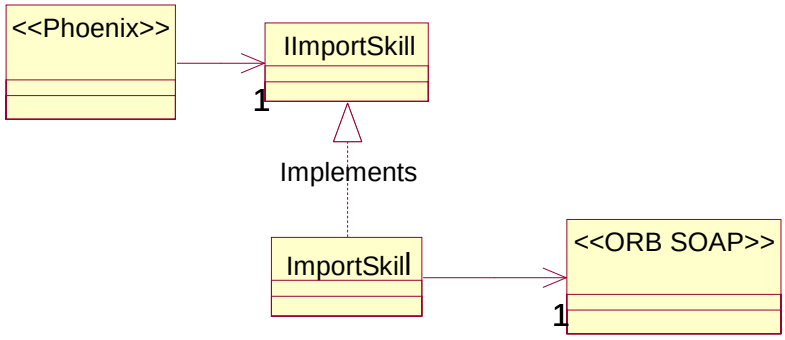


Diagramme de `classe` UML - Liens entre Phoenix et ORB SOAP

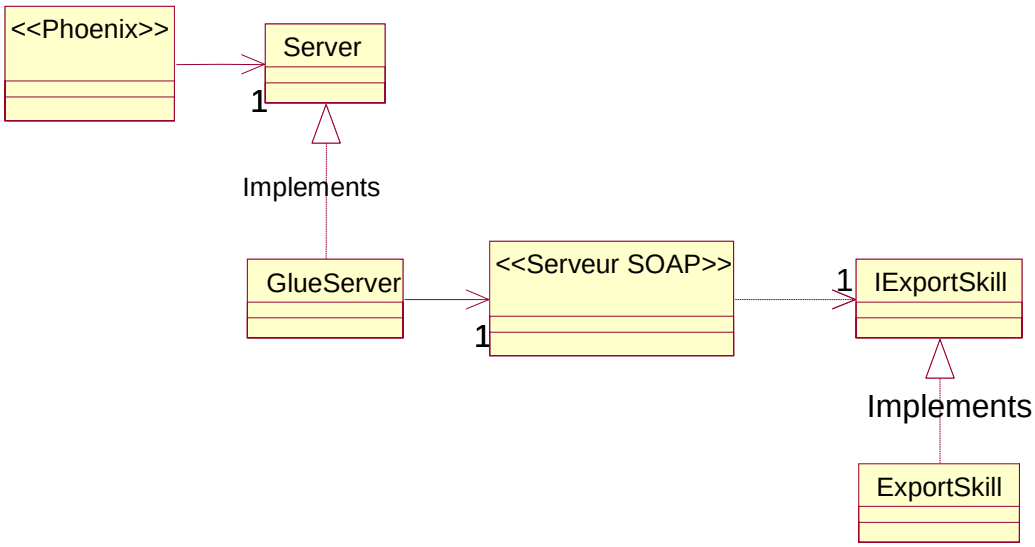


Diagramme de classe UML - Liens entre Phoenix et le serveur SOAP

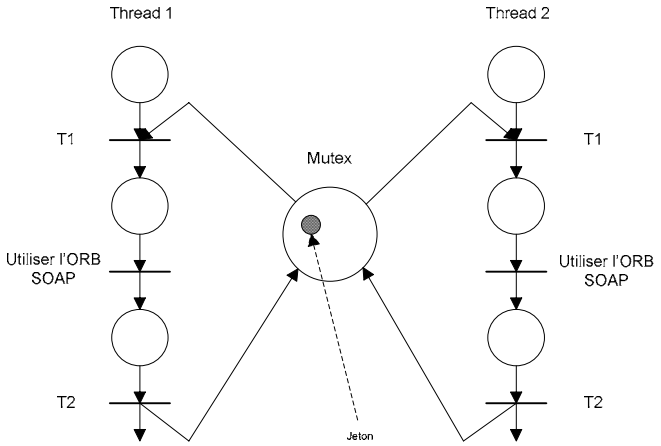
Dans la figure ci-dessus, on voit une classe, *ExportSkill* et son `interface`, *IExportSkill*. La classe *ExportSkill* est en fait la `classe` qui est déployée en tant que *web service*. Dans les deux cas, Phoenix est séparé du Serveur SOAP et de l'ORB SOAP.

Si on désire changer un serveur par un autre, il suffit de modifier les quatre `classes` : *ImportSkill, GlueServer, IExportSkill, ExportSkill*.

VII.4.11.7. Gestion des accès concurrents aux compétences

Le fait que chaque agent de Phoenix ait son propre *thread,* nous force à traiter le problème d'accès concurrents. Phoenix doit accéder à l'ORB SOAP pour envoyer des requêtes SOAP ou pour recevoir la réponse SOAP. Comme chaque agent a la compétence d'accéder à cet utilitaire, il est probable qu'un problème survienne si plusieurs agents accèdent simultanément à l'ORB SOAP. La capacité de traiter plusieurs thread à la fois sans problème s'appelle *thread safe*. Certains serveurs webs (par exemple GLUE et WASP) sont déjà *thread safe*, mais dans l'hypothèse ou on remplacerait GLUE par un autre serveur web qui n'ait pas cette caractéristique, il est souhaitable de le gérer directement dans Phoenix.

L'une des solutions architecturale est de limiter le nombre de *threads* qui peuvent utiliser l'« ORB SOAP simultanément. Seulement un thread peut utiliser l'«ORB SOAP». Cette solution s'appelle mutex (*mutual exclusion*). Le fonctionnement du *mutex* est expliqué sur le diagramme de Petri suivant :



Si un des *thread*, arrive à T1, il va attendre et prendre le jeton du *mutex*. Cette action va bloquer l'autre thread tant que le jeton du premier thread n'est pas relâché. Cette méthode permet de bloquer l'accès simultané à l'ORB SOAP. Fort heureusement, la langage JAVA gère nativement ce principe du mutex en permettant la simple déclaration d'un méthode en synchronisée (*synchronised).*

VII.4.11.8. Exportation des compétences par les Web Services

Afin de masquer la complexité de l'accès à certaines méthodes et classes des Web Services, nous avons créé une « *façade »* entre Phoenix et la partie *Web services*. Cette façade va offrir des services statiques (méthodes statiques) pour importer en ne demandant au développeur que l'essentiel. Cette *façade* est la classe « WebService ».

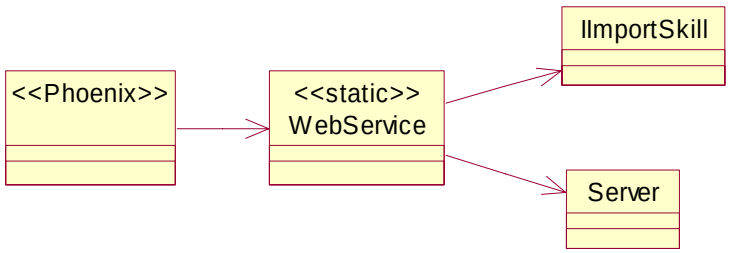


Diagramme de `classe` UML - Architecture Phoenix + Façade

L'*ExportSkill* est l'objet que l'on va déployer en tant que *web service.* Il intègre une `méthode` à distance qui retourne une réponse attachée avec un fichier .JAR contenant la compétence souhaitée. L'*IExportSkill* est son interface. Un exemple d'interface *IExportSkill* :

```
public interface IExportSkill {
      public MIMEData GetJar(String skillName)
}
```

La méthode *GetJar* va prendre le nom de la compétence et retourne un MIMEData contenant le fichier .Jar comme attachement. Le type MIMEData est une classe fournie avec

le serveur web GLUE. Cela veut dire que la classe *ExportSkill* et son interface sont propres au serveur web et doivent être recodés si on change de serveur.

La séquence d'exécution d'une importation de compétence est représentée par le diagramme de séquence suivant :

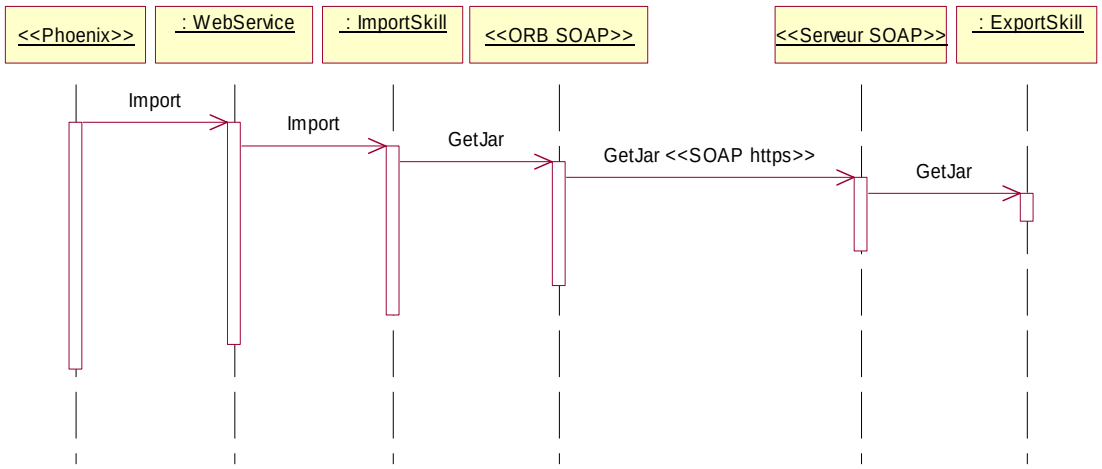


Diagramme de séquence UML d'ensemble d'architecture de *web services*

Phoenix, WebService, ImportSkill et ORB SOAP représente le coté client. Serveur SOAP et ExportSkill, le coté serveur.

VII.5. Le Rôle

VII.5.1. Le nombre d'instances d'un rôle

Un système de jetons représentant le nombre de rôles a été implémenté. En effet, lors de la création d'un rôle sur la plate-forme il est possible d'indiquer le nombre de jetons disponibles simultanément pour ce rôle. Ce nombre indiquera combien d'agents peuvent exécuter ce rôle au même moment. Par exemple, admettons qu'un rôle consiste à scruter l'arrivée de mail sur une plate-forme au sein d'un Système Multi-Agents consistant à optimiser les performances d'un réseau. Alors ce rôle n'aura peut être pas grand intérêt à être effectué simultanément par deux agents différents. On peut donc instaurer un unique jeton pour ce rôle.

La possibilité est aussi offerte d'instaurer un nombre infini de jetons ce qui permet à un rôle d'être exécuté sans limite autant de fois que souhaité.

Cette fonctionnalité permet de « contrôler » le bon fonctionnement d'une plate-forme et ainsi de veiller à ce qu'elle effectue les fonctionnalités souhaitées.

VII.5.2. La délégation de tâches

Pendant l'exécution d'un rôle, l'agent doit exécuter une succession de tâches. Pour être exécutées, ces différentes tâches nécessitent la possession de compétences spécifiques. Mais si l'agent ne possède pas la ou les compétences nécessaires, il peut déléguer cette tâche (si elle est déléguable) à un autre agent qui possède les compétences requises.

Il va donc lui falloir rechercher un agent à qui déléguer cette tâche. Une fois trouvé, cet agent effectuera la tâche pour un certain prix. C'est ce genre de relations entre agents qui leur permet d'élargir leur réseau d'accointances.

Exemple :

Soit un rôle R1 qui se compose des tâches T1, T2, T3, et qui rapporte X points d'énergie à l'agent qui l'effectue (donnés par la plate-forme).

T1 nécessite la compétence C1 pour son exécution
T2 nécessite la compétence C2 pour son exécution
T3 nécessite la compétence C3 pour son exécution
Donc la tenue du rôle R1 nécessite C1, C2, C3

Afin de pouvoir choisir ce rôle R1, l'agent a trois alternatives :

Il doit déjà posséder les compétence C1, C2 et C3
Il peut les acquérir définitivement
Il délègue la tâche Ti à un autre agent qui aura la compétence Ci

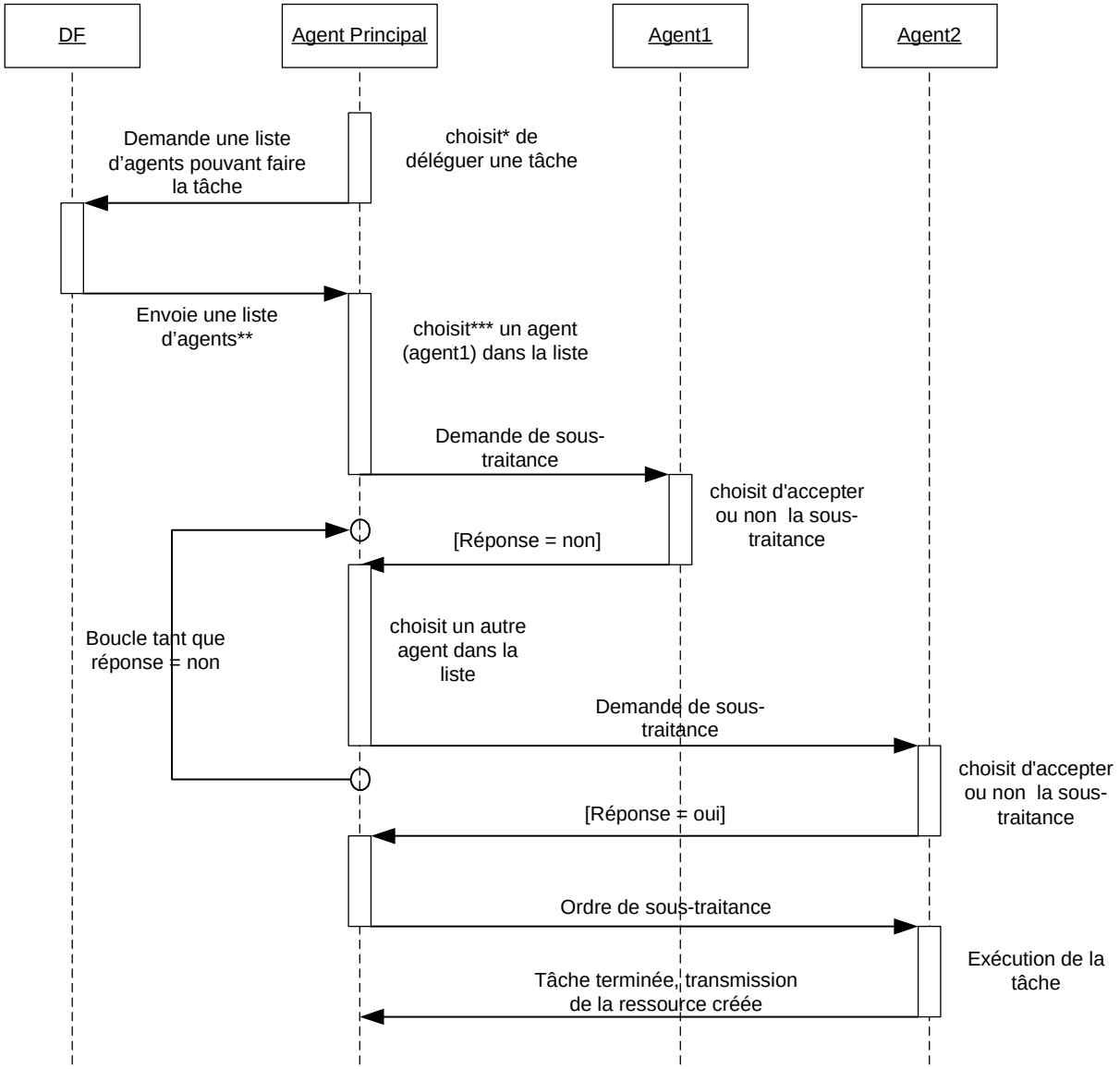
Une fois que l'agent aura exécuté T1, T2, T3 alors il aura remplit le rôle R1 et son compte sera crédité de X points d'énergie.

VII.5.2.1. Annonce d'une tâche disponible

Un agent a deux méthodes pour sous-traiter une tâche. La première consiste à contacter directement les agents pour leur signaler la disponibilité de la tâche (méthode push). La seconde consiste à référencer la tâche déléguée dans une « place de marché ».

En ce qui concerne l'annonce directe, la recherche d'un autre agent à qui déléguer la tâche se fait en premier lieu avec son réseau d'accointances (c'est à dire les agents avec lesquels il a déjà eu des contacts) et en consultant le Directory Facilitator (le service d'annuaires de type pages jaunes) qui lui indiquera la liste des agents de sa plate-forme ou d'une plate-forme distante possédant les compétences pour exécuter cette tâche. Il envoie successivement un message à chaque agent trouvé lui demandant s'il veut effectuer la tâche en échange d'une rémunération. L'agent qui reçoit le message décide alors d'accepter ou non la mission. Ensuite le message s'auto-détruit ☺

Un agent signale son acceptation ou son refus en retournant un simple message à l'émetteur de l'annonce. Après avoir retourné le message l'agent crée un processus pour exécuter la tâche demandée. A la fin de l'exécution de cette tâche, l'agent renvoie un message pour indiquer qu'il a terminé d'exécuter la tâche. Ce message indiquera si la tâche a été bien exécutée et pourra transmettre le résultat de cette tâche (l'exécution d'une tâche pouvant créer une ressource comme résultat).



* : le choix est différent selon le type d'agent (réactif ou cognitif)
** : la liste comporte en priorité des agents locaux puis des agents distants
***: le choix est différent selon le type d'agent (réactif ou cognitif)

Délégation d'une tâche par un agent réactif

VII.5.3. Choix d'un rôle par un agent

Nous avons vu que l'agent réactif exécutait toujours le même rôle. Ce rôle lui est demandé par la plate-forme. Au contraire, un agent cognitif doit pouvoir choisir « intelligemment un rôle » qui lui rapportera de l'énergie à plus ou moins long terme. Aussi, l'agent devra créer des plans d'actions qui comporteront des séquences de rôles successifs à exécuter.

Dans le diagramme ci-dessous, l'agent A va se renseigner auprès de la plate-forme sur les rôles disponibles (ici R1 et R2), puis va choisir un plan d'action P1 composé de l'exécution du rôle R1 deux fois consécutivement.

A noter qu'un agent cognitif choisi un rôle en fonction de l'énergie qu'il va lui rapporter. Pour calculer ce gain, l'agent parcourt le rôle, composé de plusieurs tâches ; chaque

tâche étant liée à une compétence. Le coût de revient du rôle correspond à la somme des compétences nécessaires pour le réaliser ajoutée à la somme des ressources consommées. Lorsqu'une tâche du rôle doit être sous-traitée, l'agent doit préalablement négocier avec l'agent sous-traitant avant d'endosser le rôle. C'est une fois que le prix de revient d'un rôle est déterminé totalement que l'agent décide s'il est rentable ou non de l'exécuter.

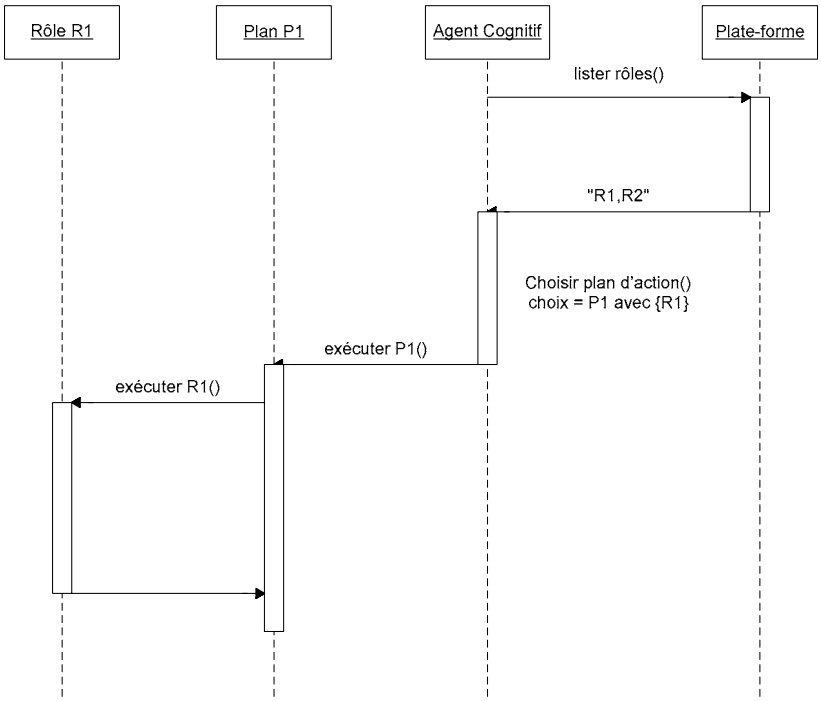


Diagramme de Séquence UML du Choix d'un rôle par un agent cognitif

VII.5.4. Exécution d'un rôle par un agent

Après avoir choisi un rôle (ou qu'on lui en ait imposé un, ici R1), l'agent va exécuter consécutivement les différentes tâches qui composent le rôle (ici T1 et T2).

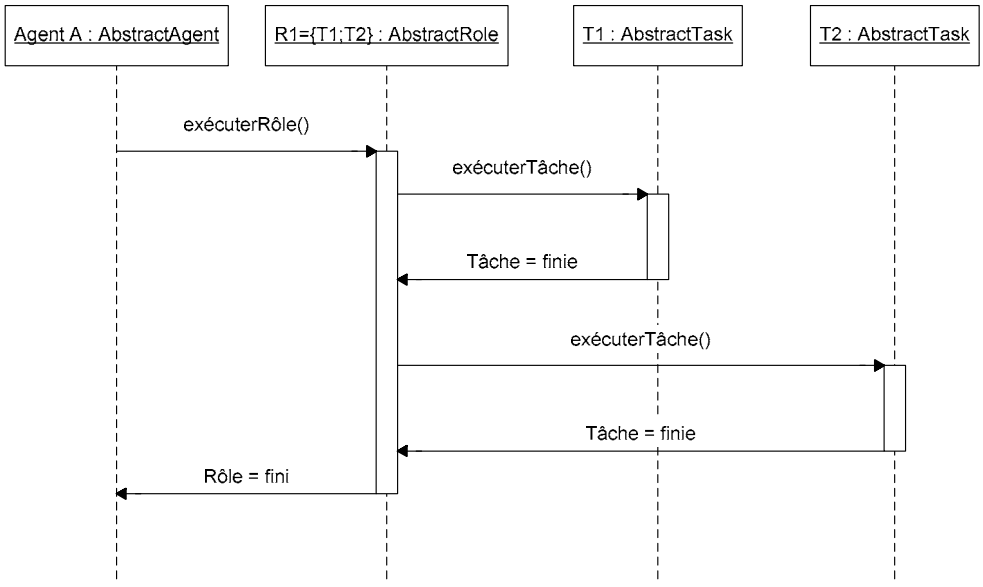


Diagramme de Séquence d'exécution UML d'un rôle R1 par
un agent ayant toutes les compétences nécessaires

VII.5.5. Estimation du prix de vente d'un rôle

Afin de choisir un rôle, un agent cognitif doit avant tout estimer l'énergie que lui apportera la tenue du rôle. Il va donc décomposer le rôle en tâches et ensuite estimer le prix de chacune d'elles. Ce prix correspond au prix de la compétence qu'elle nécessite (s'il ne la possède pas déjà) plus le prix des ressources qu'elle consomme. Mais ce prix des ressources est variable et l'agent n'a aucune certitude sur ce prix. Il ne s'agit que d'une estimation.

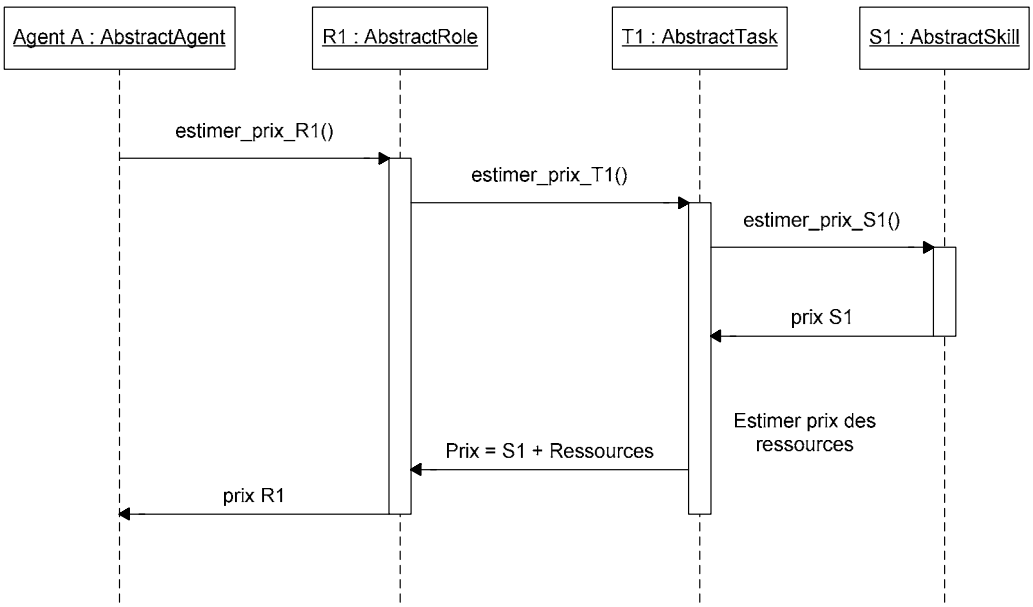


Diagramme de Séquence UML pour l'estimation du prix de vente d'un rôle R1composé d'une tâche T1

VII.5.6. Choix d'un rôle par un agent cognitif

La première étape consistera à récupérer la liste des rôles disponibles et la quantité d'offres sur ce rôle c'est-à-dire le nombre d'agents qui peuvent encore acquérir ce rôle. Pour chacun des rôles, l'agent cognitif calculera ce que rapportera ce rôle, le nombre de tâches dont l'agent n'a pas la compétence et les ressources nécessaires. On distingue deux cas où interviennent des prises de décisions différentes :

VII.5.6.1. Cas d'urgence

L'agent cognitif est dans ce que l'on appelle la **zone rouge**, c'est-à-dire que son niveau énergie est tellement faible qu'il doit rapidement acquérir de l'énergie s'il ne veut pas mourir. Ce niveau est défini par rapport à son niveau d'énergie et les taxes qu'il subit.

- L'agent fait la liste des rôles pour lesquelles il n'a aucune délégation à faire et il possède toutes les ressources nécessaires. Pour chaque élément de la liste, on affectera une valeur de priorité suivant la mémoire de l'agent sur ce rôle (croyances). Les paramètres entrant en jeu sont : son taux de réussite, son temps d'exécution et l'offre

sur ce rôle (ces paramètres sont décrits plus bas). Ceci fait, l'agent pourra choisir un rôle qui lui rapportera plus que le seuil minimal acceptable qu'il a déterminé.

- S'il n'a pas trouvé de rôle sans délégation, l'agent doit alors en choisir un parmi les rôles restants et calculer pour chacun d'eux le coût de délégation et d'achat des ressources nécessaires. Pour choisir parmi ces rôles, l'agent utilisera ses croyances sur le rôle et sur les agents à qui il pourrait déléguer la tâche (il peut utiliser son réseau d'accointances qui contient la liste des agents avec qui il a déjà eu des contacts). Ceci fait, l'agent pourra choisir un rôle qui lui rapportera plus que le seuil qu'il a déterminé.
- S'il n'a pas choisi de rôle, on recommence la boucle avec un seuil d'acceptation plus bas.

VII.5.6.2. Cas de non urgence

L'agent est dans ce que l'on appelle la **zone verte**. Il n'a pas de contrainte d'urgence.

- L'agent examine tous les rôles que ce soit avec ou sans délégation, qu'il ait les ressources nécessaires ou non. Il leur applique les paramètres qu'il a en mémoire pour chacun d'eux.
- L'agent choisi le meilleur rôle.
- Si n'a pas pu choisir de rôle, il recommence la boucle en diminuant le seuil d'acceptation d'un rôle.

Pour bien choisir un rôle, l'agent cognitif pourra évaluer la valeur d'un rôle, grâce aux informations qu'il aura sur les tâches qui le composent. En effet pour chaque tâche qu'il aura rencontré (qu'il aura exécuté ou délégué) l'agent stockera des informations telles que le taux de réussite de cette tâche, la véracité des informations (confiance dans les informations), la rapidité d'exécution de la tâche et le nombre de fois qu'il aura délégué cette tâche. Grâce à ces informations, l'agent est à même d'évaluer un rôle en évaluant toutes les tâches qui le compose.

A noter que l'agent peut décider d'entreprendre une migration vers une autre plate-forme s'il juge que les rôles qu'il effectue ne lui rapportent pas assez. Dans ce cas, un test sera effectué pour définir s'il y une possibilité de migrer et si oui, à quel prix.

Par la suite, la solution proposée ci dessus sera amenée à évoluer. Aussi, notre implémentation permettra **l'intégration d'une technique de Q-Learning**. En effet, cette technique nous est apparue comme celle répondant le mieux à notre problème car l'agent pourra alors décider de planifier ses actions et utilisera une fonction de récompense qui le guidera pour ses décisions futures.

VII.5.6.3. Structure de données nécessaire au décisionnel

Voici une liste des données nécessaires au processus décisionnel de l'agent cognitif. Cette liste constitue la structure minimale, il est bien sur possible de surcharger la classe et d'ajouter de nouveaux attributs et méthodes à la classe dérivée :

- Energie
- Taxe subie
- Croyances sur les tâches (décrites ci-dessus)
- Réseau d'accointances : contient une liste des agents avec lesquels il a été en relation.
- Seuil d'acceptation d'un rôle (énergie minimale que doit rapporter un rôle pour être choisi).

- Seuil d'acquisition d'une compétence (investissement maximal pour l'acquisition d'une compétence).

Le décisionnel intervient également pour les tâches déléguées. Pour chaque demande de délégation que l'agent reçoit, il calcule tout d'abord le seuil d'acceptation de la tâche suivant le coût d'exécution de la tâche et son niveau d'énergie. Puis il calcule la valeur de la tâche avec le prix qui lui a été proposé et les paramètres de la tâche qu'il a en mémoire (taux de réussite...). On compare cette valeur au seuil établi auparavant pour accepter ou non la tache.

VII.5.6.4. Délégation de tâches chez l'agent cognitif

Après avoir demandé à l'*AMS* (agent noyau) si le rôle peut être exécuté, l'agent cognitif exécute chacune des taches du rôle. Pour chaque tâche, l'agent vérifie s'il est mieux pour lui, en terme de rentabilité, de l'exécuter lui-même ou de la déléguer. Il déléguera la tâche s'il n'a pas les compétences ou les ressources nécessaires à son exécution ou s'il n'a pas eu de bons résultats (erreurs d'exécution…) sur cette tâche dernièrement. S'il choisit de déléguer, il regarde si l'un des agents de son réseau d'accointances possède la compétence et quels sont ses résultats dans l'exécution de cette tâche, sinon il demande au *Directory Facilitator* de lui donner une liste d'agents capables d'effectuer la tâche. S'il ne peut toujours pas déléguer la tâche, il ne lui reste plus qu'à l'exécuter lui-même s'il le peut ou sinon il génère une erreur et auquel cas, l'exécution du rôle est arrêtée.

A chaque délégation de tâche, l'agent note qu'il a délégué cette tâche ; après un certain seuil, si une tâche a souvent été déléguée et que l'agent n'a pas la compétence pour l'exécuter, il pourra décider d'acquérir cette compétence s'il le peut.

VII.6. Les tâches

Un rôle comprend un ensemble de tâches à exécuter séquentiellement ou parallèlement et suivant un ordonnancement particulier. Nous avons mis au point un langage de script afin d'autoriser l'exécution de plusieurs tâches en parallèle ou en boucle et d'ajouter des conditions. Un analyseur lexical et syntaxique était donc requis ainsi qu'un interpréteur permettant d'exécuter les rôles des agents.

De nombreux outils sont disponibles pour créer un analyseur interpréteur de script. Cependant, ils ont tous des caractéristiques particulières qui font qu'ils sont plus ou moins adaptés au problème posé. Ainsi, de nombreux outils ne s'attardent que sur l'aspect analyse lexical et syntaxique. Or une fois cette phase opérée, on souhaite exécuter le script. Pour cela certains outils construisent un arbre représentant le script fourni en entrée de l'analyseur. Reste ensuite à développer l'interpréteur qui se base sur cet arbre. De plus, la grammaire que nous allions utiliser allait être ambiguë en raison notamment des nombreuses parties facultatives des expressions :

if (expression) then instruction(s) [else instruction(s)]

Il fallait donc être certain que l'analyseur pourrait lever ce genre d'ambiguïté.

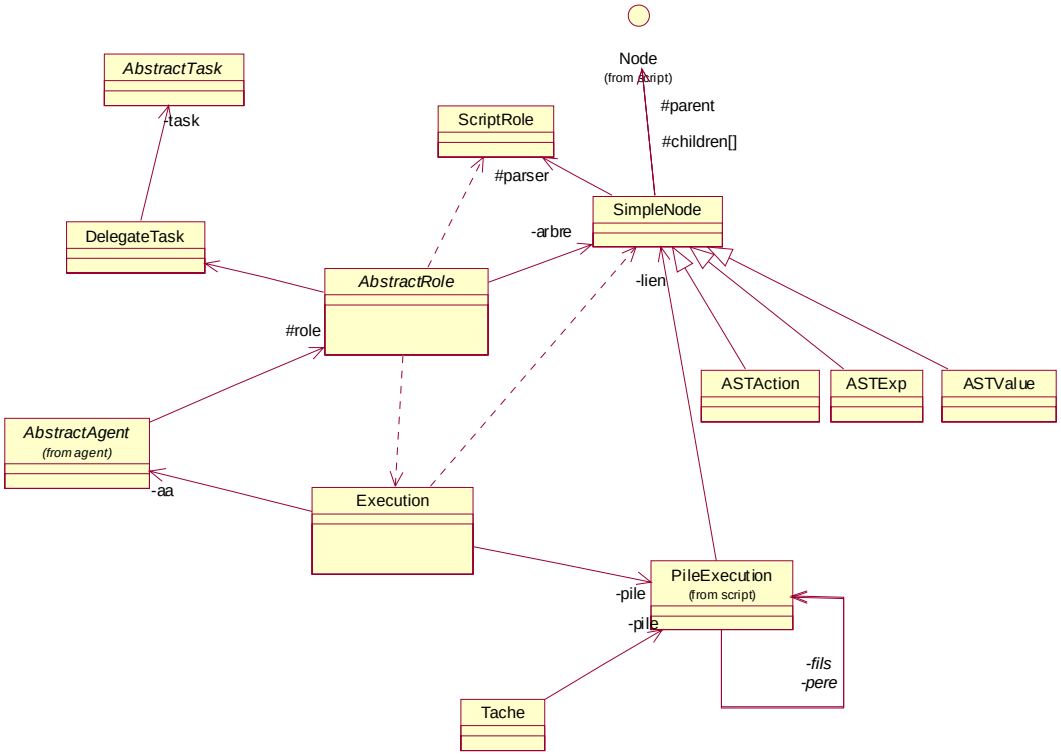
Les compilateurs classiques sont de types LALR(1) (Yacc et bison sont de ceux-ci). Cependant ils ne permettent pas de compiler des grammaires contenant certaines ambiguïtés (le classique « else » facultatif dans un test). JAVACC offre la particularité d'être LR(k)

localement [Autebert, 1987]. Un compilateur de grammaire LR(k) est très coûteux en ressource comparé à un compilateur de grammaire LALR(1) qui est plus performant mais aussi plus restrictif au niveau de la grammaire. JAVACC allie les avantages des deux types de compilateurs. Nous avons évalué plusieurs produits et notre choix s'est porté sur JAVACC[56]. C'est un des plus populaires et reconnu par SUN.

De nombreux scripts ont donc été écrits afin de valider l'analyseur et l'interpréteur. Un toy-problem fut aussi créé pour mettre en évidence les nouvelles fonctionnalités apportées par l'analyseur interpréteur. Cette application simule une chaîne de montage d'automobile avec l'exécution en parallèle de nombreuses tâches. Vous pourrez la découvrir en annexe.

VII.6.1. Définition du package Script dans Phoenix

La classe principale du package script est la classe *Execution*. Celle-ci est le lien entre la pile d'exécution, l'arbre traduisant les scripts et la classe *AbstractRole*, chargée d'exécuter les rôles. *AbstractRole* utilise les classes *ScriptRole* (pour analyser les scripts et construire les arbres) et *Execution* (pour interpréter les scripts). La classe *Execution* quant à elle, contient une référence vers la pile d'exécution, vers l'arbre correspondant au script et vers l'agent qui exécute le script.



Architecture du package script et interaction avec Phœnix

VII.6.2. Lancement d'un script

La classe *AbstractRole* permet de définir l'enchaînement des rôles qu'un agent pourra exécuter. Pour cela, on définit la classe *Xrole* (où X est le nom de son rôle) qui hérite de la classe *AbstractRole*. Cette classe contient au minimum la définition du constructeur qui peut se limiter à l'appel du constructeur de sa superclasse avec les bons paramètres. En voici un exemple :

[56] JavaCC est disponible sur http://www.webgain.com/products/java_cc/

```
public class XRole extends AbstractRole {
   public XRole(AMS ams) {
   super(ams,"XRole","XRole.prg",true,10 );
}
```

Les différents constructeurs de la classe *AbstractRole* mettent en évidence les différentes options possibles :

```
public AbstractRole(AMS ams,String name,String execution)
public AbstractRole(AMS ams,String name,String execution,int
         nbJetons)
public AbstractRole(AMS ams,String name,String execution,boolean
          flag_file)
public AbstractRole(AMS ams,String name,String execution,boolean
          flag_file,int nbJetons)
```

On indique dans ces constructeurs l'AMS utilisé et le nom du rôle. Il faut ensuite spécifier une chaîne de caractères, un booléen et un entier. La chaîne de caractère "*exécution*" permet soit d'indiquer en paramètre une chaîne de caractère correspondant directement le script (dans ce cas, le *flag_file* est omis ou alors fixé à la valeur *false*), soit d'indiquer le fichier contenant le script (le *flag_file* doit être fixé à *true*). Le nombre de jetons permet d'indiquer le nombre de rôles qui peut être exécuté simultanément par plusieurs agents.

VII.6.3. Composition d'un script

Le programme est constitué de deux parties principales : la déclaration des variables et le corps du programme. Ces deux parties peuvent être éventuellement vides mais l'ordre de ces deux parties importe.

```
// Déclaration des variables
   int i ;
   string s,t ;

// Corps du programme
   Tache1;
   i = 15;
      Tache2;
```

Les types que l'on peut utiliser sont l'entier, la chaîne de caractère ou le float. Le nom des variables doit être en minuscules, contrairement aux tâches qui doivent absolument commencer par une majuscule. Voici quelques exemples :

```
int i;
float j=12.5,k;
string s,t="chaîne de caractère";
int u;
```

Une fois les variables déclarées, on se retrouve dans le corps du script. On ne peut alors plus déclarer de variables. Les variables ont une portée globale à tout le script. Il est donc impossible d'écrire un ensemble d'instructions comme :

```
int j;
      j=5;
      { int j=14; }
```

Les instructions autorisées sont :

- l'affectation,

- la boucle for,
- la condition,
- le parallélisme,
- le mot getRoleInformation.

VII.6.3.1. L'affectation

L'affectation se présente sous la forme "*variable = expression* ;"

Exemple :

s = 15 + 2;
*s = s + 2 * 10;*

Contre-exemple :

s = t = 15 ;
s,j = 15,65;

VII.6.3.2. La boucle

La boucle for est la même que dans la plupart des langages :

for (Instruction d'initialisation ; expression d'arrêt ; instruction de fin de tour de boucle) Instructions

Remarque : si instruction est au singulier, c'est que l'on ne peut mettre qu'une seule instruction :

`for(i=0,j=1;i+j<10;i=i+1,j=j+1);` est impossible

On ne dispose pas des instructions *while* et *do…until*. En effet, la boucle for suffit amplement (à quelques modifications près) :

`while (expression) Instruction <=> for (;expression;) Instruction`

VII.6.3.3. La condition

La condition est la même que les autres langages. Attention cependant l'expression de comparaison ne peut être aussi complexe. La structure d'une expression de comparaison est la suivante : " *expression opérateur expression* "

Exemple :

```
if ( 10 < a)
```

Contre exemple :

```
if ( 10 < a < 30 )
if ( ( 10 < a ) && ( a < 30 ) )
```

VII.6.3.4. Le parallélisme

Le parallélisme est relativement facile à mettre en œuvre, la syntaxe est la suivante :

```
{ Tache1 ; | Tache2; | Tache3; }
```

Il est tout à fait possible d'imbriquer les différents blocs de parallélisme.

Le parallélisme n'est pas exécuté par un seul agent. Un agent étant muni d'un seul thread, il ne peut effectuer plusieurs tâches en même temps. Le parallélisme est donc obtenu par délégation, ce qui signifie que certaines tâches vont être déléguées à d'autres agents. Il est

possible à un agent cognitif de créer un agent réactif fils à qui il délègue une tâche temporairement.

VII.6.3.5. GetRoleInformation

Le *getRoleInformation* est une méthode d'*AbstractAgent* qui permet d'accéder à la mémoire de l'agent. Cet accès permet d'obtenir un rôle plus interactif, dépendant davantage de l'état de l'agent. Par exemple, on peut imaginer que le script continue tant que l'agent n'a pas en mémoire une certaine valeur :

```
for (; getRoleInformation("nombreDeVoiture") < 10;)
      CréerUneVoiture;
```

(On suppose évidemment pour cet exemple que lors de la tâche "*CréerUneVoiture()*", la variable "*nombreDeVoiture*" contenue dans la mémoire de l'agent est incrémentée de 1)

Pour l'instant, la valeur de retour de *getRoleInformation()* est un entier. On ne peut donc la mettre que dans une variable de type int. De plus, on peut lire dans la mémoire de l'agent mais on ne peut pas écrire. La méthode *setRoleInformation()* existe mais n'est utilisable que dans le code d'une tâche.

VII.6.4. Analyse d'un rôle

VII.6.4.1. Création de l'arbre

L'analyse syntaxique et lexicale, ainsi que la création de l'arbre correspondant à un script, s'effectue lors de la première instanciation de la classe *Xrole*. Il n'est pas en effet utile de créer plusieurs fois le même arbre pour le même rôle, il est même plutôt conseillé de ne pas le faire pour des raisons évidentes de performances.

Dans le constructeur *d'AbstractRole* (dont hérite *XRole*), on appelle la méthode construc (qui ne s'effectue qu'une seule fois par rôle comme on vient de le souligner) qui initialise le rôle. Celle-ci lance l'analyseur lexical et syntaxique. C'est à ce moment qu'une erreur de syntaxe est générée s'il y a une erreur dans le script. La construction de l'arbre est construite en même temps que cette analyse.

VII.6.4.2. Modification de l'arbre

A la suite de cette construction, on modifie légèrement l'arbre. La modification de l'arbre concerne la boucle for. On transforme la séquence

`for(T1;T2;T3)T4;` *(Instruction A)*

en

`T1;for(;T2;T3)T4;` *(Instruction B)*

Le but de cette manipulation est de faciliter l'exécution du rôle, en voici l'explication : On utilise une pile d'exécution pour exécuter un rôle (en fait un arbre d'exécution car il y a du parallélisme mais on va occulter cette précision pour le moment dans la mesure où elle n'intervient pas ici). Lorsque dans la pile d'exécution, on rencontre l'instruction A voici ce que l'on devrait faire :

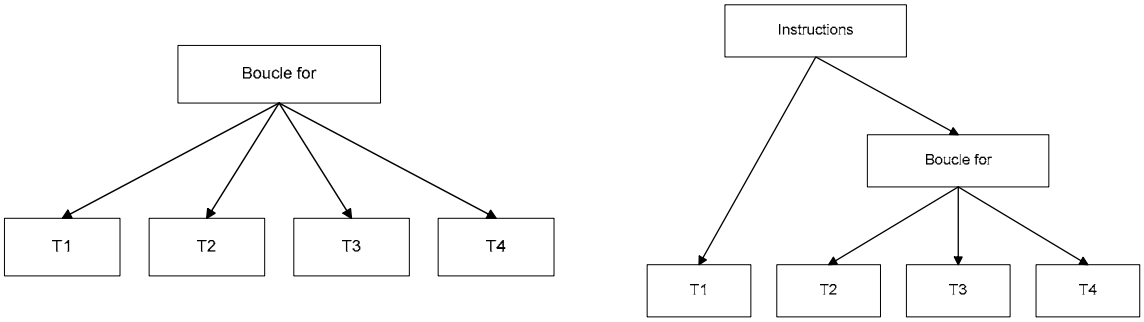
- initialiser la boucle en exécutant T1 (seulement si c'est la première fois que l'on passe sur la boucle)
- vérifier que la condition T2 est vraie.

- si c'est le cas : on empile Instruction A puis T3 puis T4. En effet, on doit d'abord exécuter T4 puis l'instruction de fin de boucle T3 puis revenir à la boucle. Si ce n'est pas le cas : on n'empile rien
- on continue l'exécution en retirant l'élément supérieur de la pile.

La difficulté concerne la condition "seulement si c'est la première fois que l'on passe sur la boucle". En effet, rien de peut distinguer le premier empilement de l'instruction A des empilements suivants. La première solution consisterait à mettre un drapeau pour savoir si c'est ou non le premier passage. Mais ce drapeau serait situé sur l'arbre lui-même alors que celui-ci ne doit pas être modifié par une exécution (un même rôle peut être exécuté simultanément par plusieurs agents).

La solution est donc de modifier l'arbre dès sa création : on transforme chaque boucle en deux instructions. L'algorithme général est le même que précédemment sauf que l'instruction T1 est exécutée comme une instruction qui serait placée avant la boucle.

Cette modification est relativement simple et permet de contourner la difficulté. De plus, cela ne nécessite qu'un simple parcours de l'arbre et une sorte de rotation gauche pour chaque boucle qui contient l'instruction T1. On passe donc du schéma 1 au schéma 2 :



VII.6.4.3. Liste des tâches

Lors du parcours de l'arbre (cf. l'opération décrite ci-dessus effectuée lors de l'initialisation du rôle), on liste les tâches incluses dans le rôle. A partir de cette liste de tâches, on peut calculer la liste des compétences (pour chaque tâche, l'agent doit posséder une ou plusieurs compétences). A partir de cette liste de compétences, on peut savoir si un agent peut ou non exécuter le rôle (un agent doit posséder toutes les compétences pour pouvoir tenir un rôle).

Imaginons qu'un agent ne sache pas faire une certaine tâche T incluse dans un rôle R mais qu'il soit capable d'exécuter toutes les autres tâches. La vérification qui stipule qu'un agent doit posséder les compétences de toutes les tâches n'est donc pas validée. En revanche, celle qui consiste à vérifier qu'un agent possède une compétence pour exécuter une tâche est bien effectuée. Pour chaque compétence manquante pour l'exécution de la tâche, l'agent devra préalablement s'assurer qu'il peut soit sous-traiter la tâche ou acheter la compétence requise. Il effectuera donc cette vérification et relancera la lecture du script à nouveau.

La liste des tâches calculée au moment de la création de l'arbre sert également pour calculer le coût d'un rôle. En effet, le coût d'un rôle est égal à la somme des coûts de l'exécution des compétences nécessaires pour chaque tâche. Cette liste est créée et stockée dès la création de l'arbre car elle est très utilisée.

VII.6.5. Interprétation du rôle

VII.6.5.1. Initialisation

Lorsqu'un agent prend un rôle et qu'il décide de l'exécuter (la méthode *doARole()* de la classe *AbtractAgent*), on exécute la méthode *run()* de *AbtractRole*. Celle-ci commence par les initialisations du rôle :

- une nouvelle pile d'exécution est créée. Celle-ci contient un lien vers la racine de l'arbre (le début de l'exécution du rôle)
- un tableau est initialisé.
- d'autres variables sont également initialisées : la tache en cours, si le rôle est toujours actif...

Le tableau permet de stocker les tâches qui ont été déléguées et qui ne sont pas encore terminées. Il faut conserver cette liste car une tâche non terminée peut bloquer l'exécution d'un script. Par exemple, si l'on a T1;T2; et que l'on délègue T1, il faut attendre la fin de l'exécution de cette tâche pour pouvoir continuer le rôle. Par contre, si l'on a {T1;|T2;} (T1 et T2 peuvent être exécutées en parallèle), on n'attend pas la fin de la tâche T1 que l'on a déléguée pour faire la tache T2. Il faut donc conserver en permanence la liste des tâches en cours que l'on met à jour en consultant les messages reçus par l'agent en train d'exécuter le rôle.

VII.6.5.2. Espace des variables

Il s'agit d'un tableau contenant les variables utilisées par le rôle. Ce tableau contient des objets, les variables peuvent être de type différent. Cependant, seuls les types simples sont gérés (flottant, entier, chaîne de caractères). Ces variables sont utilisées pour les boucles, les tests, les opérations ainsi que pour stocker des valeurs que l'on extrait de la mémoire de l'agent par l'intermédiaire de la fonction *getRoleInformation(nom de la variable)*. Cette méthode est une méthode de la classe Agent mais elle est également reconnue par l'interpréteur de script. Cela permet par exemple d'exécuter certaines tâches déterminées en fonction de l'état de l'agent.

VII.6.5.3. Exécution du rôle

L'exécution du rôle est une grande boucle qui s’exécute tant qu'il y a des instructions, tant que l'agent est vivant (car il peut très bien mourir en plein milieu d'un rôle) et tant que la dernière tâche exécutée n'a pas échoué.

Si l'une de ses conditions n'est pas vérifiée, l'exécution du rôle est arrêtée. Dans le cas où la dernière tâche a échoué, on déclare que le rôle a lui aussi échoué. En revanche, s'il n'y a plus d'instructions à exécuter, le rôle est considéré comme effectué et l'agent reçoit alors une certaine quantité d'énergie.

Au sein de la boucle, différentes opérations se succèdent :

- on commence par mettre à jour la liste des tâches en cours qui ont été déléguées. Pour cela, on consulte les messages de l'agent et on met à jour cette liste, i.e. on enlève les tâches terminées
- on rafraîchit le rôle, c'est à dire que l'on exécute le rôle jusqu'à être bloqué (voir plus bas pour plus de précisions). Ce qui met à jour également la liste des tâches en cours en ajoutant de nouvelles tâches

- on parcourt les tâches contenues dans cette liste et pour chaque tâche, on essaie de la déléguer.
- on prend soin de garder une tâche que l'agent exécutera lui-même (en effet, la délégation n'est pas intéressante pour un agent car elle l'oblige à partager les gains avec l'agent à qui il délègue)

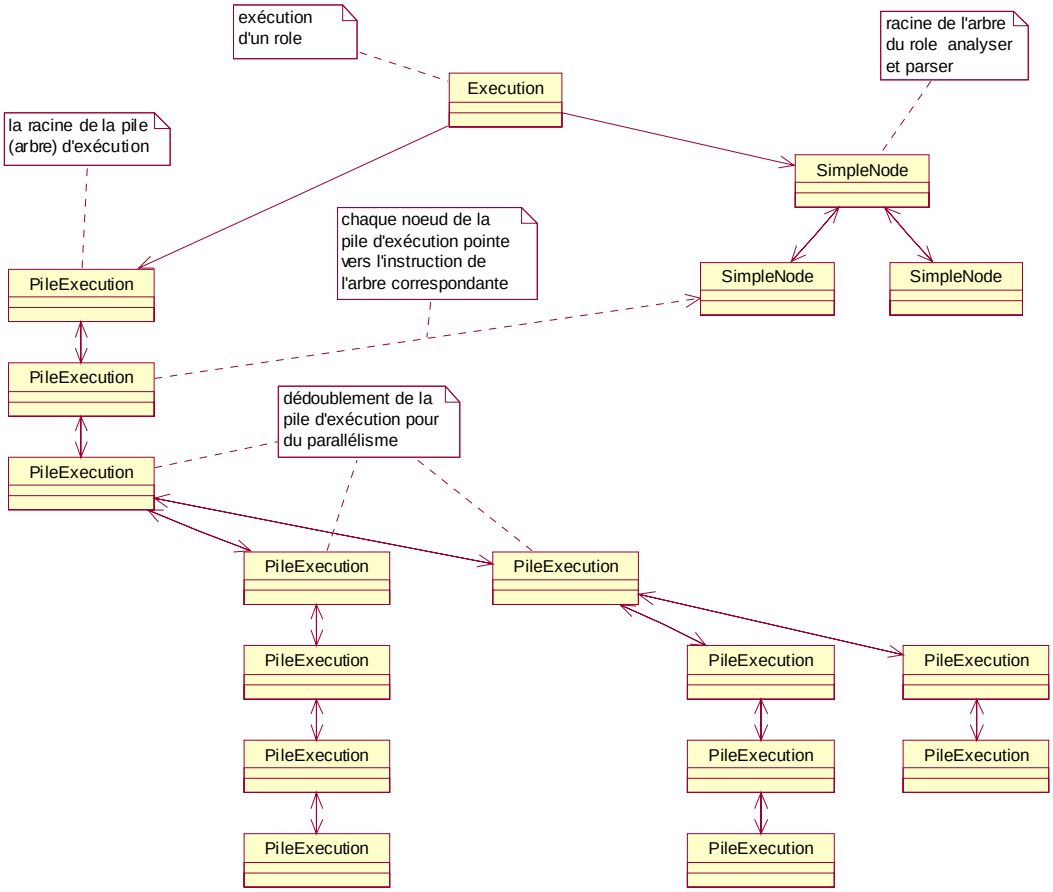


Diagramme d'instanciation

Ce diagramme représente les classes instanciées lors de l'exécution d'un rôle. Les instances de *SimpleNode* ne sont créées qu'une seule fois lors de la création du rôle par la plate-forme alors que l'instance *Exécution* est créée lors de l'exécution du rôle par un agent. La pile d'exécution est créée au fur et à mesure de l'exécution d'un rôle. Chaque élément de la pile d'exécution contient une référence vers un nœud de l'arbre auquel il se rapporte.

VII.6.5.4. Pile d'exécution

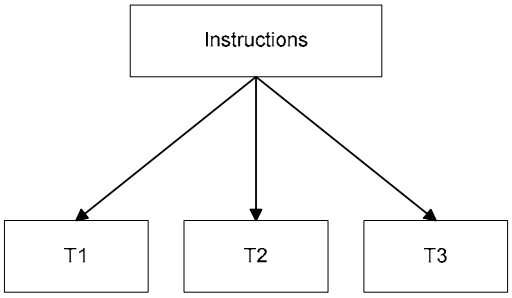
La pile d'exécution permet de suivre l'exécution du rôle : connaître les prochaines instructions à exécuter et l'ordre de cette exécution. Une pile simple n'est pas très compliquée : elle dispose de deux méthodes principales : empiler et dépiler.

Ici et du fait de l'implémentation du parallélisme, la pile d'exécution propose d'autres méthodes : empiler et dépiler un frère, il s'agit de créer (et de détruire) les différentes branches créées par le parallélisme.

VII.6.5.5. Exécution des instructions

L'exécution des instructions du rôle est relativement simple : on dépile l'instruction placée au sommet de la pile et l'on exécute l'instruction. Cette exécution peut induire l'empilement d'une ou plusieurs instructions (comme on l'a vu plus haut avec la boucle for). Ce dépilement est opéré tant que la pile n'est pas vide. Quand celle-ci est vide, c'est que le rôle est terminé. Voyons les différentes instructions que l'on peut rencontrer et les opérations qui sont effectuées :

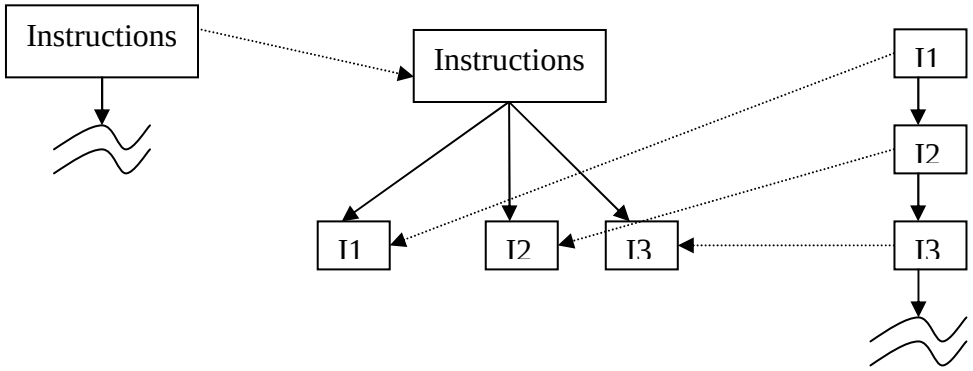
❶ Un groupe d'instruction : *{ I1; I2; I3; }*
Cela signifie que le haut de la pile contient un lien vers un nœud de l'arbre de ce style :



Les trois instructions I1, I2 et I3 (qui peuvent elles même être composées d'autres instructions) sont donc empilées (après avoir dépilée la pile) dans l'ordre I3, I2 puis I1 (il s'agit ici d'une exécution séquentielle où l'ordre est important).

Pile d'exécution avant l'exécution

Pile d'exécution après l'exécution

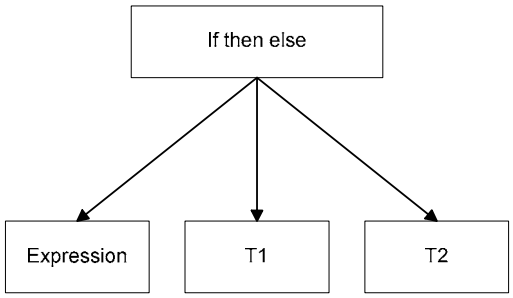


❷ Un test

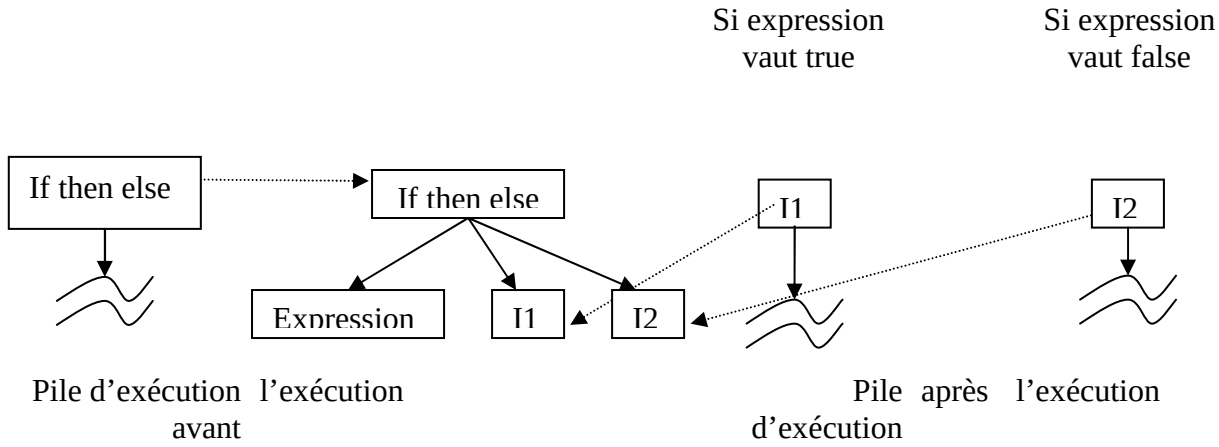
```
if (expression) I1; else I2;
```

`else I2;` est facultatif

La pile pointe donc vers ce nœud :



On dépile (cela signifie que l'on récupère le lien vers le nœud de l'arbre). On évalue l'expression (qui est le premier fils du nœud dépilé), puis suivant le résultat de cette expression, on rempile le deuxième ou le troisième (s'il existe) fils afin qu'il soit à son tour exécuté.



❸ Une tache

Dans le cas d'une tâche, on met à jour la liste des tâches en cours. Cette liste contient les tâches non commencées, les tâches commencées et certaines tâches terminées qui n'ont pas été enlevées (cette opération est effectuée lors d'un nettoyage). Un marqueur permet de distinguer l'état des tâches. Chaque fois qu'on rencontre une tâche dans la pile d'exécution, on la rajoute dans cette liste.

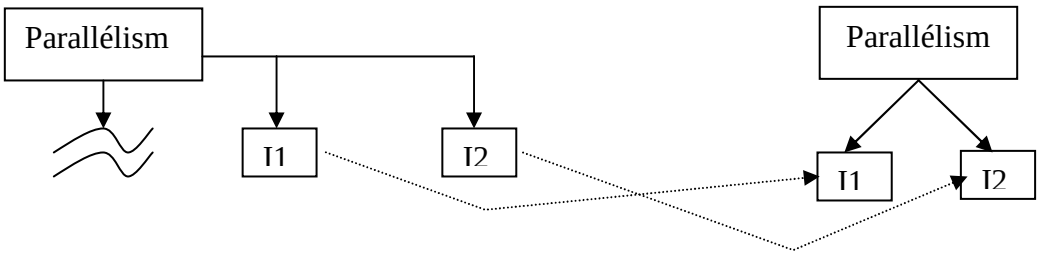
Lorsque l'on rencontre une tâche, on ne dépile pas réellement la pile. En effet, la tâche n'est pas encore exécutée. On met donc un autre marqueur au sommet de la pile afin de savoir que la tâche a été traitée mais n'a pas encore été exécutée. C'est lorsque la tâche sera finie que l'on dépilera réellement la pile d'exécution. Cela a pour but de bloquer l'exécution (on verra qu'elle ne bloque que la pile d'exécution mais pas l'arbre d'exécution dans sa totalité, très important dans le cas du parallélisme).

Chaque fois que l'opération dépilement-exécution est fructueuse (c'est à dire que l'on peut réellement dépiler quelque chose), on la réitère. C'est ce que nous appelons l'opération de rafraîchissement de la tâche (*while (pile.refresh()>0);*). Une fois que cette boucle est effectuée, la liste des nouvelles tâches que l'on peut effectuer est mise à jour.

On aura pris soin, avant d'exécuter cette boucle, de consulter les messages de l'agent afin de mettre à jour la liste des tâches en cours (ce qui a pour but de débloquer le rôle puisqu'une tâche terminée provoque un dépilement dans la pile ce qui permet d'exécuter les instructions suivantes)

VII.6.5.6. Parallélisme

Lorsque l'on rencontre le bloc *{ I1; | I2; }*, il s'agit de parallélisme. Pour cela, on dédouble la pile d'exécution et chaque branche correspond aux différentes instructions (ici, on aurait une branche pour I1 et une autre pour I2). La structure de la pile d'exécution a alors cette forme :



Lorsque l'on exécute la pile (c'est à dire que l'on effectue l'opération dépilement-exécution), on regarde non pas le sommet de la pile mais les sommets des piles. L'ordre n'est pas important, cependant il est actuellement exécuté de gauche à droite, le premier élément de la liste puis le deuxième.

Dans le diagramme ci-dessous, on exécute un rôle simple qui consiste à exécuter séquentiellement trois tâches Tache1, Tache2 et Tache3. La délégation n'est pas mise en œuvre. L'exécution correspond à celle qui était utilisée avant l'analyseur-interpréteur. Ce genre de script ne nécessitait que la décomposition de chaînes de caractères.

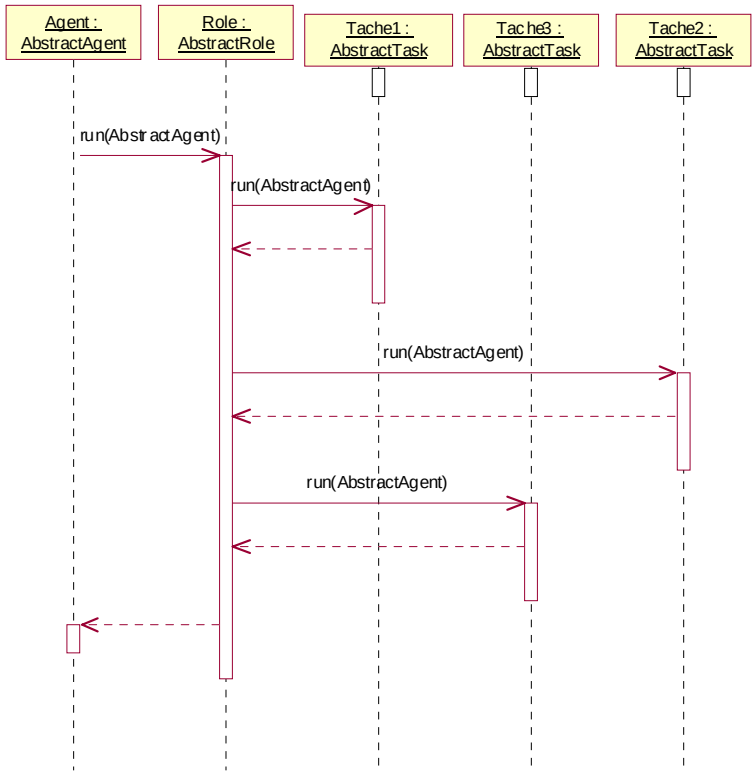


diagramme de séquence - exécution séquentielle de tâches

En revanche, lors d'une exécution en parallèle, un agent peut-être amené à déléguer certaines tâches (mais il n'est pas obligé). Dans le diagramme suivant, les tâches Tâche1 et Tâche2 sont effectuées simultanément par deux agents différents. La troisième tâche Tâche3 est effectuée par l'agent qui exécute le rôle (l'agent Agent1). Ce diagramme montre également les messages utilisés pour la délégation.

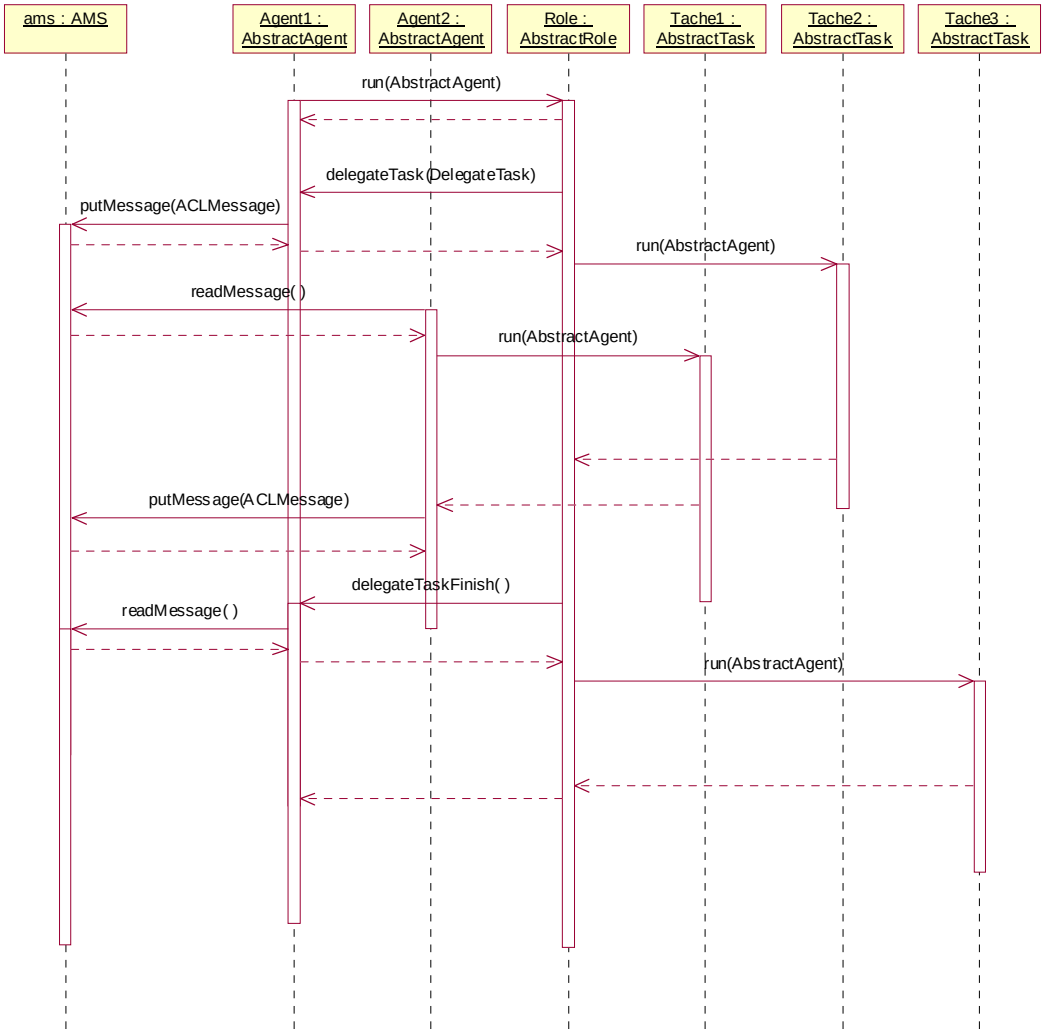


Diagramme séquentiel – exécution parallèle par délégation

L'un des problèmes actuels de cette version de l'interpréteur est que la compatibilité des types n'est pas vérifiée. Par exemple la somme entre une chaîne de caractères et un entier n'est pas possible et doit lever une erreur d'exception. Il serait intéressant aussi de développer un éditeur de script qui permettrait au programmeur phoenix de valider son script avant de l'exécuter. Aujourd'hui, c'est seulement une fois la plate-forme lancée et le rôle lu, que le système détecte une éventuelle erreur.

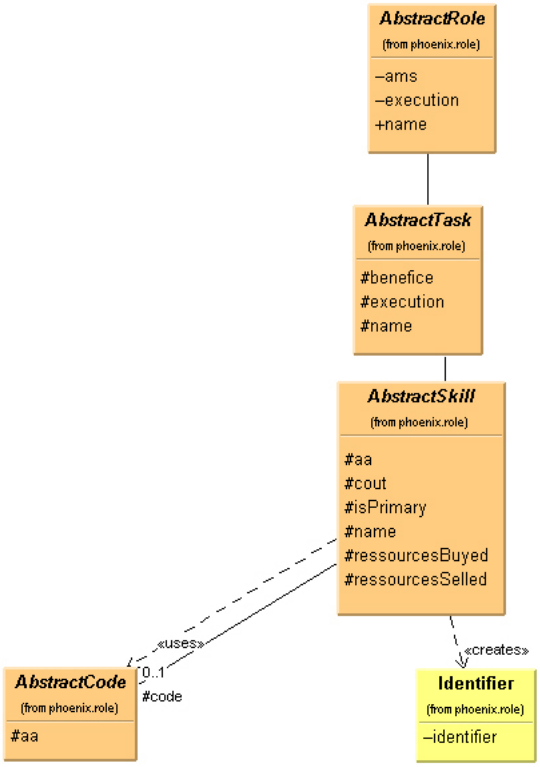
VII.7. Les compétences

VII.7.1. Modélisation et utilisation

Une compétence est une connaissance élémentaire (ou atomique). La connaissance de certaines compétences permet à l'agent de déterminer quels rôles il peut tenir et quelles tâches

il peut exécuter. Certaines compétences dites cognitives peuvent être apprises, interprétées et améliorées tandis que des compétences purement réactives ne peuvent être modifiées. L'apprentissage de certaines compétences peut demander la connaissance de compétences pré-requises (tout comme par exemple l'apprentissage de la physique demande la connaissance des mathématiques).

Une compétence coûte (en général) de l'énergie pour s'exécuter et peut demander zéro ou plusieurs ressources pour s'exécuter. Les ressources consommées seront effacées de l'environnement.



Exemple d'implémentation du modèle Rôle - Tâche – Compétence

Nous avons souhaité reprendre le concept de compétences systèmes et des compétences métiers propres au EJB (chapitre Choix Technologiques). Nous proposons donc de représenter les compétences sur deux niveaux ou deux couches. Cependant nous n'avons pas souhaité imposer cette représentation aux développeurs d'applications SMA avec Phoenix.

VII.7.2. L'acquisition de compétences

La philosophie de Phœnix est de démarrer la construction d'une application à partir d'agents éventuellement vides (i.e. incompétents) et de leur adjoindre dynamiquement les compétences nécessaires à la réalisation de certaines tâches et à la tenue d'un rôle.

Ces classes de compétences peuvent être réactives ou cognitives. Des compétences réactives peuvent être associées à un agent réactif ou cognitif alors que des classes cognitives ne peuvent être rattachées qu'à un agent cognitif.

Tous les agents ne possédant pas toutes les compétences, il s'avère parfois qu'un agent veuille exécuter une tâche pour laquelle il ne possède pas la compétence nécessaire. Il y alors

deux alternatives : l'agent peut déléguer la tâche ou il peut décider d'acheter cette compétence auprès d'un autre agent.

VII.7.3. La recherche de compétence : un problème sémantique ?

Lorsqu'un agent recherche une compétence sur la plate-forme sur un réseau distant, il recherche un nom de fichier (et une signature authentifiant l'unicité de ce fichier). Le rôle décrit l'ordonnancement des tâches et les identifie clairement également par leur nom. Chaque tâche spécifie quelle compétence utiliser en la nommant explicitement. La tâche T1 peut par exemple nécessiter la compétence S1 et la compétence S2. En spécifiant explicitement le nom de la compétence à posséder, nous évacuons le problème de la sémantique qui consisterait à dire « je recherche une compétence qui calcule une transformée de Fourier ». Une telle requête nous retournerait plusieurs compétences possibles ayant des noms différents et nous ne saurions pas exactement quelle méthode de cette compétence appeler pour effectuer notre calcul.

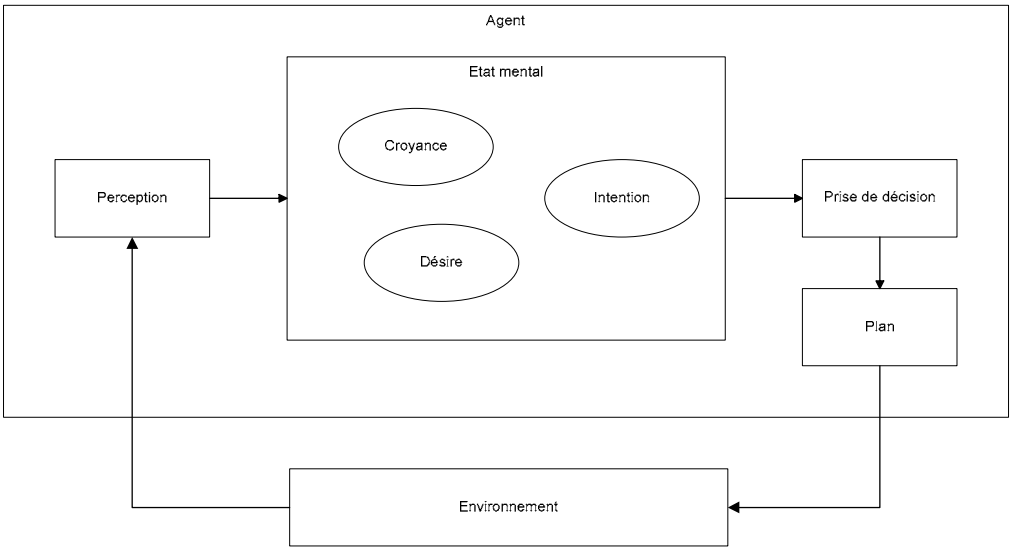
Nous pouvons ajouter dans l'entête de la compétence des Meta-Données qui, telle une page web, peut contenir des champs avec le Titre, les mots-clés et un descriptif. Cela n'a pour le moment pas été implémenté mais pourrait constituer une évolution future de la plate-forme.

VII.8. L'état mental

L'état mental représente une forme d'intelligence artificielle qui se base sur les croyances de l'agent à propos de son environnement, des autres agents (accointances) et sur lui-même (désirs). Grâce à lui, l'agent cognitif est capable de prendre des décisions ayant un sens. Celles-ci seront fonction du but recherché par l'agent (survivre, gagner le plus d'énergie possible …). L'agent pourvu de cet état mental pourra tirer des leçons sur les décisions qu'il a pu prendre dans le passé.

Il pourra aussi effectuer des actions dont les résultats lui seront bénéfiques à plus ou moins long terme, par exemple il peut garder dans son état mental l'envie de changer de plate-forme, et dès que cela sera possible (en récupérant les compétences requises par exemple) effectuer cette action.

A partir de sa **perception** de l'environnement, l'agent cognitif met à jour ses **certitudes** et **croyances** qui seront à la base de sa décision. Ensuite le processus de décision (que nous modéliserons par la suite) utilisera l'expérience acquise dans le passé pour déterminer une planification lui permettant d'atteindre son but : gagner un maximum d'énergie.



Nous avons fait le choix de ne pas recourir à la logique formelle pour traiter des croyances, comme cela est proposé dans les BDI[57]. Nous utilisons simplement un système de mise à jour de certaines variables. Ce système est relativement proche des règles d'un système expert.

VII.8.1. Les mémoires impliquées dans l'état mental

Chaque agent dispose en fait de plusieurs mémoires distinctes possédant chacune des particularités. Il y a deux types de croyances. Le premier type est induit par la persistance des informations dans la mémoire : par exemple, imaginons que l'agent a déjà emprunté une route menant d'un point à un autre mais que cette route ait été fermée depuis. L'agent se rappelant de cette voie va l'emprunter alors qu'elle n'aboutira pas. Le second type de croyance est induit par l'utilisation de mémoires subjectives : si un agent ne dispose pas d'une information dans une mémoire subjective, cette mémoire va tenter de fournir une réponse à la requête en utilisant par exemple un mini système expert ou un réseau de neurones. Nous n'imposons pas de système décisionnel en particulier. L'utilisateur de Phoenix pourra utiliser les classes qu'il souhaite selon le type d'application à développer. Nous mettons uniquement à sa disposition des packages JAVA de réseaux de neurones ou de règles décisionnelles. Ces packages sont disponibles en Open Source.

VII.8.1.1. La mémoire de rôle

C'est une mémoire sure, de type associatif (*java.util.Hashtable*) (à un nom de variable on associe un objet), elle permet au rôle de l'agent de stocker des valeurs, aucune interprétation n'est faite sur les valeurs, la demande d'une variable absente de la mémoire provoque une erreur d'exception. Les valeurs sont valides toute la durée du rôle, et sont effacées en fin de rôle (il existe cependant une option « static » permettant de rendre une variable inamovible).

VII.8.1.2. La mémoire de l'accointance

Cette mémoire rassemble des informations sur les agents avec lesquels l'agent a déjà travaillé (dans le cas de ventes ou de délégations). Plus que mémorisés, les agents sont évalués suivant un coefficient de satisfaction, modifié chaque fois qu'une transaction a lieu

[57] Believes, Desires, Intensions

avec cet agent : par exemple un agent vendant une information déjà connue sera mal considéré, tandis qu'un agent réalisant une tâche déléguée très rapidement sera bien noté. Dans tous les cas, les critères de satisfactions sont soumis aux effets de l'état mental de l'agent. Cette mémoire de l'accointance va entrer en ligne de compte lors de la recherche de partenaires : un agent va d'abord chercher à travailler de nouveau avec un agent qui l'a satisfait par le passé. Il préférera travailler plutôt avec un inconnu que de traiter à nouveau avec un agent lent ou inefficace. Techniquement ces comportements sont codés sous formes de règles (SI…ALORS…SINON) pour une question de rapidité d'exécution mais il est aussi possible d'utiliser un réseau de neurones (plus lent et plus difficile à stabiliser).

VII.8.1.3. La mémoire des croyances

Les croyances peuvent porter sur différents objets de la plate-forme mais la mémoire des croyances rassemble pour le moment des informations concernant l'environnement et les prix des ressources. Nous avons implémenté des méthodes dans la classe AMS qui permettent à l'agent qui les invoque de recevoir des informations comme le contexte d'exécution. A titre d'exemple :

- Une méthode comme *GetNbAgents*() retourne le nombre d'agents présents sur la plate-forme à un instant donné.
- Une méthode comme *GetAvgNbAgents*() retourne le nombre d'agents présents en moyenne sur la plate-forme depuis son lancement. Cela permet d'avoir une idée sur le taux d'occupation.
- La méthode *GetLaunchTime*() retourne le nombre de seconde écoulées depuis le démarrage de la plate-forme.
- La méthode *GetNbMigrationAgents*(X, Y) retourne le nombre d'agents qui ont migré entre deux dates.

Un agent peut utiliser ce genre de méthodes pour actualiser ses croyances sur la plate-forme. Si elle redémarre souvent et si le nombre de migration d'agents est beaucoup plus élevé que sur une autre plate-forme, il pourra en déduire que la plate-forme n'est sans doute pas fiable et qu'il vaut mieux migrer vers une autre plate-forme d'accueil. Cela se traduit par des règles du type :

IF ((P1.GetAvgNbAgents() – P2.GetAvgNbAgents() > 500) AND P1.GetLaunchTime() < 100.000) THEN COEFF_RELIABILITY = 30.

En ce qui concerne l'environnement, l'agent ne va mémoriser que certaines informations car sa mémoire est limitée. Ping L'interrogation de cette mémoire peut donc produire des réponses qui ne sont pas toujours vraies : elles peuvent d'une part devenir fausses avec le temps (l'environnement a changé depuis que l'agent l'a vu : croyances fortes). Faire une demande sur une partie de l'environnement inconnue ne provoque pas d'erreur : l'agent va utiliser un petit système expert afin de tenter de faire des assertions sur l'environnement. Le chaînage avant peut être utilisé pour déduire de nouveaux faits (d'où le terme de règle de production).

Durant la phase de recherche du chaînage avant le système infère les faits connus et ceux de l'espace de travail avec les clauses antécédentes des règles pour déterminer quelles règles peuvent être déclenchées. Dans une base de connaissances avec de nombreux faits et règles, la phase de recherche prend énormément de temps, donc on testera seulement les règles dont les clauses antécédentes se réfèrent à des faits nouvellement ajoutés dans l'espace

de travail. Une fois que la phase de recherche terminée et l'ensemble de conflit produit, on passe à l'étape de résolution de conflit. Cette phase est certainement l'étape la plus importante du chaînage avant. Quand l'ensemble de conflit est vide ou contient une seule règle, le problème est trivial. Cependant, dans beaucoup de cas il y a plus d'une règle à déclencher, Plusieurs algorithmes existent afin d'effectuer ce choix :

- choisir la première de l'ensemble, c'est simple et cela marche dans certain cas
- choisir la règle ayant le plus grand nombre de clauses antécédentes. L'idée ici est de sélectionner la règle la plus spécifique avant de déclencher les règles plus générales qui ont moins d'antécédents.
- choisir la règle qui se réfère à la donnée qui a été changée le plus récemment. Cette méthode demande de conserver en mémoire cette donnée
- choisir une règle au hasard

Une stratégie courante est d'assigner une priorité aux règles qui peuvent être utilisées dans le processus de sélection. L'avantage est que cela réduit de beaucoup le nombre de règle à rechercher et tester. Nous avons utilisé une version de ce système avec une base de règles très réduite. Il est cependant possible d'ajouter de nouvelles règles et de raffiner la base de faits, qui reste très grossière, mais les mécanismes de base du système de croyance sont présents.

VII.8.2. Apprentissage

L'apprentissage est un vaste domaine de recherche. Nous entendons par apprentissage, l'actualisation des connaissances sur le monde environnant [Beaune, 2001].

Une forme simple d'apprentissage est d'actualiser le niveau de confiance qu'un agent accorde à un autre agent. Cette notion de confiance intervient lors de l'exécution d'un rôle. En effet, suivant le résultat issu du déroulement du rôle, l'agent peut modifier son expérience. Par exemple, un agent A1 qui délègue une tâche T1 un agent A2, va actualiser son expérience en fonction du retour de l'agent A2. Si l'agent A2 lui retourne le résultat, la confiance de A1 envers A2 va augmenter. En revanche, si l'agent A2 n'a pas assumé sa mission, la confiance de A1 va baisser.

Une autre notion d'apprentissage concerne le raffinement de ses connaissances sur un domaine particulier. Cet apprentissage peut être supervisé ou non. Par exemple un agent peut apprendre à reconnaître des formes, auquel cas il faudra peut être que le développeur prenne le temps de supervisé un réseau de neurones.

De plus, l'exécution de son rôle aura un impact sur l'environnement et donc indirectement sur ses croyances. On appellera certitude une croyance coefficientée à 100%. L'apprentissage d'un agent cognitif nécessite des compétences particulières (skills). Ces skills peuvent concerner :

- Apprentissage par réseaux de neurones
- Apprentissage par Q-Learning
- Apprentissage par système expert
- Apprentissage par réseaux bayésiens

Nous avons implémenté pour le moment les apprentissages par réseaux de neurones et par système expert dans la partie « Décisionnel ». Les autres formes d'apprentissage seront implémentées par la suite. Nous n'apportons rien d'innovant dans ces algorithmes

d'apprentissage. Nous proposons simplement au développeur un ensemble de classes issues du monde Open Source lui permettant d'implémenter plus facilement un tel système. Dans le cadre d'un réseau de neurones, par exemple, nous laissons au développeur le soin de paramétrer les poids de son réseau car ce genre de système d'apprentissage est très dépendant de l'application envisagée. Nous n'en parlerons donc pas davantage.

VII.9. Fonctionnement de la plate-forme

C'est un système autonome composé d'un agent noyau, d'un environnement et d'un système de transport de messages. Dans le système, plusieurs plates-formes doivent pouvoir coopérer et notamment permettre aux agents de chacune des plates-formes de communiquer avec ceux d'une autre. A un niveau de granularité n, la plateforme est un SMA mais elle est un simple agent à un niveau n+1, dans la mesure où les plateformes communiquent entre elles, au même titre que les agents entre eux. Plusieurs plateformes communiquant et coopérant ensemble sur un réseau constituent donc aussi un SMA.

VII.9.1. Agent Noyau

L'agent Noyau (AMS) est un agent particulier disposant de pouvoirs et de compétences spéciales. C'est l'agent qui est chargé du bon fonctionnement d'une plate-forme, c'est-à-dire de l'exécution des rôles. Pour cela, il gère le cycle de vie complet des agents. Il peut leur donner naissance, les stopper ou les faire passer dans l'au-delà. Il sait retrouver tous les agents sur la plate-forme locale et possède des informations sur eux. Il offre des services aux agents comme la transmission des messages, c'est en quelque sorte l'interface entre la plate-forme et les agents.

Il peut modifier l'environnement et surpasser toutes les règles imposées aux autres agents. Il peut par exemple ajouter des ressources, les déplacer, les supprimer etc. Il est l'agent intermédiaire entre le concepteur du système et le SMA. Si un utilisateur décide d'intervenir sur une simulation, pour ajouter un événement par exemple, il en donne l'ordre à l'agent noyau qui réalise l'action pour lui.

L'agent noyau est aussi responsable de la taxation en énergie des agents présents sur la plateforme. Il peut ajuster les taxes locales comme il le souhaite. Il gère toute la circulation de l'énergie, sa création, sa suppression etc. Il peut aussi proposer un prix de départ pour un service pour stabiliser le système plus rapidement.

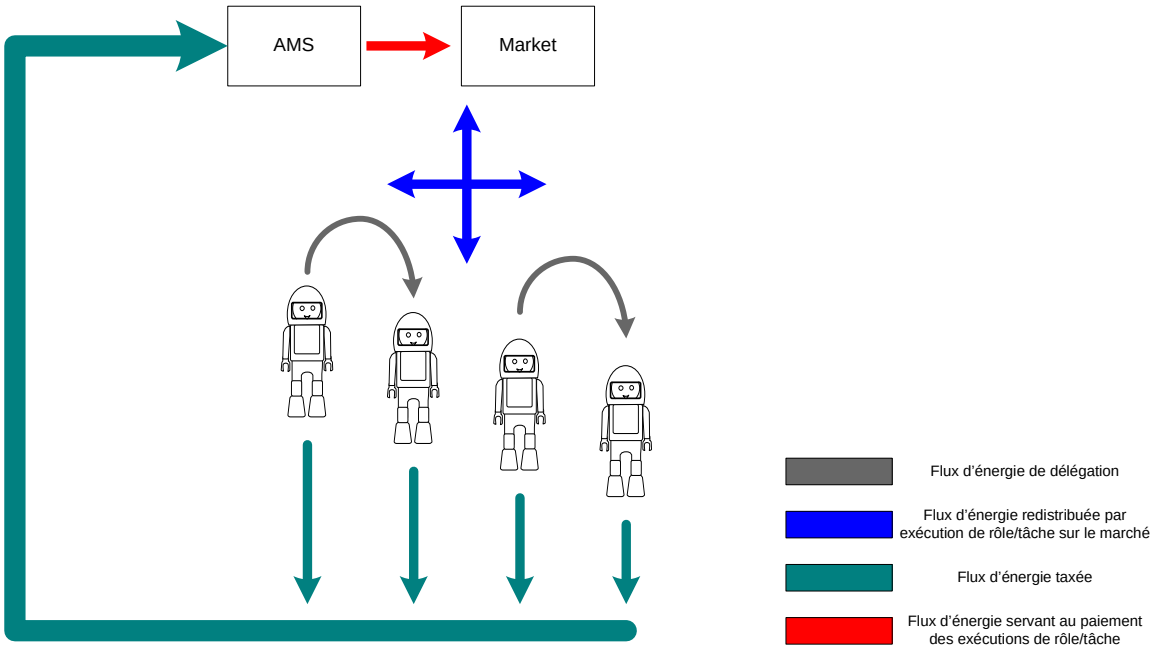
Au démarrage, tous les prix sont identiques et constants dans l'ensemble du système. Les agents vont ensuite modifier les prix de leurs services en fonction de la loi de l'offre et de la demande. L'agent noyau qui taxe les agents de la plate-forme locale pourra actualiser le montant de ces taxes comme il le souhaite.

VII.9.2. Energie

L'énergie circule dans une plate-forme comme un fluide dans des canalisations. Si on fait un parallèle avec l'économie, il s'agit d'une monnaie que les agents (les entreprises et les individus) s'échangent au profit d'un bien ou d'un service. Le but des agents est de maximiser leur capital d'énergie. La seule façon pour un agent de gagner de l'énergie est d'effectuer des rôles proposés par la plate-forme locale ou des plates-formes distantes. Un agent meurt lorsqu'il n'a plus d'énergie.

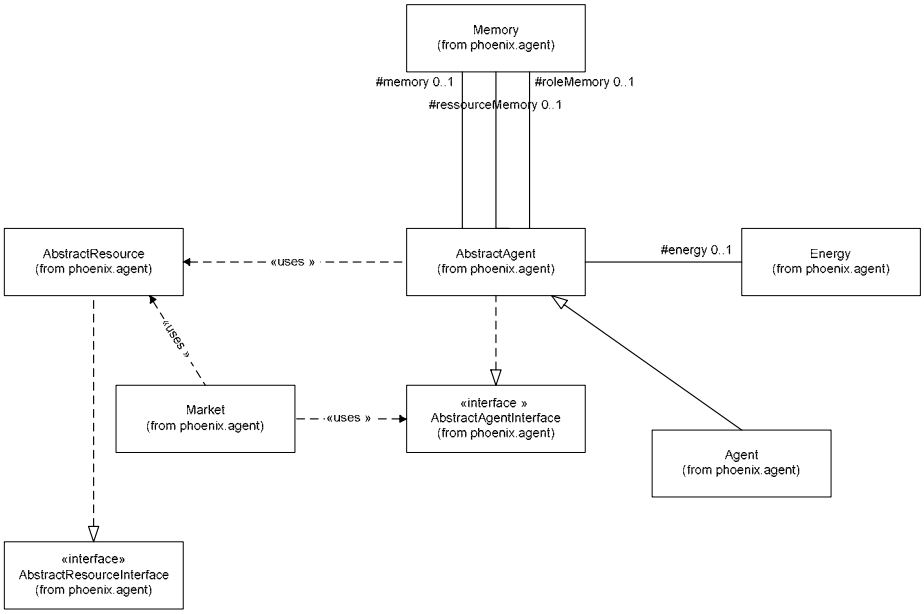
Le principe de l'énergie est efficace dans la mesure où il introduit une contrainte sélective sur les agents : seuls les meilleurs agents sont amenés à survivre et à continuer de

travailler sur la plate-forme. D'autre part, les échanges de ressources et les délégations de tâches étant soumis à la loi de l'offre et de la demande, la répartition des rôles et des tâches est auto-régulée puisque prendre un rôle produisant une ressource surabondante n'est pas rentable pour les agents. Il s'agit en fait d'un système capitaliste idéal: sans considérations éthiques. Ce système d'énergie régit toute la plateforme et maintient l'auto-organisation du système. Techniquement, nous utilisons simplement un système d'actualisation de variables. Chaque agent possède une variable sur la plate-forme qui pourrait s'apparenter à un compte bancaire. La plate-forme joue le rôle de banquier et assure la transaction lorsqu'un agent cède de l'énergie à un autre agent. Nous avons fait le choix de faire gérer les niveaux d'énergie par la plate-forme pour éviter à un développeur d'agent de forcer le niveau d'énergie de son agent artificiellement, ce qui pourrait s'apparenter à un acte de piratage. Cela permet aussi à la plate-forme de savoir précisément la situation énergétique de l'agent et donc de l'imposer en conséquence (il est difficile d'échapper au FISC). S'il migre sur un nouvel hôte, il transfère aussi son compte (comme si on émigrait dans un autre pays). A noter qu'un agent ne peut avoir un autre compte sur une plate-forme suisse ☺. D'une manière générale, un agent n'a qu'un seul compte exploitable simultanément.

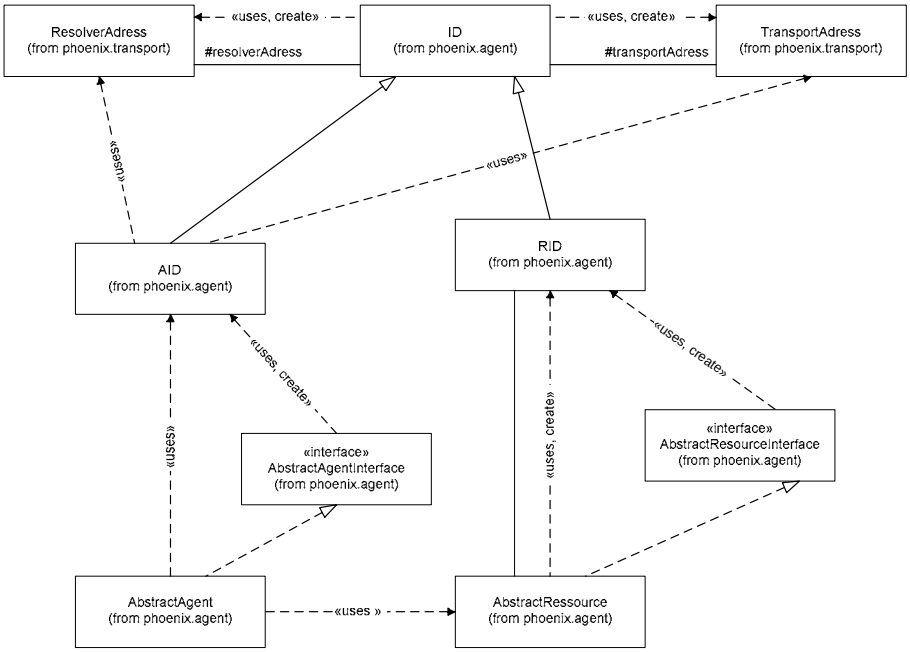


VII.9.3. Ressource

Une ressource est toute entité de la plate-forme qui n'est pas un agent. Elles peuvent être achetées, vendues et créées. Elles peuvent être détenues par les agents ou par l'environnement.



Dans un environnement, les ressources ne sont jamais inépuisables. Phœnix possède donc des classes génériques permettant de manipuler des ressources comme de l'eau, de la nourriture, des temps processeurs etc. Ces classes sont basées sur un principe de compteur et gèrent un accès simultané par plusieurs agents. Des agents peuvent entrer en concurrence pour utiliser des ressources. En outre pour les objets partagés sur un réseau, nous utilisons directement des systèmes à objets distribués (CORBA). Chaque ressource doit être identifiée formellement. La figure ci-dessous montre le système d'identification des agents et des ressources :



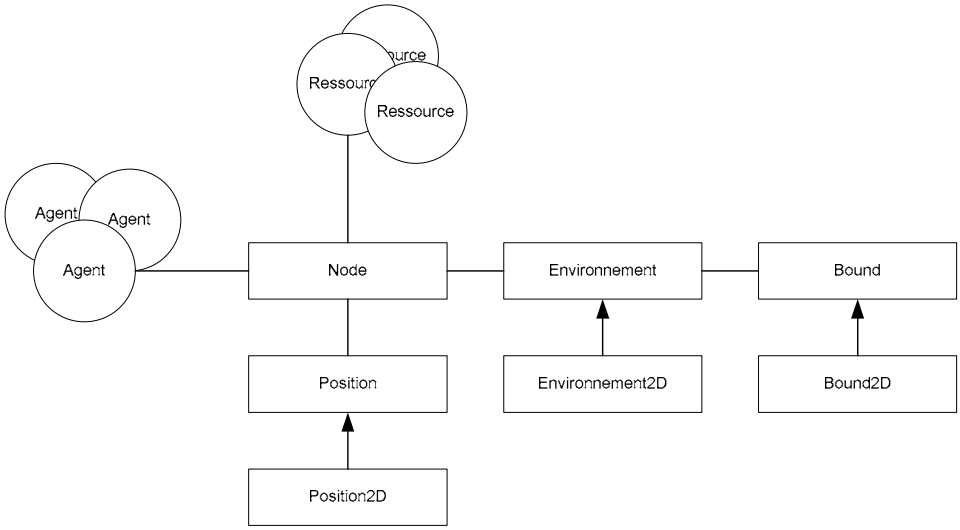
VII.9.4. Environnement

C'est un espace dans lequel évolue un ou des agents, une sorte de vaste zone d'échange. Cet environnement peut être implicite ou explicite, c'est-à-dire que le concepteur du SMA peut l'instancier ou non. Dans le cas d'un environnement physique et géométrique, il peut être en deux dimensions ou en *n* dimensions. Pour une simulation d'un trafic routier ou

d’un environnement biologique, comme une colonie de fourmis, il est possible d’instancier un environnement en deux dimensions composé de cellules comme un automate cellulaire. Une cellule est reliée aux 8 cellules voisines directes. L’environnement offre la possibilité d’une communication indirecte, ce qui permet de communiquer d’un agent à un autre par l’intermédiaire d’un autre agent ou d’un autre objet. Ce type de classes permet de simuler des colonies d’insectes.

Dans certains cas, lorsqu’il s’agit d’étudier plutôt la construction de galeries d’insectes sociaux, il est préférable d’utiliser un environnement en trois dimensions. Dans ce cas, une cellule possède 23 voisins. Les agents utilisent ces cellules comme parcours élémentaire pour se déplacer d’un endroit à un autre dans l’environnement. L’environnement se réduit à une carte liant des cellules adjacentes.

Les types d’environnements sont en réalité très hétérogènes. On trouve toutes sortes de structures, les tableaux pour une carte plane, les graphes pour un réseau. Pour rester indépendant de la structure interne des différentes sortes d’environnement que l’on peut trouver, on ajoute une surcouche au dessus. Ainsi, tous les environnements exposent une structure commune comme vue principale. Tous les environnements étendent une interface commune qui les force à respecter une certaine structure. On utilise une Hashtable comme précisé plus haut.



L'environnement implémenté dans Phoenix est en fait une classe abstraite qui offre les services de base pour pouvoir écrire un environnement. En effet, écrire un environnement non abstrait aurait été bien trop restrictif au regard de la diversité des problèmes devant être traités par Phoenix. Cependant, afin de ne pas laisser les utilisateurs de Phoenix sans aide lorsqu'ils auront à écrire leurs environnements, tous les services génériques ont été implémentés. Toutes les requêtes à l'environnement sont faites par message, la classe nommée phoenix.environnement.AbstractEnvironnement offre donc toutes les primitives de traitement de message, ainsi que des méthodes statiques de créations de messages correctement formatés pour créer des requêtes. L'environnement gère aussi toutes les requêtes concernant les déplacements d'agents ou de ressources, l'exploration du voisinage par un agent, le changement des contraintes de cellules, ou la recherche d'agents et/ou de ressources dans le voisinage. D'un point de vue plus interne, l'environnement est représenté par un graphe de cellules, ce qui permet en particulier d'employer les algorithmes classiques de parcours en profondeur, en largeur, de recherche de chemins, ou de chemin de poids minimal (en rapport avec les coûts de déplacements induits par les contraintes cellulaires). Plusieurs types de contraintes peuvent être imposés aux cellules : en particulier on peut choisir le coût

énergétique demandé aux agents pour y entrer, le nombre d'agents pouvant être simultanément dans une cellule (possiblement aucun, pour représenter des cellules contenant des obstacles ne bloquant pas le champ de vision des agents), le nombre de ressources pouvant être transportées dans la cellule, etc. Il est facile de concevoir un environnement comprenant d'autres types de contraintes cellulaires (en surclassant la classe phoenix.environnement.Rule).

VII.9.5. Réseau d'accointances

Les agents sont amenés à coopérer les uns avec les autres. Ils doivent se rappeler les rapports eus avec les autres agents afin de mieux décider des coopérations futures. Aussi, un agent doit pouvoir stocker une liste d'agents (avec qui il a été en rapport) ainsi qu'un jugement qu'il aura établi sur chacun d'eux suivant le déroulement des actions ayant eu lieu entre eux. Il s'agit d'un carnet d'adresse amélioré. Dans le cas où un agent migre vers un hôte différent, il en informe les agents « fournisseurs » et « sous-traitant » afin qu'ils procèdent à l'actualisation de leurs réseaux d'accointances.

VII.9.6. Le market

Contrairement à l'environnement, le market n'est pas juste une classe abstraite, mais une classe à part entière. La classe Market prends en charge tout ce qui concerne les mises en ventes de ressources entre les agents. Du point de vue des agents, l'environnement peut-être vu comme un autre agent dont le seul service est de mettre en contact les acheteurs et les vendeurs. Le market prend en charge les ventes par type de ressources (ressources anonymes) ou par RID (ressources nommées). Tous les types de ressources sont échangés dans le seul et unique Market de la plate-forme. Le Market est implanté en utilisant plusieurs tables de hachage (java.util.Hashtable), toutes les requêtes sont faites par messages classiques. Sur demande le Market peut tenter de contacter des Markets distants, afin de trouver une ressource non disponible sur le Market local pour un agent local ou distant. L'utilisation de messages pour les requêtes au Market est avantageuse à plusieurs titres : elle permet de présenter l'aspect d'un agent habituel aux autres agents et permet facilement d'implémenter la coopération entre les Markets de différentes plates-formes.

VII.9.7. Le modèle graphique

Les agents sont seuls responsables de leurs affichages graphiques. La représentation d'un agent repose sur l'utilisation d'un JavaBean qui peut être simple ou complexe. Des détails sur le fonctionnement des JavaBeans sont donnés dans le chapitre dédié aux choix technologiques.

VII.9.8. Délégation

La délégation d'une tâche par un agent peut s'appliquer lorsqu'il n'a pas les compétences nécessaires à l'exécution de cette tâche ou qu'il lui manque des ressources pour le faire. Dans ce cas, l'agent va demander à un autre agent d'effectuer cette tâche pour lui. En contre partie, l'agent qui exécutera la tâche sera dédommagé (en énergie) par l'agent demandeur, au même titre qu'une société sous-traite et paie son sous-traitant. Avant de prendre en charge l'exécution d'un rôle complet, l'agent doit s'assurer qu'il sait exécuter toutes les tâches composant ce rôle. Dans le cas où une compétence manquerait pour l'exécution d'une tâche, il doit s'assurer au préalable qu'il peut assurément déléguer cette tâche à un autre agent. Une négociation débute alors entre les deux agents.

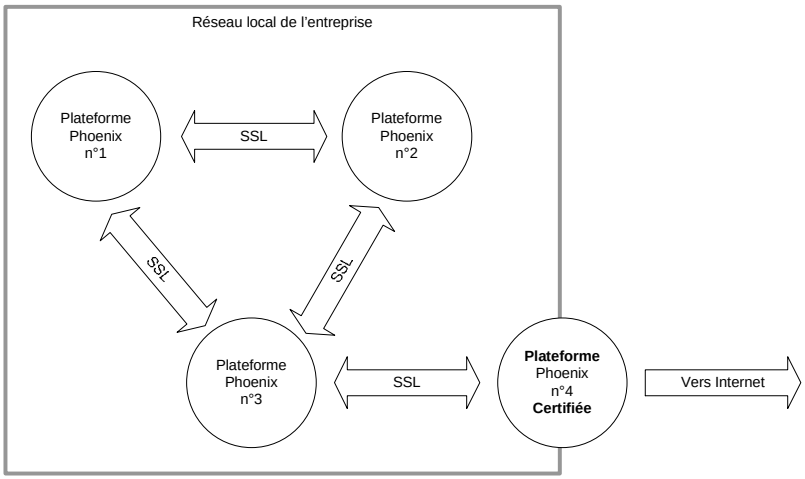
VII.9.9. Un mode de fonctionnement distribué

Contrairement à la première implémentation de Phoenix qui utilisait massivement RMI, nous avons allégé les communications en supprimant les appels à RMI en local. Nous avons migré en CORBA 2.3 et seul le cas de messages envoyés entre deux plates-formes fera appel à la méthode de façon distribuée. En outre, nous avons décidé de séparer l'implémentation de la partie CORBA du reste de la plate-forme. Sous Phœnix, vous trouverez dans le package myCorba, la classe Gestionnaire qui centralise le mécanisme CORBA pour la plate-forme. CORBA devrait évoluer au cours du temps, et si des modifications fondamentales doivent être effectuées, seule, cette classe devrait être à modifier.

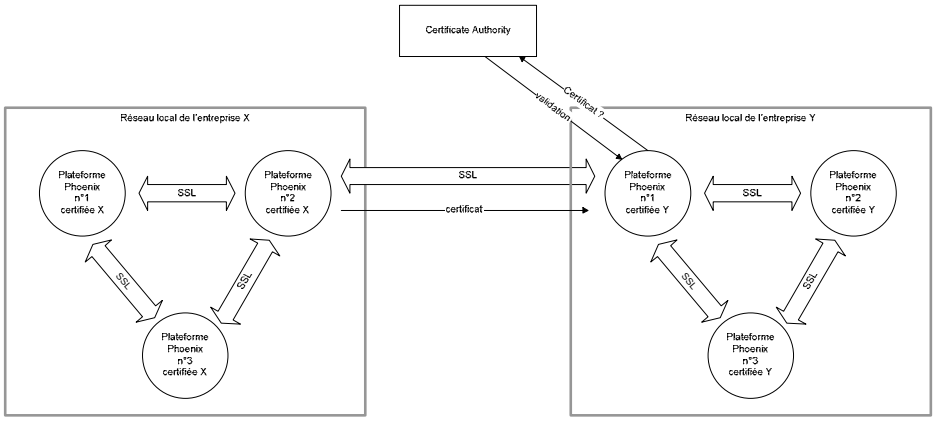
VII.9.10. Sécurisation des transactions

Chaque communication entre plate-forme est cryptée en utilisant l'algorithme RSA de SSL. En ce qui concerne l'authentification, la signature d'un certificat auprès d'un « Certificate Authority » est payante et renouvelable tous les ans. Nous avions plusieurs solutions :

- un certificat unique pour toutes les plates-formes : c'est à dire que lorsqu'une plate-forme PHOENIX cliente voudra valider le certificat d'une plate-forme phœnix serveur quelconque, celle ci sera enregistrée auprès du CA et reconnue en tant que « Plate-forme Phœnix » utilisant le « Certificat Phœnix ». Ceci implique que toutes les entreprises utilisant au moins une plate-forme PHOENIX utilise le même certificat pour se faire valider auprès d'un Certificate Authority. L'avantage est le coût d'achat unique d'un certificat. L'inconvénient est que toutes les entreprises partagent le même certificat ; si une personne mal intentionnée parvient à « casser » le certificat Phœnix et la clé privée, ce sont toutes les entreprises utilisant la plate-forme qui se trouvent menacées.
- Un certificat par plate-forme : chaque plate-forme Phœnix possède son propre certificat, et est enregistrée auprès d'un Certificate Authority. L'avantage est que chaque plate-forme devient unique vis a vis du notaire informatique qui est chargé d'authentifier les certificats. La sécurité en est donc accrue car chaque plate-forme possède sa propre clé privée pour décoder les messages. Chaque plate-forme est capable de fournir un certificat signé et ainsi d'être accédée directement en tant que serveur. L'inconvénient est d'ordre financier car une entreprise utilisant un grand nombre de plates-formes Phœnix est dans l'obligation de faire signer autant de certificats que de plates-formes.
- Un seul certificat par réseau : l'entreprise n'achète qu'un seul certificat et une connexion via Internet nécessitant une authentification sera relayée obligatoirement par la plate-forme possédant ce certificat. L'inconvénient est que toutes les requêtes relayées vont surcharger la plate-forme possédant le certificat. Le fonctionnement est décrit sur la figure suivante :



- Toujours un seul certificat par réseau mais utilisé par toutes les plates-formes Phoenix de l'entreprise. C'est cette solution qui a été retenue. Elle est illustrée sur la figure ci-dessous :



L'implémentation de la sécurité par SSL et certificat rend Phoenix opérationnel pour des applications de commerce électronique et de places de marché.

VII.9.11. Certification d'une compétence ou d'un agent

Lorsqu'un agent désire acquérir une nouvelle compétence ou sous-traiter une tâche à un autre agent, se pose aussi un problème de sécurité. Cette sécurité intervient au niveau de la transaction et au niveau de la certification du code exécutable. En outre un agent « escroc » peut proposer ses compétences et en profiter pour s'introduire dans un système non sécurisé. Comment également faire confiance à un agent ? Va-t-il bien m'envoyer la compétence achetée après lui avoir céder quelques points d'énergie ?

Un agent A1 possède la compétence C1, qui permet de calculer un nombre de Fibonacci en un temps T1 de calcul. Un agent A2 signale à l'agent A1 qu'il possède la compétence C2 qui fait la même chose que C1 mais deux fois plus rapidement car son algorithme est optimisé. A1, souhaitant proposer des services de meilleure qualité tout en optimisant ses ressources, décide d'acheter C2. Comment A1 peut il être certain de la qualité de la compétence qu'il va acheter et être certain que ce n'est pas un virus ou un cheval de Troyes qui va s'exécuter à la première utilisation de la compétence C2 ? Cela peut être aussi un « fake » de l'agent A2 qui avait pour seul objectif de voler quelques points d'énergie à A1.

En ce qui concerne l'agressivité du code machine, le langage JAVA apporte une solution de compromis à cette question. S'il n'est pas compilé à la volé avec un JIT, le code JAVA est interprété par une machine virtuelle qui respecte la politique de sécurité par défaut ou imposée parle propriétaire de la machine.

Il existe aussi la possibilité d'exécuter l'application en sécurisant le noyau du système d'exploitation de la machine d'exécution. Les systèmes Janus [Goldberg and al., 1996] et Subterfuge reposent sur cette approche. Le système intercepte tous les appels systèmes réalisés par l'application avant leur exécution. Ces appels sont analysés pour évaluer leur agressivité.

Une autre approche consiste à intégrer des dispositifs de sécurité dans les fonctions du système d'exploitation. C'est ce que proposent les « Linux Security Modules » qui intègrent des points de vérification avant les sections sensibles des fonctions systèmes. Il s'agit donc d'effectuer des branchements qui déroutent l'exécution du programme vers un module de sécurité que le propriétaire de la machine ajoute au noyau de son système d'exploitation.

Ces systèmes ne sont pas encore totalement maîtrisés (domaine de la R&D) mais présentent des caractéristiques intéressantes.

Il est clair que dans le cadre de la certification des résultats, l'application ne peut pas compter sur les protocoles et la sécurisation des communications. C'est à l'application elle-même ou au système d'apporter une parade à cette possibilité de corruption. Il existe néanmoins deux approches : l'approche système et l'approche application.

- L'approche système repose sur ces techniques comme les exécutions redondantes avec vote et test ponctuel. Une troisième technique, qui repose sur les deux autres, consiste à attribuer une crédibilité à chaque résultat et aux machines fournissant les résultats. Les ressources acquièrent donc de plus en plus de crédibilité au fur et à mesure que le nombre de vérifications ponctuelles validées augmente. De même, les résultats acquièrent une crédibilité de plus en plus grande à mesure que le nombre de résultats concordants pour une même tâche augmente. Cette technique semble tout à fait en phase avec les BDI [Herzip & al., 2000] qui consistent à changer les croyances de l'agent sur son environnement.
- En complément de cette approche système, la certification des résultats peut être effectuée par l'application après réception du résultat. Dans le cas où le code source de l'application n'est pas ouvert, le cryptage des résultats permet de résoudre le problème très simplement. Ce cryptage est réalisé au moment du calcul par l'application et avant leur stockage ou leur transmission. Bien entendu, si le code source est ouvert, cette parade est inutile.

Il est également possible de créer un algorithme qui certifie une application côté client auprès de l'application côté serveur. Cela reprend le principe de la clé publique et de la clé privée dans les systèmes d'encodage comme le PGP (www.pgpi.org)

On peut aussi imaginer une simple certification de code comme celle basée sur le label WebTrust (www.cpawebtrust.org) du commerce électronique. Il faudrait alors déclarer son application cliente désirant correspondre avec un serveur à un organisme constitué d'experts chargés de vérifier le code de l'application. Cela reprend l'idée d'un driver de périphérique certifié par l'éditeur d'un système d'exploitation.

VII.9.12. La persistance

La persistance consiste à enregistrer l'état d'un programme en mémoire sur un support comme un disque dur. La restauration réalise l'opération inverse. La plate-forme Phœnix est tenue de maintenir l'état de tous les agents ; elle est par conséquent responsable du processus de persistance. C'est l'AMS qui se charge de déclencher la persistance et la restauration. Nous nous sommes inspirés de la plate-forme Concordia de Mitsubishi pour la réalisation de ce module. Celle-ci utilise une sérialisation Java avec un support de checkpoints.

On peut considérer deux types de persistance : la persistance dite de sécurité qui permet, par exemple d'effectuer une récupération sur erreur, et une persistance applicative qui permet d'enregistrer les résultats des agents. Nous développerons ici la persistance de sécurité. La seconde utilisera typiquement une base de données quelconque et le choix est laissé à l'utilisateur de Phœnix (c'est-à-dire au concepteur d'agents).

VII.9.12.1. Granularité de la persistance

Bien que l'idéal aurait été de figer l'état de la mémoire à un instant t, ce qui permet de sauvegarder au niveau de l'instruction, la granularité de sauvegarde se limitera pour le moment au niveau des tâches pour une simple raison de complexité. Autrement dit, une tâche en cours d'exécution à l'instant de la sauvegarde ne sera pas prise en compte dans cette version du logiciel. Nous avons imaginé plusieurs alternatives pour le système de la persistance :

1er choix : l'AMS envoie un message aux agents afin qu'ils s'enregistrent à la fin de la tâche courante.
Problème : la cohérence de la plateforme risque de ne pouvoir être garantie (un autre agent peut modifier l'environnement lors de la sauvegarde) ; de plus des tâches peuvent ne pas finir (boucles infinies).

2ème choix : L'AMS appelle endort les agents et appelle une méthode de sauvegarde de chacun d'entre eux. Ces derniers enregistrent une mémoire de sauvegarde de la tâche précédente. Cela implique que les agents doivent mettre à jour cette mémoire à la fin de chaque tâche.
Problème : il peut y avoir beaucoup de sauvegardes des informations et l'occupation mémoire des agents est doublée.

3ème choix : la sauvegarde s'apparente à un vidage mémoire : tout est enregistré à un instant T.
Problème : la restauration risque d'être très délicate en particulier au niveau de l'endroit où l'agent doit reprendre sa tâche.

4ème choix : La sauvegarde s'effectue à un moment où tous les agents sont en fin de tâche ; ainsi la cohérence de la plateforme est maintenue.
Problème : le problème des tâches infinies n'est toujours pas résolu ; cependant une solution partielle a été trouvée (voir spécification de la sauvegarde des agents).

Phoenix intègre un langage de script, présenté plus haut, qui malgré l'intérêt qu'il représente pour découper les rôles en tâches, complexifie grandement la persistance. En effet, l'utilisation de variables au niveau des rôles implique que ces dernières soient également sauvegardées. La granularité de la persistance étant définie au niveau des tâches, cela implique que chaque agent, à la fin de chaque tâche, doit sauvegarder l'état contextuel dû à ces variables afin d'être en mesure de reprendre une tâche sauvegardée. Le langage autorise le parallélisme et il devient donc très compliqué de garantir l'état des variables. La meilleure

solution serait de sérialiser l'arbre et la pile d'exécution du langage de script. Le langage de script ayant été conçu à l'aide d'un générateur de compilateur (JFlex) spécifique à Java, il est raisonnable de considérer que la persistance des tâches soit liée au module du langage de script.

Cette solution semble pourtant être contraire au fait que l'on veut pouvoir se détacher des spécificités de Java telles que la sérialisation d'objets. En effet, on veut pouvoir porter facilement la plate-forme vers un autre langage d'implémentation.

VII.9.12.2. Format de la persistance

Pour une meilleure portabilité entre plates-formes hétérogènes, on se basera sur le format XML pour sérialiser les données. Nous avons également imaginé de crypter les données pour assurer une bonne confidentialité. Nous avons vu précédemment qu'un agent peut établir une communication sécurisée avec un autre agent mais si l'un d'eux enregistre ses données sur un disque de façon totalement transparente, l'intérêt de crypter les communications s'en trouve amoindri.

Comme Concordia, Phoenix est programmé en JAVA. Il est donc possible d'utiliser une sérialisation standard propre au langage. Nous nous sommes plutôt intéressés à un format de sérialisation XML qui présente d'autres avantages :

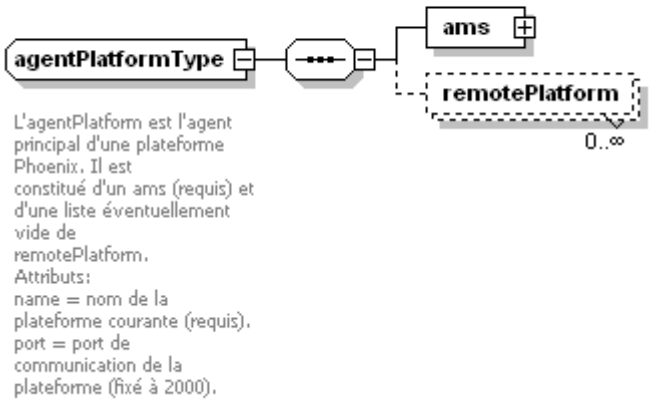
- Indépendance quant au langage cible : Phœnix est actuellement implémentée à l'aide de Java qui propose la sérialisation des objets. Cependant, il peut être envisagé de porter la plateforme vers un autre langage dans lequel la sérialisation n'est pas facilement implantable.
- Format universel : un format commun à toutes les plates-formes permet d'envisager de porter des applicatifs SMA d'une plate-forme à une autre.
- Une DTD générale propre à Phœnix peut permettre son utilisation dans d'autres fonctionnalités (envoi/réception de messages, communication inter plates-formes, Web Services et SOAP, …)
- Phœnix doit suivre les normes dictées par la FIPA. Il est à noter que, par l'intermédiaire de XML couplé à XSLT, il est parfaitement envisageable de convertir un message XML vers un message ACL/SL0 par exemple.

A noter que depuis le 2 mai 2001, XML Schéma est devenu une recommandation du w3c. Une étude de ce langage de définition de modèle XML, m'a décidé à l'utiliser en lieu et place d'une DTD classique. En effet, il présente, entre autres améliorations, un aspect d'héritage des entités qui permettra de modéliser plus finement l'architecture de Phœnix. Ainsi, une entité CognitiveAgent pourra être utilisée en lieu et place d'une entité AbstractAgent dans un fichier XML. Voici le détail du schéma **phoenix.xsd**

schéma location	**D:\Phoenix\essaiXML\phoenix.xsd**	
Elements	***Groups***	***Complex types***
agent	system	accType
cognitiveAgent		agentPlatformType
platform		agentRefType
role		agentType
skill		amsType
		dfType
		environmentType
		mailboxType
		marketInformationType
		marketType

		memoryType
		remotePlatformType

Pour chacun des éléments et des types complexes, nous listons ses attributs, ses enfants, son utilisation etc. Nous définissons ainsi un schéma XML pour chacun d'entre eux. Nous présenterons ci-dessous un seul exemple pour le type complexe agentPlatformType. Les autres types sont définis dans les dossiers techniques du module Persistance du projet Phoenix.

diagram	
children	**ams remotePlatform**
used by	element **system/agentPlatform**

attributes

Name	Type	Use	Default	Fixed	Annotation
port	xs:positiveInteger			2000	
name	xs:string	required			

annotation

documentation

L'<agentPlatform> est l'agent principal d'une plateforme Phoenix. Il est constitué d'un <ams> (requis) et d'une liste éventuellement vide de <remotePlatform>.
Attributs:
name = nom de la plateforme courante (requis).
port = port de communication de la plateforme (fixé à 2000).

documentation

the <agentPlatform> is the main agent of a Phoenix platform. It is made of an <ams> and an eventually empty list of <remotePlatform>.
Attributs:
name = the name of the current platform (required).
port = the platform's communication port (fixed at 2000).

source

```
<xs:complexType name="agentPlatformType">
  <xs:annotation>
    <xs:documentation xml:lang="fr">
L'<agentPlatform> est l'agent principal d'une plateforme Phoenix. Il est
constitué d'un <ams> (requis) et d'une liste éventuellement vide de
<remotePlatform>.
Attributs:
name = nom de la plateforme courante (requis).
port = port de communication de la plateforme (fixé à  2000).
</xs:documentation>
    <xs:documentation xml:lang="en">
the <agentPlatform> is the main agent of a Phoenix platform. It is made of an <ams>
```

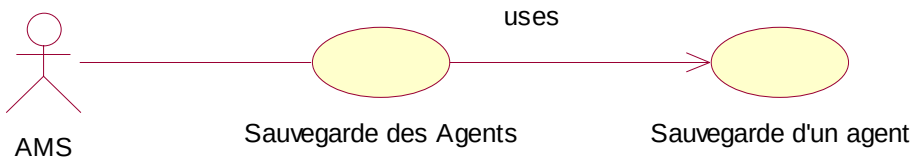
```
and an eventually empty list of <remotePlatform>.
Attributes:
name = the name of the current platform (required).
port = the platform's communication port (fixed at 2000).
</xs:documentation>
 </xs:annotation>
 <xs:sequence>
  <xs:element name="ams" type="amsType"/>
  <xs:element name="remotePlatform" type="remotePlatformType" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:attribute name="port" type="xs:positiveInteger" fixed="2000"/>
 <xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>
```

VII.9.12.3. Principe de sauvegarde et de restauration

L'AMS est responsable de l'intégrité de la plateforme ; il est donc chargé de la persistance. Pour ce faire, il demande à tous les agents de la plate-forme de s'enregistrer. Il indique à tous les agents qu'une sauvegarde va être effectuée. Ces derniers terminent la tâche en cours à la suite de laquelle ils envoient un message à l'AMS indiquant qu'ils sont prêts et en attente. L'AMS tient à jour une liste des agents ayant répondu pendant un temps d'attente donné. A l'issue de ce temps d'attente, l'AMS stoppe tous les agents (y compris ceux qui n'ont pas répondu), demande aux agents de sa liste à s'enregistrer et réveille tous les agents de la plateforme. Ainsi, les démons (tâche avec une boucle infinie) ne participent pas à la persistance mais ne la bloque pas.

En ce qui concerne la sauvegarde en elle-même, les agents construisent un arbre DOM (Document Object Model) qui sera ensuite sérialisé dans un fichier dont le nom sera celui de l'AID de l'agent. A noter, que la validation de l'arbre DOM contre le schéma *phoenix.xsd* n'est ici pas nécessaire et ne sera appliquée que lors de la restauration des agents ; ceci afin de se garantir contre d'éventuelles modifications accidentelles des fichiers (A noter que la restauration n'utilisera pas un arbre DOM mais emploiera le modèle SAX (Simple API for XML) pour des raisons de performance).

L'arbre DOM sera construit avec tous les composants de l'agent candidat à la persistance (voir phoenix.xsd). Le schéma UML du cas d'utilisation de la sauvegarde est le suivant :



Cas d'utilisation UML - Sauvegarde des agents

Du fait de l'implémentation existante, toutes les tâches d'une plate-forme héritent de *AbstractTask*. Par conséquent et par soucis d'efficacité, nous n'utiliserons pas le système des messages de la plate-forme, mais un indicateur statique (flag booléen) dans l'*AbstractTask* notifiant à tous les agents qu'à l'issue de leur tâche en cours, ils doivent s'arrêter et indiquer à l'AMS qu'ils sont prêts à être enregistrés.

Les objets *AMS, Agent, AbstractTask, ReadyAgentList, AgentDOM* et *Node* doivent être enregistrés. L'objet *ReadyAgentList* contient la liste des agents enregistrés auprès de

l'AMS. L'objet *AgentDOM* est l'arbre DOM associé à chaque agent. L'objet *Node* est un composant de l'*AgentDOM*. Le Graphe de collaboration UML est le suivant :

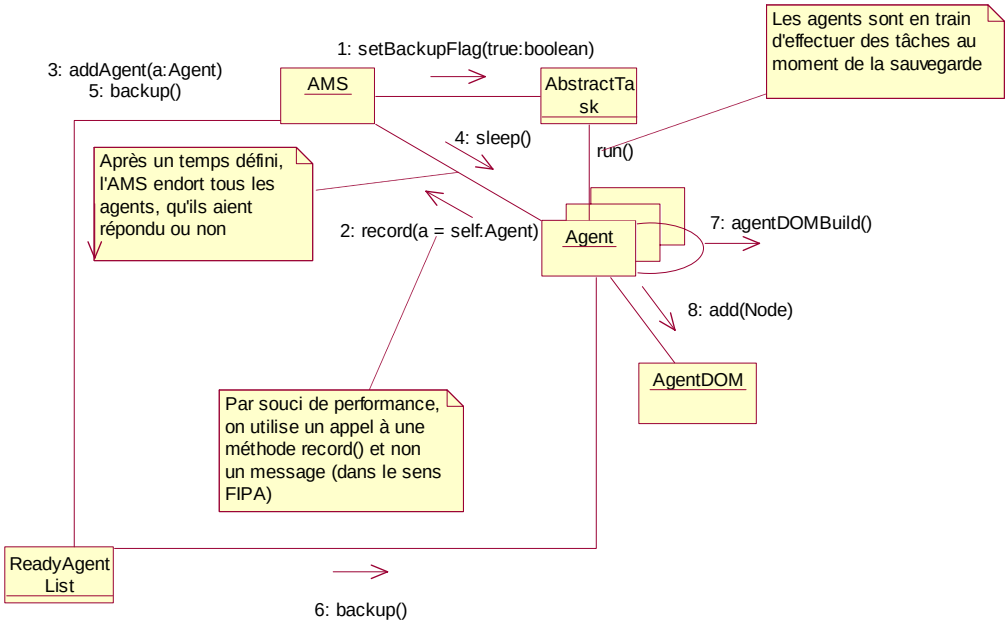
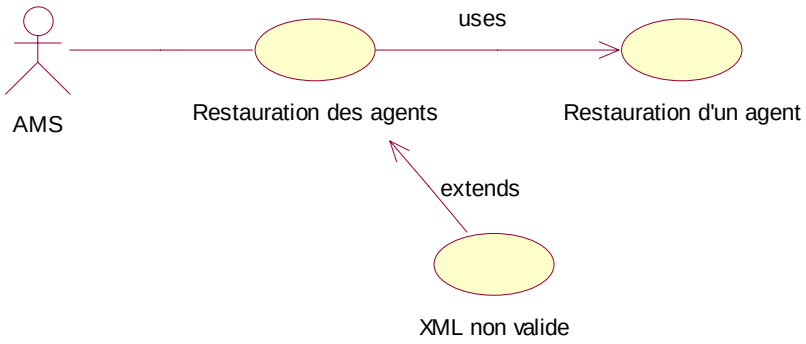


Diagramme UML de collaboration de Sauvegarde des agents

Du fait que la persistance est gérée par l'AMS, ce dernier est naturellement responsable de la restauration des agents. L'utilisateur est le déclencheur de la restauration ; ici le terme d'utilisateur est à prendre au sens le plus large, il désigne tout aussi bien l'utilisateur physique de la plateforme que tout autre programme associé. La plateforme étant stoppée, l'AMS parcourt le répertoire de sauvegarde des agents défini lors de l'initialisation de la plate-forme. Pour chacun de ces fichiers, un parser SAX parcourt la structure définie par ces derniers et construit les objets adéquats selon le contexte courant. La construction des objets de la plate-forme doit utiliser les constructeurs déjà définis. Lorsque toutes les structures ont été recréées, la plate-forme peut être réveillée.

Remarque sur le choix du parser, on utilise ici un parser SAX pour des raisons d'efficacité. En effet la lecture séquentielle des documents XML suffit à la création des agents ; il n'est donc pas nécessaire de construire tout un arbre DOM. Contrairement à la sauvegarde, le parser doit être de type validant selon le schéma phoenix.xsd, ceci afin de garantir la validité des documents XML (transition par réseau, modifications intentionnelles des fichiers, …). Si un des fichiers n'est pas validé, toute la restauration doit être abandonnée. Le schéma UML du cas d'utilisation de la restauration est le suivant :



Cas d'utilisation UML - Restauration des agents

La restauration implique les objets *AMS, Agent, SAXValidatingParser, XMLDocument* et *ValidationException*. L'objet *SAXValidatingParser* est le parser SAX chargé de parcourir et valider les sauvegardes des agents. L'objet *XMLDocument* est le document XML associé aux agents sauvegardés. Sa source est volontairement non définie ; cela peut aussi bien être un fichier que tout autre flux. L'objet *ValidationException* est une exception jetée par le parser SAX lorsqu'un *XMLDocument* n'est pas valide au regard du schéma XML fourni par phoenix.xsd. La restauration doit être annulée si une telle exception est traitée par l'AMS. Le Graphe de collaboration UML est le suivant :

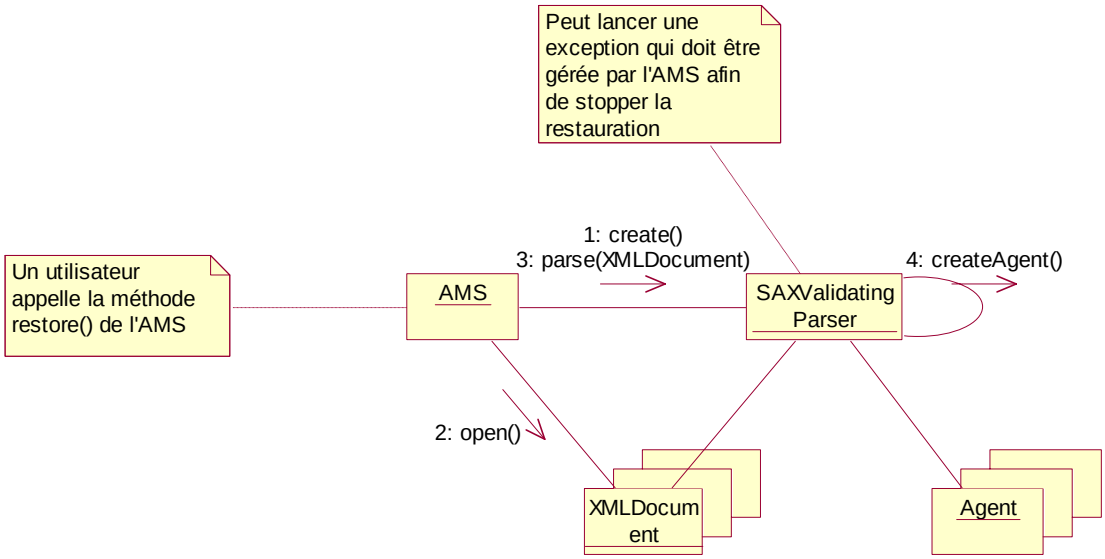
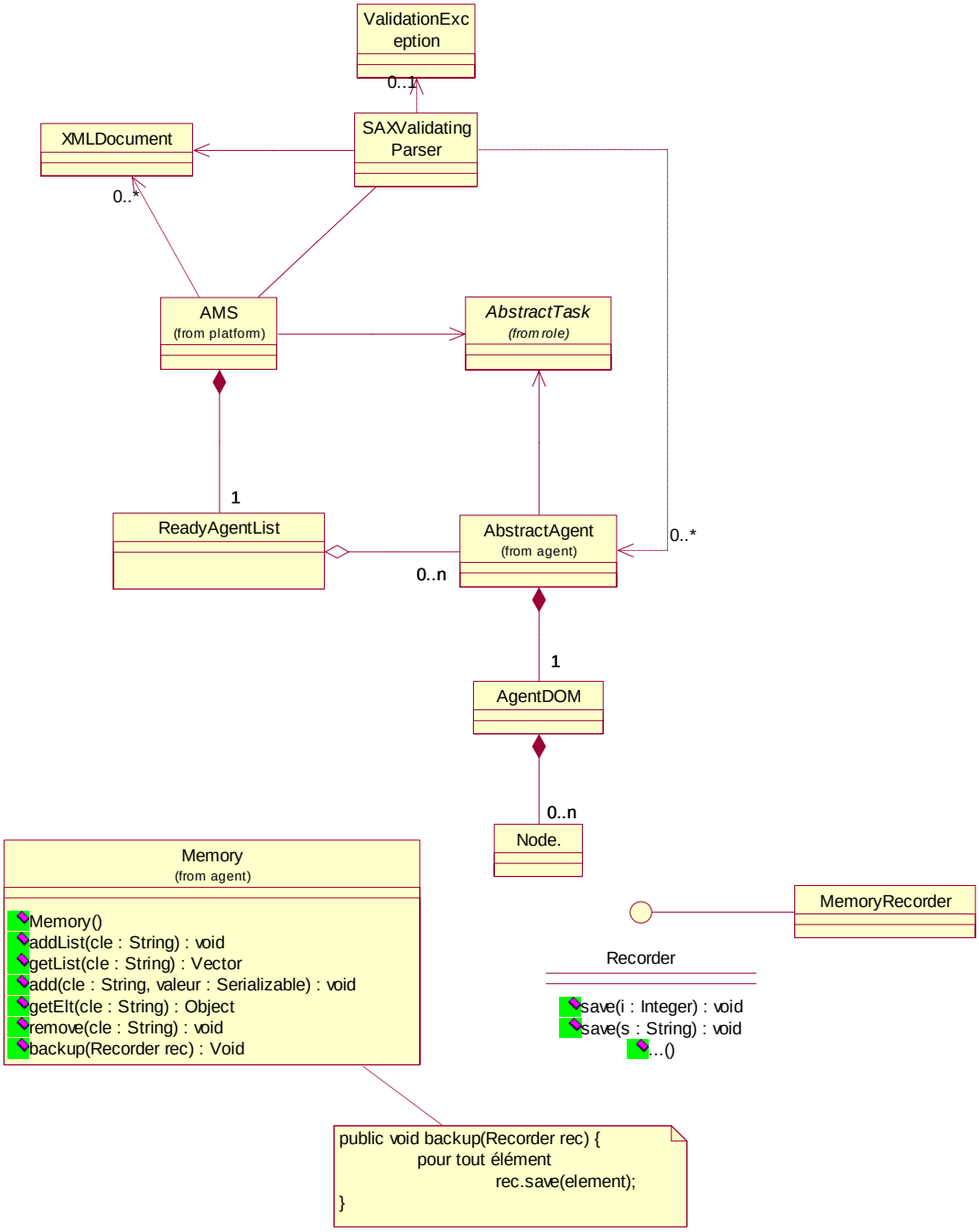


Diagramme UML de collaboration de Restauration des agents

Afin de valider le fonctionnement de notre système de sauvegarde et de restauration, nous avons modifié une petite application Phoenix (recherche des drapeaux) dans laquelle nous avons appelé des méthodes depuis l'IHM. Cet exemple sera décrit en annexe.

Le diagramme suivant présente un graphe d'héritage du système de sauvegarde et de restauration :



VII.10. Une passerelle vers le Grid

Phoenix repose sur un principe d'exécution de compétences distantes ou locales exploitant les ressources locales des plates-formes. Les rôles des agents sont assimilés à des services. Phoenix est donc intrinsèquement un système de Grid Computing. Cependant, il existe aujourd'hui des standards pour autoriser le développement de système de Grille plus harmonieusement. Ces standards diffèrent bien entendu des systèmes implémentés dans Phoenix. Nous proposons des classes interface entre Phoenix et les API de Grid.

VII.10.1. Les services de gestion

Dans le champ des Grid, il existe plusieurs systèmes ayant des rôles bien définis :

VII.10.1.1. GRAM

Le GRAM (Grid Resource Allocation Manager) est le composant dans l'architecture du Grid qui fournit les services de gestion des ressources. Il fournit une API qui permet de soumettre un job sur tout ou partie de la grille, de l'administrer et de le suspendre, de façon transparente. Ce gestionnaire dialogue avec les gestionnaires de tâches des machines distantes qui lui envoient régulièrement des informations sur l'état d'avancement des jobs. Il est composé des éléments suivants :

- Le Gatekeeper, processus de base d'une application Grid Computing, il crée un Job Manager pour manipuler les demandes d'allocation de ressources.
- Le Job Manager, qui démarre un job sur un site. Un job est constitué d'un ou plusieurs processus d'allocation de ressources.
- Le GRAM Reporter, qui met à jour le MDS par les informations sur les ressources.
- Le Moniteur de ressources, qui mène le système local d'ordonnancement et les ressources

VII.10.1.2. MDS

Le MDS (Metacomputing Directory Service) est un fichier de description détaillée de l'ensemble des éléments de la grille. Il permet la découverte et la gestion de l'information sur les ressources impliquées dans la grille. Il s'appuie sur un annuaire LDAP (Lightweight Directory Access Protocol) et contient les informations sur le modèle de la machine, le nombre et le type de processeurs, la taille mémoire, disque etc. Il tient compte des conditions définies par les propriétaires, c'est-à-dire les créneaux horaires, la quantité de puissance machine utilisable etc. Il fournit les services suivants : Le GRIS (Grid Resource Information Service) et le GIIS (Grid Index Information Service).

VII.10.1.3. GSI

Les mécanismes du GSI (Grid Security Infrastructure) sont utilisés pour l'authentification, l'autorisation et la délégation pour le calcul distant. Il permet d'accéder à la grille avec un logon unique et non en étant contraint de passer par les filtres de sécurité de chacun des éléments de la grille. Il utilise une architecture à clé publique avec des certificats pour chaque service de la grille. Le GSI garantit donc la sécurité mais aussi la simplicité d'accès à l'ensemble.

VII.10.2. Un protocole de transfert de fichiers

Basé sur le FTP, le protocole GridFTP est performant fiable et sécurisé. Il est surtout optimisé pour les réseaux étendus à large bande. Deux fonctions, le Globus Replica Management et le Replica Catalog[58], permettent à un utilisateur qui a besoin d'accéder à des quantités très importantes de données d'en extraire une partie et de la répliquer quelque part sur la grille.

[58] Plus d'information sont disponibles sur www.globus.org et sur www.gridforum.org

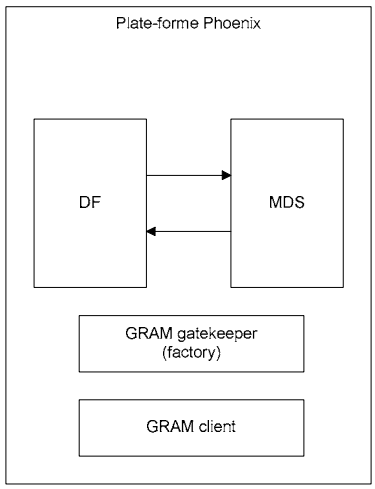
VII.10.3. Accès au Grid par les Web Services

Il existe aujourd'hui le standard OGSA (Open Grid Services Architecture). Cette architecture, issue d'une collaboration entre Globus et IBM vise à rapprocher les concepts de grille informatique et de services web. Chaque élément de la grille fournit un ensemble de ressources, chaque ressource est considérée comme un service de la grille (*GridService*). Un *GridService* est défini comme un web service qui fournit un ensemble d'interfaces bien définies et qui suit des conventions spécifiques. Les services principaux sont : les *factorys* responsables de la création dynamique des services, le registre qui est une sorte de « pages jaunes » contenant les descriptions des services, y compris les *factorys*, et le mapper qui fait correspondre à chaque description de service un identifiant (handle) retourné par un *factory*.

Le Laboratoire National d'Argonne dans l'Illinois compte orienter le développement de la version 3 de son logiciel Globus Toolkit vers le standard OGSA. Globus identifie les ressources, leur état et leur disponibilité, en sécurise l'accès et permet l'exécution de tâches et l'échange de fichiers sur l'ensemble. Ce projet américain, qui a joué le rôle de précurseur est soutenu par IBM, HP ou Sun mais il n'existe pas à ce jour d'API de développement. De plus, les performances des WS risquent d'être pénalisantes pour les applications Grid, qui sont essentiellement utilisées dans le cadre de projets scientifiques. C'est pour cette raison que les chercheurs de l'INRIA, qui travaillent sur Grid ASP se tournent plutôt vers le futur CORBA version 3.

VII.10.4. Adaptation de Phoenix et du modèle FIPA à l'accès au Grid

Le service MDS se rapproche du DF propre au modèle FIPA. Les agents Phoenix décrivent leurs compétences sous formes de services. Ces services peuvent être accessibles via un Web Service. Afin qu'une plate-forme Phoenix puisse être reconnue comme un élément d'une grille, il s'agit d'adapter le DF au MDS. Cela peut être réalisé par une classe « passerelle » chargée d'établir une correspondance entre les deux.



Les plates-formes Phoenix dialoguent entre elles grâce au système FIPA (DF, AMS…) et dialoguent avec un système Grid grâce au MDS et aux différents protocoles requis. Ce système de passerelle est en cours d'implémentation à l'heure où j'écris ces lignes.

VII.11. Conclusion

Comparé à d'autres outils existants dans la littérature, le middleware Phoenix rassemble toutes les caractéristiques des autres systèmes complets :

- un modèle complet tâche, rôle, compétences
- un environnement dans lequel évoluent les agents
- une plate-forme portable distribuée qui permet l'interopérabilité entre des architectures hétérogènes
- une gestion décentralisée des plates-formes disponibles
- des agents communicants capables de négocier entre eux
- des agents pouvant être déplacés d'une plate-forme à une autre

- un langage de programmation spécialisé basé sur Java (avec une extension pour la gestion du parallélisme).

Cependant, le couple YAMAM/Phoenix n'utilise pas de système de contraintes (elles sont remplacées par les compétences et l'environnement), ni d'éditeur graphique pour l'écriture des agents. Cela fera l'objet d'une future extension. Plusieurs notions nouvelles sont introduites dans notre plate-forme, parmi lesquelles :

- la notion d'énergie : chaque agent a pour but principal de maximiser son énergie en menant à bien des rôles.
- la notion d'agent cognitif : les agents sont capables d'apprendre par eux même les compétences (écrites par un développeur) qui lui permettent de prendre des rôles plus lucratifs en terme d'énergie.
- la notion de mémoire : l'agent se souvient de ses actions et de son environnement et fait des suppositions sur les parties de l'environnement qu'il ne connaît pas. Sa mémoire est un premier pas vers une forme d'état mental. Elle peut mener l'agent à changer de rôle ou à donner des résultats légèrement différents.
- l'architecture point à point qui élimine la notion de client ou de serveur.

Phoenix est un middleware qui comprend environ 200 classes, ce qui le positionne d'ores et déjà dans la catégorie des logiciels complexes. Outre le nombre de classes, nous utilisons des algorithmes parfois complexes comme des réseaux de neurones, ce qui rend le code assez difficile à maintenir. Cerise sur le gâteau, l'exécution de Phoenix repose sur une utilisation massivement parallèle, qui induit nécessairement une philosophie de développement différente. Le temps de mise au point fût donc considérable et nous avons tenté d'appliquer des techniques de gestion de projet comme l'Extreme Programming [Benard & al., 2002] pour piloter au mieux les équipes et assurer la qualité du processus de développement.

Nous possédons aujourd'hui un prototype opérationnel du logiciel qui nous permet de développer des petites et des grandes applications. Le bon fonctionnement de notre plate-forme fut validé avec la réalisation de quelques Toy-Problems[59] présentés en annexe.

La partie dédiée à l'auto-organisation du système par échange d'énergie devra être approfondie pour en tirer davantage de performance. Un agent « contrôleur » pourrait être développé pour connaître les flux d'énergie circulant dans le logiciel. Cela pourrait nous permettre d'améliorer son fonctionnement global et peut être en tirer des règles comportementales. Le module décisionnel d'un agent est en relation directe avec ce système d'échange d'énergie et devra être amélioré pour tirer mieux partie de la mémoire des actions et des décisions. Nous poursuivons notre étude sur un système d'apprentissage qui permettrait aux agents de ne pas commettre plusieurs fois les mêmes erreurs et favoriser ainsi la circulation de l'énergie et finalement accroître l'efficacité globale.

Le prototype dont nous disposons est suffisamment stable pour assurer le développement d'applications de plus grande envergure. Nous présentons dans le prochain

[59] Les Toy-Problems sont des petits programmes montrant une ou plusieurs caractéristiques du middleware comme la distribution, l'auto-adaptation etc. Ces implémentations rapides permettent également de tester la fiabilité du produit et offrent un premier retour d'expérience. Un certain nombre d'exemples types ont été proposés par le groupe ASA [ASA, 2002]. Ils permettent de comparer le nombre de lignes de code nécessaires pour l'implémentation de chacun d'entre eux sur différentes plates-formes et comparer ainsi le niveau d'abstraction du langage.

chapitre le projet de simulation ANTIC dont le fonctionnement s’appuie sur notre middleware Phoenix.

Chapitre VIII. Une application professionnelle du Middleware Phoenix : Le projet ANTIC

« Le privilège des grands, c'est de voir les catastrophes d'une terrasse » – Jean Giraudoux.

Résumé : ANTIC est un logiciel de simulation pour les sapeurs-pompiers. Il propose la simulation d'une catastrophe industrielle qui sert de support de formation pour les décideurs de la Sécurité Civile. Cette simulation potentiellement distribuée s'appuie sur notre plate-forme SMA.

VIII.1. Avant-Propos

L'utilisation des grands réseaux permet de jouer une simulation sans réunir physiquement les participants. ANTIC (Apprentissage des Nouvelles Technologies pour l'Induction des Compétences) était un projet originellement financé par le ministère de l'industrie dans le cadre des autoroutes de l'information. Le projet ANTIC I consistait à construire un simulateur de catastrophe distribué pour la formation interactive en ligne des officiers supérieurs de la Sécurité Civile. L'architecture informatique de ce premier projet est exposée plus loin dans ce chapitre. Fort de son succès, ce projet a connu une suite, également financée par le Ministère de l'Industrie : ANTIC II. Ce deuxième volet s'adressait plutôt aux sous-officiers de la Sécurité Civile, avec une approche plus orientée terrain. Ces différents projets entrent dans le cadre d'une formation à la gestion de catastrophe dans une optique opérationnelle. Une catastrophe est vue comme une accumulation d'incidents se déroulant dans le temps.

VIII.2. Terminologie

Comme dans chaque domaine, il est souhaitable de définir quelques mots de vocabulaire. Le domaine de la formation n'échappe pas à cette règle :

- *Un formateur* : Il transfère lui-même un savoir (le contenu pédagogique) à un apprenant. Selon le cas, il sera appelé « enseignant », « professeur » ou « intervenant ».
- *Un tuteur* : guide de l'apprenant. Il veille à ce que celui-ci acquière bien les compétences prévues selon le plan de formation. Le cas échéant, il peut conseiller l'apprenant pour qu'il s'améliore ou l'aider ponctuellement mais il n'enseigne pas. Dans le cas de la formation en alternance, il peut assurer un suivi de l'apprenti au sein de l'entreprise.
- *Le moniteur* : ce terme assez vague peut désigner aussi bien un formateur qu'un tuteur.
- *Le conseiller pédagogique* : il encadre ou coordonne à la fois les apprenants, les tuteurs et les formateurs. Il conseille l'apprenant sur le choix de son parcours et veille au bon déroulement des formations.

- *Apprenant* : celui qui apprend. Il acquiert des compétences au cours d'une formation. Ce terme ne précise pas le statut de la personne concernée (élève, stagiaire, apprenti, etc.)
- *Stage* : programme d'acquisition de compétences.
- *Session de formation* : moment durant lequel l'apprenant acquiert une série donnée de compétences. A un stage pourra correspondre un grand nombre de sessions de formation, chacune étant réalisée au bénéfice d'un groupe distinct d'apprenants.

VIII.3. Etude de l'existant - simulation pour la sécurité

La simulation informatique est un outil complémentaire pour la formation des Sapeurs-Pompiers. Elle ne remplace en rien un exercice sur le terrain mais elle présente des avantages non négligeables. Un exercice terrain nécessite une logistique lourde et coûteuse et est soumis à des contraintes météorologiques. De plus, la répétitivité est insuffisante et sa reproductibilité est aléatoire. L'interactivité est difficilement maîtrisable et il n'y a pas de situation extrême. Toutes ces raisons suffisent à justifier une simulation informatique. Le site de Valabre est connu chez les Sapeurs-Pompiers pour être un des plus beaux sites de simulation en France.

Plusieurs projets furent montés ces quelques vingt dernières années en vue d'améliorer, lors d'une catastrophe, le processus décisionnel des officiers de la Sécurité Civile. H. Caetano et J. Mathieu [Caetano & Mathieu, 1997] ont développé des systèmes de formation fondés sur la simulation pour la prise de décision en situation de crise. Le système SAGESSE était destiné aux officiers de la sécurité civile, il proposait une simulation d'un feu d'hydrocarbure pour tester les aptitudes d'officiers au commandement. Un module d'E.I.A.O venait compléter le système [Worms, 1992]. Le projet Européen FORMENTOR-Formation s'appuie sur le système FORMENTOR développé par le centre de recherche de la CEE à Ispra dans le cadre d'un projet EUREKA, avec la participation de Aérospatiale Protection Systèmes et Cap Sogeti Innovation. L'objectif est la formation à la conduite, et à la prise de décision en cas d'incidents, de processus industriels complexes tels que le remplissage en combustible d'un satellite lors de son lancement. Le système INTEMPERIES [Caetano & Mathieu, 1997] était utilisé pour l'analyse des catastrophes naturelles [Caetano, 1992]. Il servait de noyau de simulation à la base de systèmes prônant la coopération entre institutions intervenantes en cas de catastrophe. Le système de raisonnement à base de cas REBECAS de Sophie Rougegrez [Rougegrez, 1994] fut utilisé dans un système de prédiction de comportement de feu de forêt. Un système à base de connaissances fut construit par Denis Malavieille dans [Malavieille & al., 1995] dans le cadre d'une aide à la décision lors d'accidents majeurs. Ce système est connexe à MKMS [Ermine & al., 1996] qui est une méthode pour la gestion des connaissances.

Tous ces projets montrent l'intérêt que porte la Sécurité Civile aux systèmes d'informations aussi divers soient-ils. En amont d'une catastrophe, de nombreux systèmes comme [Regis & al., 2002] ou [Buisson, 1990] tentent de modéliser et détecter des événements naturels. Nous allons présenter dans les prochains paragraphes quelques catégories de systèmes informatiques inhérents à la gestion ou la prévention de catastrophe.

VIII.3.1. Système d'information pour la gestion de crise

Une situation catastrophe peut parfois générer une réelle situation de crise[60]. La crise du Mont Saint Odile en 1992 en est un exemple typique. Des recherches furent entreprises pour tenter de prévenir ce genre de situation. Le projet ABIS (Agent Based Interpretation System) est géré par Alain Cardon, actuellement directeur du LIH. Le système modélisé dans ce projet traite de situations complexes comme des situations d'urgence ou de crise [Cardon, 1997] et [Cardon & Durand, 1997]. Le système a pour objectif de fournir des résultats au niveau du sens du phénomène qu'il décrit afin de prendre des décisions coopératives et de justifier les intentions des différents opérateurs d'une situation de crise [Durand & al., 1999]. Il permet en quelque sorte de représenter le processus d'élaboration de la décision par un processus d'émergence des caractères de l'organisation globale [Cardon, 1996b], [Tranouez & al., 1999] et [Durand & al., 1999]

Il utilise pour cela des agents spécifiques aspectuels, des agents morphologiques et des agents homéostatiques, les rôles de ces derniers étant de vérifier des mesures internes de cohérence et de stabilité. Le prototype de ce système est décrit dans la thèse de Franck Lesage [Lesage, 2000]. Ce projet repose sur une plate-forme spécifique nommée DMAP (Distributed MultiAgent Platform) développée en VisualWorks SmallTalk.

VIII.3.2. Système d'aide à la gestion de crise

Dans [Jaber & al., 1998] est présenté un système d'agents logiciels intelligents pour favoriser la coopération entre systèmes d'aide à la décision dédiés à la gestion de crise.

Ce système permet de faire cohabiter des Systèmes d'Aide à la Décision de différentes natures. Le principe repose sur un agent logiciel qui sert de couche au dessus de chaque SAD. Les agents communiquent alors entre eux. Cela évite de faire une re-ingénierie de chaque SAD pour les faire communiquer. Ce système entre dans le cadre de recherches conduites dans le domaine de la prévention et la lutte contre les incendies de forets. Voir les travaux de [Guarnieri, 1994] et [Wybo & al., 1998]. L'auteur signale que le principe n'est pas opérant dans le cadre de la gestion de crise, car la complémentarité et l'échange d'information et de services est alors un facteur indispensable.

VIII.3.3. Système de réalité virtuelle pour la formation aux plans d'intervention

Le projet SIMPLI-CIVILE est développé à l'Ecole Nationale d'Ingénieurs de Brest. Il concerne la simulation des plans d'Intervention de la Sécurité Civile. La réalisation repose sur un système multi-agents et sur une simulation interactive graphique en Réalité Virtuelle.

Le projet a donc pour but la modélisation de sites industriels à risques pour former les cadres de la Sécurité Civile. Il est né d'une collaboration entre l'ENIB et le SDIS29. Il se compose de deux phases : la première concerne l'acquisition des données (SIG, photos, plans…) pour la modélisation des sites et la conception de plans d'intervention. La seconde étape consiste en la réalisation d'applications de simulation et de formation.

Pour la phase d'acquisition des données, il y a plusieurs sources telles que les BDD urbaines (SIG) pour les réseaux de transports, d'eau, d'électricité etc. et les données du site industriel tel que le plan de masse, des photographies du site, la localisation d'outils de sécurité etc. L'acquisition de ces données permet à la Sécurité Civile de réaliser des Plans

[60] Patrick Lagadec : Pour un fonctionnement efficace des cellules de crise : http://www.ecole.org/2/CF191295.pdf

d'Opérations Internes (POI) qui concernent la mise en œuvre des moyens internes de l'entreprise en cas d'incident, des Plans Particuliers d'Interventions (PPI) qui concernent l'organisation des secours publics et des Plans Etablissement Répertorié (PER) qui permettent au premier commandant des opérations de secours la prise de décisions rapides. Ces données permettent aussi de représenter assez fidèlement le site en trois dimensions. Les plans d'interventions sont modélisés sous forme de rôles (actions) et de scénarios (contraintes entre actions). Le projet utilise de la logique temporelle (extension de la logique de Allen).

Une fois le site modélisé, il est possible dans la phase 2, d'effectuer des simulations de type test de sécurité. Par exemple on peut étudier l'impact que peut avoir une explosion sur une cuve de produits chimiques. Cela permet d'effectuer des sessions de simulation pour les officiers dans le cadre de leur formation. L'outil permet l'immersion d'un instructeur et d'un stagiaire dans le site en 3D par le biais d'avatars. Au cours d'une simulation, l'instructeur réalise dynamiquement des scénarios d'incidents (fuite, feu…) et évalue la réaction du stagiaire. Le stagiaire dispose d'informations (les plans d'interventions, nature du stockage…) et des moyens matériels d'intervention ainsi que des outils de communications. Le stagiaire prend aussi en charge la gestion du personnel de sécurité qui est représenté par des agents informatiques autonomes. Cet outil se nomme AréVi [Harrouet, 2000] (Atelier de Réalité Virtuelle) et constitue le résultat de la thèse de Ronan Querrec. C'est une plate-forme de développement d'environnements virtuels 3D distribués. Il gère aussi des périphériques de RV comme les souris 3D et permet l'export au format VRML. Cette plate-forme fut développée au LI2 (Laboratoire d'Informatique Industrielle) de l'ENIB. Elle est couplée à Oris[61], un langage de programmation parallèle permettant aussi la modification dynamique du code. Il permet notamment la mise au point en ligne de systèmes multi-agents et constitue aussi à ce titre une plate-forme SMA relativement générique qui propose un langage interprété. Il s'interface avec les langages C++, JAVA et Prolog et fonctionne sur TCP-IP. Oris est aussi développé par le laboratoire LI2.

VIII.3.4. Système de détection de feux

Le système Bosque [Criado & Cardonna, 1996] est dédié à la détection précoce de départs de feux de forêts. Il permet de renseigner le centre de commandement sur l'évolution d'un feu grâce à un réseau de caméras. Il gère des images d'un feu et des images infrarouges. Il s'agit d'un système de détection réel et non d'un simulateur. A noter que T2M Automation[62], fort de son expertise dans le domaine de l'analyse et du traitement de l'image, et en partenariat avec le Centre d'Essais et de Recherche de l'ENtente[63] (C.E.R.EN.) se sont attachés à apporter une solution adaptée au problème de la détection automatique des feux de forêts. Le système ARTIS FIRE se compose d'un réseau de capteurs répartis sur l'ensemble de la zone à surveiller, ainsi que d'un Poste Central de Contrôle (PCC). Il existe aussi le système DEDICS (Distributed Environmental Distaer Information and Control System), [Wybo & al., 1998], qui englobe la détection, la navigation GPS et l'aide à la prise de décision.

VIII.3.5. Système de communication de groupe

Le système Florinus [Schmidt, 1995] est spécialisé dans la communication et l'échange d'information entre différents groupes de pompiers engagés sur le terrain. Il assure la localisation des moyens terrestres et aériens par GPS et la gestion d'échanges radio pour donner le point d'une situation sur le terrain. Il gère aussi l'état des ressources en eau.

[61] http://www.enib.fr/~harrouet/oris.html
[62] Le site web de la société T2M est accessible sur www.t2m.fr
[63] Voir le site internet du Pôle de Recherche et de Formation sur la Gestion des Risques et des Dangers http://www.cindy.ensmp.fr/D08_HOFF.pdf

VIII.3.6. Système de météorologie

Les conditions climatiques sont d'une importance capitale dans la gestion d'une catastrophe. Le système Météo [Chambina & al., 1998] assure la gestion et l'analyse des données météorologiques provenant des capteurs de Météo-France. Il fournit des informations comme la direction et la vitesse du vent, l'humidité, la température et la pluviométrie. Dans le même ordre d'idée, Weatherlink[64] est un projet pilote d'alerte météo aux Etats-Unis. Il s'agit d'un système d'alerte des habitants en cas de cyclone ou de tornades. Les internautes inscrits au programme ont sur leur écran une fenêtre pop-up rafraîchie toutes les dix minutes, qui vire au rouge et lance une alerte sonore en cas de risques de cyclones ou encore d'inondations. Développé dans le Colorado par Front Range Internet, le projet devrait s'étendre à d'autres états américains.

VIII.3.7. Système d'aide à la décision

Lorsqu'une catastrophe se manifeste, les Sapeurs-Pompiers évaluent la situation en fonction d'un ensemble de critère et prennent une décision tactique pour la juguler. Les actions doivent être réfléchies rapidement et mise en œuvre efficacement. Comme disait Bergson, « Il faut agir en homme de pensée et penser en homme d'action ». Afin d'améliorer qualitativement le prise de décision humaine, elle peut être assistée par un outil informatique. Le système *Wilfried* [Guarnieri & Wybo, 1995] est un système d'aide à la décision spécialisé dans l'analyse du risque de feu de forêt. Il fournit un ensemble de modèles de simulation, combinés à un Système d'Information Géographique. Il interpole des données météo, calcule des niveaux de risques, gère les ressources en eau et simule la propagation des feux.

VIII.3.8. Système de main courante

Le système MCI [Vauche & Guarnieri, 1997] est une main courante informatisée. C'est un SGBD qui gère l'ensemble de l'information traitée par le Centre Opérationnel de Commandement. Il gère donc le suivi des messages, le suivi des situations et effectue un certain nombre de statistiques. Un système similaire fut appliqué dans les débuts du projet ANTIC.

VIII.4. ANTIC I

Le projet ANTIC I était le résultat d'une réponse à l'appel à projet « Autoroutes de l'information » du Ministère de l'Industrie. Ce premier projet s'adressait à l'école nationale des Sapeurs-Pompiers. Il s'agissait de développer un logiciel de simulation et de formation multimédia sur réseau pour les officiers Sapeurs-Pompiers français. Le logiciel devait permettre de développer la simulation d'exercices d'état-major utilisés pour la formation et l'entraînement à la gestion opérationnelle et au commandement des officiers de la sécurité civile. Ces exercices sont dispensés sous la forme de scénarios catastrophes joués par des équipes de stagiaires sous le contrôle d'un instructeur, équipes et instructeurs étant situés dans des lieux séparés et distants. Plusieurs scénarios furent imaginés permettant d'englober la quasi-totalité des grands types de catastrophes. La sécurité civile utilise donc six scénarii de base autour desquels peuvent venir se greffer de multiples variantes. Le simulateur assure pour cela la mise en situation, la gestion du déroulement d'un scénario et l'évaluation des stagiaires.

[64] http://home.frii.com/community/weatherfaq.html

L'intérêt pédagogique fût d'assurer une formation homogène et efficace à des officiers opérant dans des régions différentes du pays tout en améliorant qualitativement le suivi et le débriefing de l'instructeur. L'objectif du projet était donc de placer des officiers en situation de crise maximum pendant une période d'environ deux heures et d'évaluer leurs réactions face à l'ampleur de la catastrophe. Toutes les actions et les décisions doivent être mémorisées afin d'être analysées par le formateur responsable de la simulation.

Une contrainte majeure est d'incorporer l'outil informatique dans les méthodes de travail existantes sans les modifier dans un premier temps. A terme il est probable que la technologie devenant de plus en plus présente dans notre vie, les méthodes de travail évolueront ou même changeront radicalement aidées par l'avènement de l'ère numérique.

VIII.4.1. Les partenaires du projet

Cinq partenaires sont regroupés autour de ce projet.

- ***L'INESC*** (Institut National d'Etude de la Sécurité Civile) dirige le projet et fournit l'expertise.
- ***L'INSA*** de Rouen est responsable de l'architecture du système, du module de simulation et de l'intégration.
- ***L'équipe CNRS*** de J. Rogalski [Rogalski & al., 1989] est chargée de l'ergonomie.
- ***La société ADB - EURIWARE*** est chargée de la modélisation et de l'implémentation des données dans la base de données Matisse, et des modules cartographiques.
- ***La société A6 Mediaguide*** est chargée de l'éditeur de scénario et du module de débriefing.

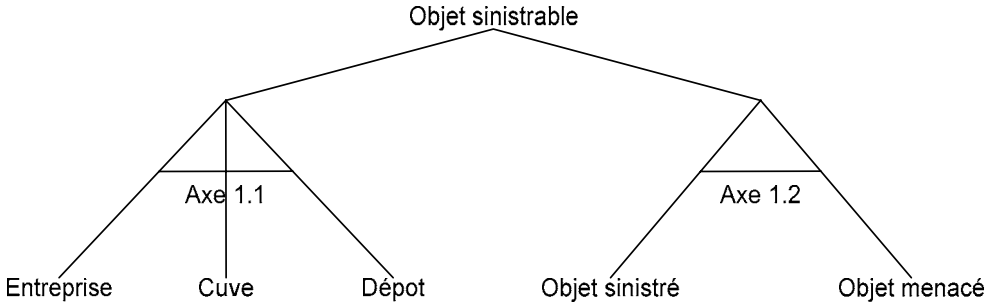
VIII.4.2. Technologies mises en oeuvre

Le logiciel est architecturé autour d'une base de données objets centrale contenant un scénario catastrophe. Un simulateur situé sur le serveur de réseau se charge de lire des scénarii et de distribuer des événements temporels aux différents logiciels clients. Ce projet utilise des technologies comme la distribution d'objets, la vidéo numérique, la synthèse vocale et la visioconférence. En outre nous souhaitons également pouvoir « simuler » un stagiaire manquant dans la simulation. Pour cela, un comportement préalablement programmé est entré dans la base de données couvrant toutes les actions à effectuer en réponse à un événement déterminé. Ce rôle peut être tenu par un agent logiciel particulier. La première version d'ANTIC était néanmoins très statique car conçue autour d'un scénario lui-même assez statique.

VIII.4.3. Ontologie ANTIC I

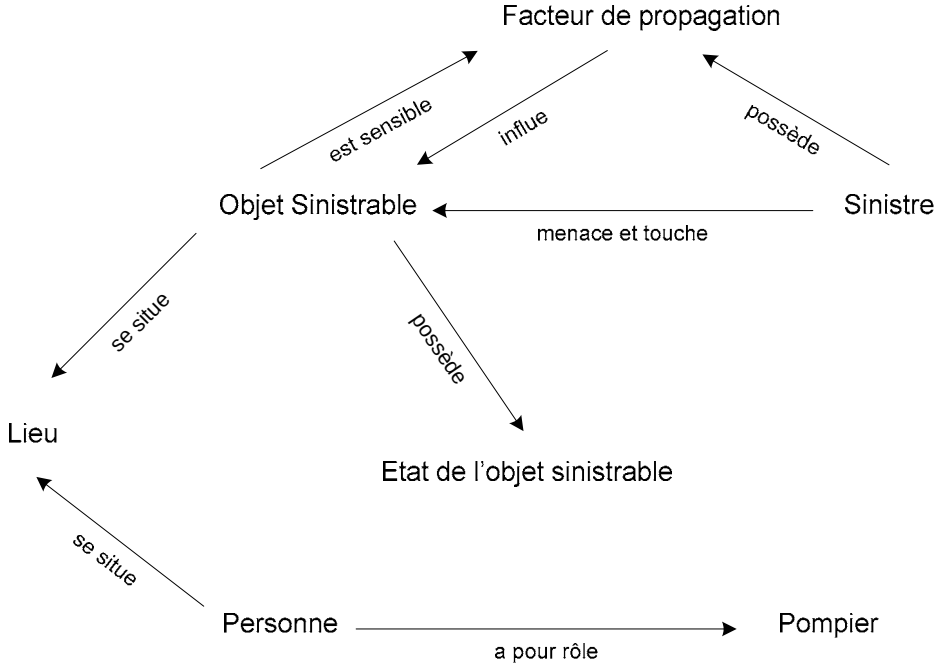
Préalablement à la conception du projet, nous sommes passés par une phase d'analyse pour laquelle nous avons défini une ontologie du domaine. Nous avons utilisé l'outil X-Kaos.[65] Les ontologies produites ressemblent à :

[65] Site web de Xchaos : http://cmap.coginst.uwf.edu/



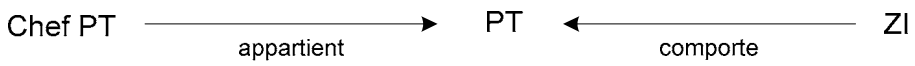
VIII.4.4. Vues et définitions des liens sémantiques

Nous avons ensuite défini des liens sémantiques sur des graphes. Un exemple est donné ci-dessous :



ZI==comporte(1,1)==>PT : une zone d'intervention ne comporte qu'un point de transit, et un point de transit ne comporte qu'une zone d'intervention

Chef PT==appartient(1,1)==> PT : un chef PT n'appartient qu'à un point de transit, et un point de transit n'a qu'un chef PT



Nous ne pouvons présenter dans ce document qu'un petit extrait de l'ontologie. Il est possible de se reporter à [Jouet, 2001] dans le cadre de son rapport de DEA qui présente une ontologie plus complète du projet.

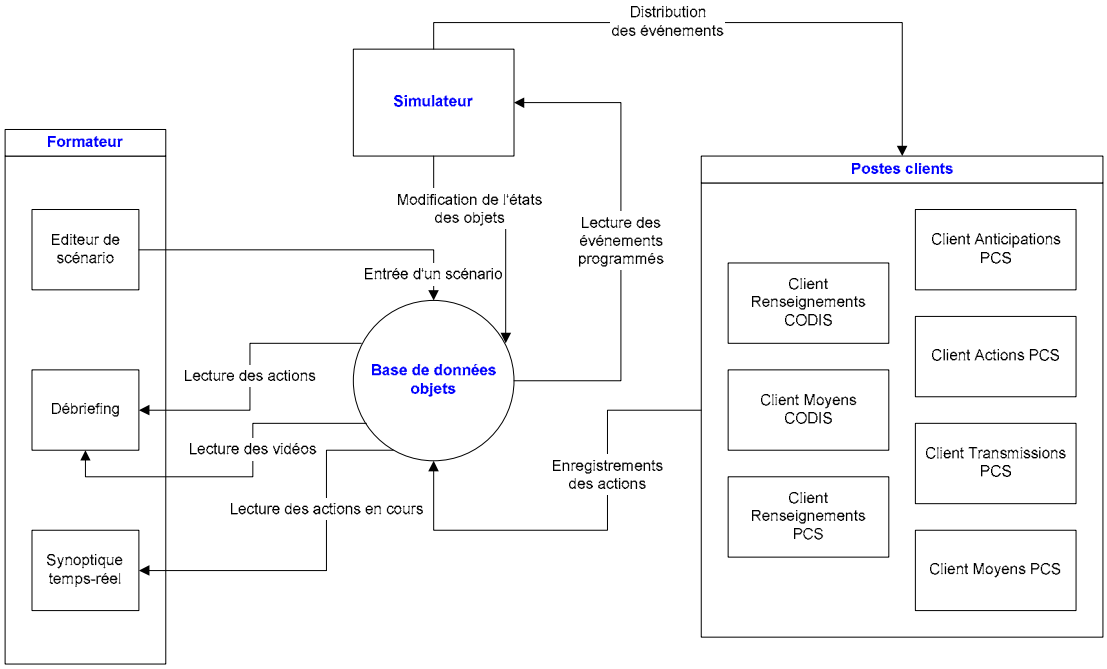
VIII.4.5. Les composantes du système

Le système de simulation était composé de plusieurs outils :

- le simulateur lui-même qui déclenche des événements programmés lors de la création du scénario de crise, il simule l'évolution du sinistre selon des règles préétablies, et

gère les communications Homme/Homme (ex: stagiaire vers instructeur) Homme/Machine (ex: stagiaire vers chef d'agrès) Machine/Machine (ex: responsable de l'entreprise sinistrée vers chef d'agrès)

- le debriefing qui permet à l'instructeur de revenir sur des éléments précis du déroulement de la simulation et de rejouer certaines parties, afin de mieux comprendre comment et pourquoi le sinistre a évolué comme il l'a fait et d'apporter des critiques sur les actions des stagiaires.
- l'éditeur de scénario, qui permet à un instructeur d'intégrer un nouveau scénario qui sera utilisé lors de la simulation du sinistre. Sont décrits ici, les éléments du monde qui sont utilisés par les acteurs de la simulation (simulateur, instructeur, stagiaire). L'éditeur de scénario permet d'instancier l'ontologie du domaine.
- Les postes opérateurs qui fournissent aux stagiaires les moyens d'interaction avec le système. Ils sont constitués d'un poste informatique client donnant accès aux outils liés aux divers rôles des officiers, et d'un certain nombre de périphériques (fax, téléphone, …).



Dans la configuration la plus simple, les participants sont regroupés en trois équipes réparties dans trois lieux différents, proches ou distants. La première regroupe les stagiaires chargés des tâches dévolues au CODIS (Centre Opérationnel Départemental d'Incendie et de Secours). La seconde équipe regroupe les stagiaires attachés au PCS (Poste de Commandement sur Site). La dernière équipe est constituée du ou des formateurs.

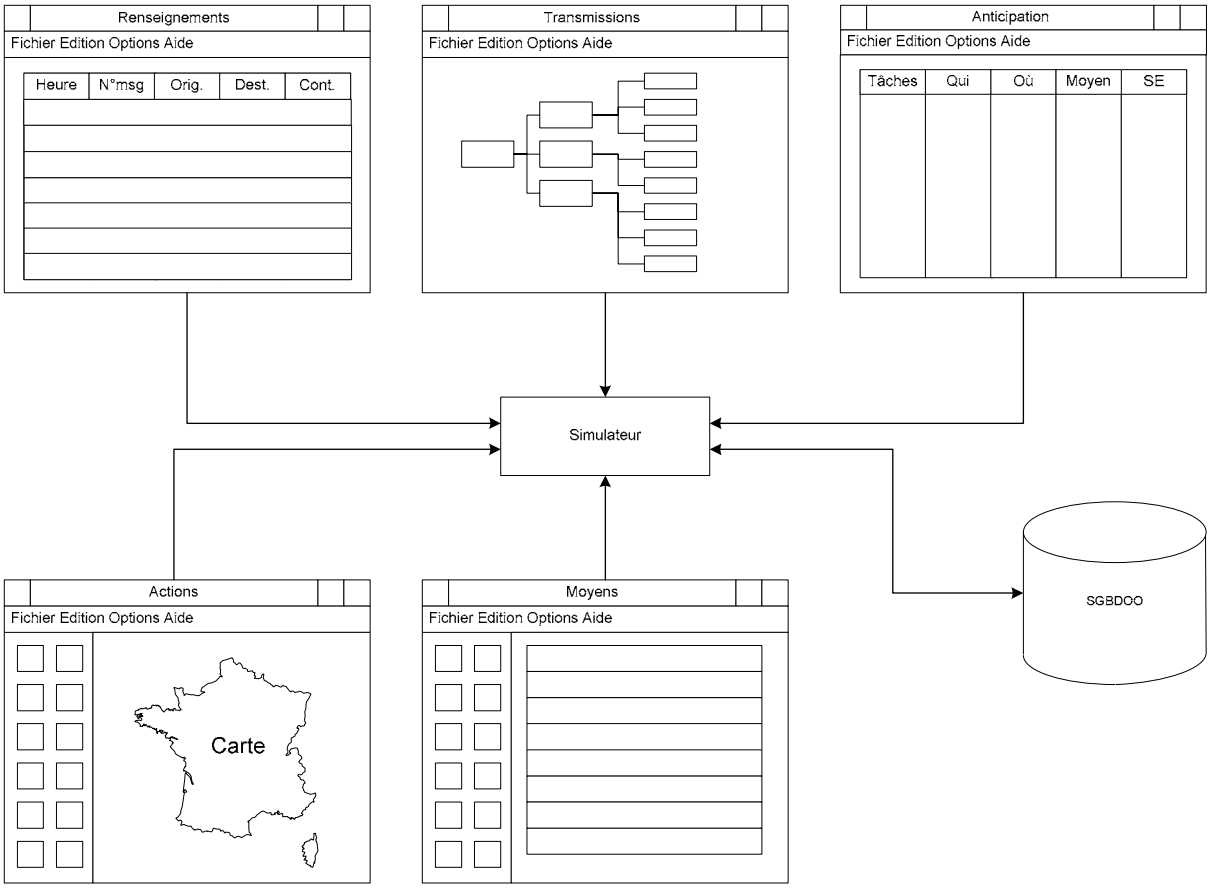
La majorité des actions réalisées par les stagiaires durant la simulation est mémorisée dans la base de données objets. Cet enregistrement a pour objectif un débriefing aisé pour le formateur. La durée d'une session de simulation est d'environ 2 heures. Chaque stagiaire prend la responsabilité d'un ou plusieurs rôles et peut être amené, pour cela, à utiliser différents modules logiciels (tableurs, calculs, cartographie vectorielle IGN, gestion des moyens opérationnels et gestion des communications). Le simulateur se charge de lire le scénario catastrophe de la base de données, de centraliser et de distribuer les événements. Il affiche en outre un synoptique en temps réel pour le chef instructeur, lui permettant de suivre précisément le déroulement du scénario en lui offrant une vision synthétique. L'exercice est

intégralement filmé en vidéo numérique au format MPEG, ce qui permet un accès précis et instantané grâce à une indexation réalisée par le formateur, pendant ou après la simulation.

Les capacités de réaction du simulateur n'étaient pas encore paramétrables dans cette phase du projet mais il était déjà possible d'introduire des alternatives dans le scénario. Il n'était pas possible de jouer des simulations de différents niveaux de difficulté. En fonction de degré de complexité demandé, le simulateur ne pouvait s'adapter de lui-même au niveau de l'élève. En revanche le formateur pouvait envoyer de nouveaux événements catastrophes en cours de simulation pour accroître la difficulté de la simulation tout en conservant le canevas imposé par le scénario originel. Ce fil conducteur existe sous la forme de contraintes de différentes formes (physique, temps, espace, …) que le simulateur devait respecter.

Le formateur peut aussi décider d'incarner un personnage intervenant dans le scénario comme un Préfet ou un Commandant des Opérations de Secours (COS). Si tel est le cas, il reçoit et traite les appels téléphoniques le concernant, sinon, le simulateur prend ce rôle en charge.

Le système est découpé par rôles et non pas par acteurs. Un officier peut prendre en charge éventuellement plusieurs rôles. Par exemple, dans le cadre d'une catastrophe de gravité modérée, le premier officier présent au PC peut prendre en charge plusieurs fonctions comme gérer les communications, gérer les moyens, anticiper l'évolution de la situation… Si la catastrophe s'accentue, le PC monte en puissance. Les communications se font de plus en plus nombreuses, les moyens impliqués sont aussi plus nombreux etc. Pour cela on augmente le nombre d'officiers et chacun prend un rôle déterminé. Le premier qui gérait simultanément plusieurs rôles va déléguer à d'autres officiers qui intègrent le PC. Chacun s'occupe alors d'un rôle particulier. Le PC peut contenir jusqu'à six personnes jouant chacun un rôle. Pour chacun de ces six rôles, nous avons développé un logiciel spécifique que nous allons présenter ci-dessous. Dans un souci pédagogique, chaque logiciel est relié au poste formateur par un système de visioconférence. Il est également possible au formateur de consulter à distance l'écran du stagiaire puisqu'il possède les mêmes logiciels mais en lecture seule.



Tous ces différents logiciels sont connectés à un serveur central sur lequel se situe une base de données contenant le scénario de la session de formation.

VIII.4.6. Le simulateur

Cette version d'ANTIC proposait un simulateur sur un serveur central. Ce simulateur fonctionnait sous deux modes : un mode purement statique qui n'exécutait que les événements prévus dans le scénario et un mode dynamique qui ajoutait un peu d'aléas dans le scénario. Ce mode dynamique était déjà conçu comme un système multi-acteurs (mais les acteurs étaient très statiques et n'exécutaient que des actions-reflexes. Ils ne communiquaient pas directement entre eux comme dans un SMA. Il comportait une interface graphique et un noyau de simulation qui gérait les « acteurs ».

Ce noyau travaillait sur un environnement régi par un automate cellulaire pour effectuer certains calculs de position. Un affichage graphique permettait de rendre compte de l'état de cet automate cellulaire et donc d'un aspect de la catastrophe. Le terrain était composé de cellules. Chaque cellule est référencée sur un plan par des coordonnées cartésiennes. Un véhicule pouvait se déplacer sur des zones particulières comme une route pour un camion, un fleuve pour un bateau etc. Une cellule sur le terrain possède des propriétés particulières telles que le passage d'une canalisation, d'une route, un état de pollution et des liens de cause à effet avec des phénomènes locaux pouvant faire changer l'état de la cellule. Chaque cellule comporte des informations sur sa surface (bâtiment commercial, industriel, forêt etc.) et possède un certain caractère d'inflammabilité.

Grâce aux caractères d'inflammabilité et en conjonction avec la météo, c'est à dire la direction et la force du vent, la présence de pluie ou non, un feu pouvait se propager d'une zone à une autre.

Le réalisme n'étant pas notre préoccupation majeure, la propagation se calculait grâce à une amélioration de « l'algorithme des proies » : Une zone comportant un caractère d'inflammabilité certain émet un signal dont l'intensité décroît proportionnellement à la distance entre elle et le prédateur (le feu). Les proies jouent ainsi un rôle d'attracteur pour le ou les prédateurs. La propagation à différentes cellules est une émergence de la combinaison de différentes réactions à des stimuli, le tout reposant sur l'automate cellulaire. Les besoins en fidélité varient selon le niveau de commandement auquel s'adresse la formation. Les officiers enfermés dans un PC n'ont pas la même vision du sinistre que les sapeurs en lutte sur le terrain. Il faut donc pouvoir simuler à différents niveaux de précision et d'abstraction.

D'autre part, il s'avère qu'aucun modèle physique de propagation de feu, de nuage nucléaire ou toxique, n'est actuellement entièrement satisfaisant. Il est donc parfois plus utile de recourir à des heuristiques ou des simplifications mieux dominées par le formateur. La simulation est actuellement basée sur un modèle minimal. Il faut choisir un niveau d'abstraction correct permettant de satisfaire d'une part les besoins de rapidité de développement, de rapidité d'exécution et d'autre part de précision de la réaction. Une base de connaissance pourra être greffée par la suite. L'état graphique de l'environnement (par exemple un nuage toxique se déplaçant) est une simple matrice de points correspondant à une image raster. Cette matrice de données doit être mise à jour à chaque top du timer. Le calcul peut donc être simple comme un jeu de la vie ou provenir d'un logiciel de calcul spécialisé dans le déplacement de nuages.

La modélisation par acteurs (ou agents) et par rôles permet de complexifier la simulation en ajoutant d'autres organismes d'Etat comme la gendarmerie ou le SAMU. Ces acteurs supplémentaires peuvent être des agents informatiques aux réponses prédéterminées ou de réels stagiaires.

VIII.4.7. Fonctionnement d'un scénario

Un scénario est construit sur une ontologie du domaine [Charlet & al., 1994]. L'ontologie dans ANTIC se découpe en trois parties interdépendantes. Ces parties décrivent donc l'ensemble des éléments du domaine, pouvant être statiques et dynamiques. Elles constituent un modèle en couches dans lequel elles sont interdépendantes.

Un scénario s'appuie sur des éléments généraux qui sont vrais quelque soit le type de sinistre, quelque soit sa géographie, etc. La description des éléments généraux revient donc à définir un méta-modèle dans le cadre duquel les scénarios seront développés. Ensuite, la séance gère l'ensemble des données concernant le déroulement du scénario lors de la simulation elle même. L'ontologie décrit et organise en détail les messages, les événements, les composantes du monde (terrain, moyens, organisations), le sinistre et le domaine didactique (MRT) [Lagadec, 1991].

Un scénario catastrophe telle une explosion de produits toxiques est entré dans une base du SGBDO Matisse. Il est basé sur une succession de séquences intervenant à une date, sur des réactions attendues des stagiaires, sur des conditionnelles et des contraintes. Ces relations temporelles se rapprochent de la logique temporelle de Allen [Allen, 1983] et [Bestougeff & Ligozat, 1989]. Le scénario est basé sur deux notions : l'événement et l'action :

Evenement(Id_Evt, Nom_Evt, Id_Act, Id_Acteur, Tps_Min, Tps_Max, Action_Echec);

Action(Id_Act, Pre_Cond, Post_Cond, durée, nom_methode, liste_parametres)

Un objet ou agent peut changer d'état dans le temps. Un événement conduit à un changement d'état de l'environnement mais c'est l'action qui agit sur l'environnement,

l'événement n'est que le déclencheur de l'action. Une action va se déclencher si toutes les préconditions sont présentes. La post-condition se réduit à un ou plusieurs envoi(s) de messages aux entités affectées par l'action présente.

Deux types de plans sont mis en œuvre :

- le plan du scénario se réduit à une suite partiellement ordonnée d'événements, c'est à dire un graphe dont chaque sommet est l'événement.
- le plan d'un acteur qui est une suite ordonnée d'actions séquentielles.

Exemple :

Un ordre est donné par le commandant C à un sapeur-pompier de commencer la manœuvre d'extinction du feu. Cela se traduit par les commandes

Evenement(Id_Evt, Nom_Evt, Id_Act, Id_Acteur, Tps_Min, Tps_Max, Action_Echec);
Action(Id_Act, Pre_Cond, Post_Cond, nom_methode, liste_parametres)

Evenement (Pompier attaque le feu)
Action (si qté eau > x et intensite feu < y et ... alors OK sinon Envoie Evt_Msg « impossible »)
Evenement(12^{e}, « eteindre feu », 45a, Feu, 0, 0, Replie()) ;
Action(45a, ..., ..., feu.diminue(param1, param2)) ;

Chaque entité devient un acteur qui gère le déclenchement de ses méthodes en fonction des événements déjà arrivés. Il gère donc une liste d'événements qui affecte ou non une variable booléenne. Lorsque toutes les conditions sont réunies, c'est à dire tous les booléens à VRAI, alors la méthode correspondante peut être appliquée.

Un événement en entraîne d'autres automatiquement. Cela constitue des réactions types enseignées dans la formation des pompiers. Par exemple, dans le cadre d'une alerte de type POI, le CTA doit renseigner le CODIS du départ des moyens Sapeurs-Pompiers vers le sinistre. Il s'agit d'une action réflexe.

Description des cascades d'événements :

Evt initial	***Evt***	***Description de l'événement***	***Agents impliqués***
1		Alerte : incendie hydrocarbure: message de l'entreprise vers le CTA	Entreprise, CTA
↳	1b	Mise en place des moyens POI de l'entreprise	Entreprise
↳	2	Départ des moyens POI des sapeurs pompiers alertés par le CTA	SP, CTA
↳	6	Le CODIS fait partir le PCS vers le sinistre	CODIS, PCS
↳	14	Le CODIS envoie un document à envoyer au CIRCOSC	CODIS, CIRCOSC
↳	16	Le commandant du CODIS est le COS	COS
↳	20	L'officier moyens du PCS ordonne au PT de constituer une réserve tactique	PCS, PT

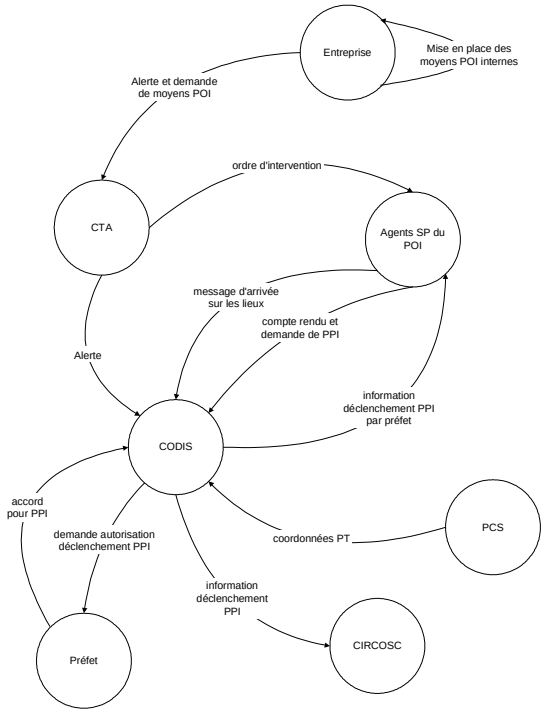
La figure ci-contre présente un diagramme de flux représentant la mise en place d'un PPI.

Les acteurs sont ici très simples (du type réactif). Considérons l'agent CTA, s'il reçoit une demande d'intervention des moyens SP pour un POI alors il a deux tâches à accomplir:

1. il envoie un message « ordre départ » à l'agent SP POI en donnant les coordonnées du sinistre en paramètre et celui-ci s'exécute. Les véhicules de secours se mettent en route.

2. il envoie un message d'information au CODIS.

Lorsque l'agent SP POI reçoit le message, il déclenche lui-même une série de réactions en chaîne etc.



VIII.4.8. Perspective de ANTIC I

Dans sa première version, le logiciel déroulait un scénario de façon statique, c'est à dire sans ajouter « d'intelligence » à la simulation. Le logiciel fonctionnait néanmoins de façon assez satisfaisante mais restait à l'état de prototype. Suite à une démonstration réalisée à l'INESC en fin de projet, le Ministère de l'Industrie a décidé de financer une suite à ce projet de simulation : le projet ANTIC II qui sera décrit dans les prochains paragraphes.

VIII.5. Le projet ANTIC II

Ce projet a débuté environ un an après la fin du projet ANTIC I. Il devait consolider ANTIC I qui était vu comme un prototype prouvant la faisabilité technique de notre simulateur de catastrophe. La deuxième phase avait donc pour objectif de mettre l'accent sur des réactions plus dynamiques du simulateur. Ce dernier devra adapter le scénario en fonction des moyens mis en œuvre par les stagiaires. Il devra aussi permettre dans l'avenir d'intégrer d'autres organismes d'Etats dans la simulation tels que la gendarmerie, la préfecture ou le SAMU. Ce deuxième projet sera plus orienté « terrain » et s'adressera aux officiers et sous-officiers du SDIS27 à Evreux.

Dans ce nouveau projet, un stagiaire est modélisé de la même manière qu'un agent purement informatique, c'est à dire comme une entité communicante et agissant sur son environnement. Chaque agent est responsable de son état. En fonction des perturbations qu'il reçoit de son environnement il modifie son état interne. C'est seulement quand un certain nombre de préconditions sont réunies, que l'agent va entreprendre une action ou va invoquer une méthode chargée de changer son apparence ou sa position. Chaque agent doit posséder une certaine autonomie décisionnelle. Notre problème devient typiquement multi-agents : La lecture du scénario catastrophe provoque l'instanciation de différents agents plus ou moins autonomes. A titre d'exemple, chaque individu d'un groupe d'intervention de la Sécurité Civile est modélisé par un agent, ils partagent des ressources comme de l'eau et se déplacent

dans un environnement qui est ici le lieu de la catastrophe modélisée. Chaque agent possède un (ou plusieurs) but(s) : maîtriser le sinistre, soigner des blessés, informer ou désinformer la presse. Ils peuvent s'associer pour lutter contre le sinistre ou peuvent entrer en conflit. Les différents agents de la simulation peuvent exploiter des ressources de l'environnement.

Le passage au « tout agent » pose de nouvelles problématiques telles que le coût des communications engendré par la distribution massive. La méthodologie d'analyse est également radicalement différente puisque orientée agent. Ce domaine à lui seul est encore à l'état de recherche.

VIII.5.1. Les partenaires du projet

ANTIC II perdait deux des partenaires de ANTIC I. Il fut donc conduit par :

- l'INSA de Rouen en partenariat avec la société ARAKNYDE, en charge de la dynamique du modèle
- A6 MediaGuide en charge du scénario pédagogique et de l'interface graphique
- Le SDIS27, dirigé par le Colonel Berthelot et encadré du Capitaine Jacques et du Lieutenant Hardy, en charge de l'expertise technique.

VIII.5.2. Solutions techniques retenues

Le projet est découplé en trois grandes parties : le noyau de simulation, l'IHM et le scénario. L'INSA et la société ARAKNYDE disposaient d'un modèle multi-agents (YAMAM) et d'une plate-forme implémentant ce modèle (Phoenix). La société A6 disposait du logiciel d'enseignement SERPOLET présenté ci-dessous :

SERPOLET (Système d'Enseignement et de Recyclage Par Ordinateur Liant Expertises et Technologies) est le nom d'un générateur de didacticiels puissant qui peut être couplé à une plate-forme d'enseignement du même nom afin d'optimiser ses potentialités. Ce logiciel auteur est constitué de deux applications.

- *SERPOLET Auteur :* Ce système permet la création et l'adaptation de modules pédagogiques. Il est organisé par une interface, l'Intégrateur De Connaissances, qui contient trois éditeurs.
- *SERPOLET Guide :* Ce sous-système permet le déroulement de la formation. Les modules se succèdent un à un. Le Guide appelle les liens vers les briques didactiques, et contrôle le respect des liens prévus par l'auteur au niveau de la notion, du module et de la formation. Il donne également des informations sur l'état du cursus et du module : celui-ci peut être « non connu », « en cours », « connu », « à revoir » ou « déjà vu » (pour les modules de type diaporama informatif). Le formateur et l'apprenant peuvent accéder au livret électronique qui rassemble des informations sur l'étudiant : description administrative, pré acquis, cursus en cours, objectifs, données pratiques (coordonnées du tuteur…), retours et correspondance échangés sur le campus, données pédagogiques (informations réservées enseignant). Ces données seront cryptées à terme. Elles sont stockées sur le serveur et en miroir sur le poste de l'apprenant. Grâce à des filtres sur l'écran, la souris, le clavier, des capteurs de zones sensibles et de temps, le système Guide peut récolter des informations sur le déroulement des cours (résultats d'évaluation, problèmes, nombre d'erreurs, interruption, durée…) qu'il restitue dans une interface Administrateur à part. Ce système de retours pédagogiques fait l'objet d'un brevet international. Ces informations sont éventuellement transmises au système Suivi (sous-système de la plate-forme d'enseignement) pour le traitement.

Phoenix, écrit en JAVA, et Serpolet, écrit en C++, n'étaient pas conçus pour communiquer ensemble. Les différents partenaires ont du apporter quelques modifications sur chacun de leurs outils logiciels pour leur permettre de communiquer. Une solution fut proposée au format XML.

VIII.5.3. Utilisation du modèle YAMAM pour le projet ANTIC2

Notre modèle YAMAM est bien entendu particulièrement adapté au projet ANTIC dans la mesure où il a été conçu originellement dans ce cadre. Nous avons vu dans ANTIC1 que le PC et le CODIS comprennent des officiers et que chaque officier a une fonction bien déterminée : un des officiers prend en charge les communication, un autre s'occupe des moyens, un autre tente d'anticiper l'évolution du sinistre etc. A un homme correspond donc une fonction.

Lors d'une intervention classée d'importance faible ou moyenne, un officier peut remplacer un ou plusieurs de ses collègues et prendre en charge plusieurs actions comme la gestion à la fois des moyens et de l'anticipation. Cela nous a donc amené à découpler l'agent de sa fonction, c'est-à-dire de son rôle. Un officier Sapeur-Pompier peut donc prendre en charge plusieurs rôles. Il y a la notre découplage entre l'entité Agent et l'entité Rôle. En fonction de l'évolution de la catastrophe, le PC ou le CODIS peuvent monter en charge et accueillir davantage d'officiers. Il s'agit donc d'une redistribution des rôles. Un agent prend en charge dynamiquement un nouveau rôle.

Chaque rôle requiert l'exécution de plusieurs actions qu'on appelle les tâches. A titre d'exemple, la gestion des communications peut se diviser en plusieurs actions éventuellement récurrentes comme :

- surveiller l'arrivée d'un fax
- surveiller l'arrivée d'un message radio
- rendre compte au CODIS
- alerter le CIRCOSC
- alerter le préfet
- classer les communications et en faire une synthèse sur un tableau…

Nous retrouvons là la décomposition des rôles en tâches de notre modèle d'organisation YAMAM.

Enfin, chaque action requiert une compétence particulière. L'envoi d'un fax requiert la connaissance de l'appareil, conduire un camion requiert aussi la compétence adéquate etc. Cela est vrai aussi bien pour les Sapeurs-Pompiers « réels » que pour les Sapeurs-Pompiers « virtuels ». Dans notre simulation d'un feu d'entrepôt, un agent informatique effectue une reconnaissance du terrain dès l'arrivée sur les lieux. Cette reconnaissance consiste à longer les murs et à recenser les positions de toutes les portes et fenêtres. L'algorithme qui s'occupe de cela est contenu dans une classe compétence « SkillReco ». Nous voyons bien ici l'articulation entre l'entité tâche et l'entité compétence, que l'on retrouve dans notre modèle YAMAM.

Enfin la notion de dynamique est prise en compte. Un rôle pris en charge par un officier peut être délégué à un autre officier dès l'arrivée de celui-ci dans la simulation. Lorsqu'un officier supérieur arrive sur les lieux, c'est lui qui prend le rôle de commandant des opérations. Il y a donc une prise en charge dynamique des rôles. YAMAM autorise cette dynamique grâce à sa gestion des agents découplée des rôles.

L'utilisation des mémoires de rôles de la plate-forme Phoenix permet aussi à un agent ayant pris en charge un rôle de sauvegarder certaines informations sur le contexte. Par exemple, le feu s'est déclenché à l'heure HH:MM et la cuve d'hydrocarbure est montée à XX degrés Celcius. Lors de la passation de son rôle, il peut transmettre aussi ses informations au nouvel agent qui endosse le rôle. Certaines plates-formes SMA ne savent transmettre facilement ce genre d'information dans la mesure ou un changement de rôle nécessite de tuer le premier agent pour en lancer un second.

Le projet ANTIC étant un jeu de rôles pour la formation, la présence de chacun des joueurs se révèle très importante. Dans la pratique, il est difficile de rassembler tous les participants. Le simulateur devait donc pouvoir aussi simuler un stagiaire officier absent dans la simulation et le remplacer par un joueur artificiel, connaissant au préalable toutes les actions à mener. Nous avons donc ajouté dans les scénarios de catastrophe des instructions « actions attendues » qui permettent non seulement de remplacer le joueur mais aussi de vérifier qu'il a bien accompli toutes les actions indispensables à la tenue de son rôle. Cela est indispensable pour l'instructeur qui va ensuite réaliser son débriefing. Le modèle YAMAM offre la possibilité de se focaliser sur un rôle et non sur l'agent qui l'exécute. Un même rôle peut alors être joué par un agent réel ou un agent virtuel et même passer de l'un à l'autre dynamiquement.

Cette dynamique du modèle permet d'introduire aussi facilement des événements nouveaux en cours de simulation. L'instructeur peut décider que les stagiaires gèrent trop aisément la situation et complexifier la simulation en ajoutant de nouveaux éléments comme retarder un véhicule ou déclencher un autre sinistre sur un autre site.

VIII.5.4. Choix d'un scénario

La formation au commandement chez les Sapeurs-Pompiers s'effectue sur trois types de scénarii en formation traditionnelle :

- Feu : d'habitation, industriel, de forêt
- Secours à personnes : accident de transport collectif, feu industriel avec dispersion de nuage toxique par exemple…
- Atteinte à l'environnement : inondation, pollution… (moins intéressant pour ANTIC 2 car il s'agit dans ces cas de se projeter dans le temps et pas forcément d'agir dans l'urgence).

Le feu industriel a été retenu pour être simulé dans le projet. Le SDIS27 a proposé de s'inspirer d'un sinistre existant pour lesquels ils disposaient de tous les documents (plans, enquête…). De plus, ce cas très grave était un sujet assez riche pour une formation car il a fait intervenir de nombreuses ressources humaines et matérielles. Les conditions de la simulation et du jeu par les stagiaires ont été définies d'après l'expertise métier des sapeurs-pompiers du SDIS 27.

Le SDIS dispose pour les zones à risque de plans de prévention. Des plans parcellaires indiquent les ressources en eau et les établissements sensibles. Pour ceux-ci, des plans d'établissements répertoriés apportent des photos et d'autres informations pour accélérer le travail en cas de sinistre. Lors de l'appel de signalement du sinistre, les informations données par la personne et le plan d'établissement répertorié induisent un « départ type » avec dimensionnement des moyens. Un logiciel d'alerte (*Geoconcept*, le plus répandu) associe une cartographie spécifique à des cartes IGN.

VIII.5.5. Mode d'utilisation

L'application ANTIC II peut s'utiliser durant une formation :

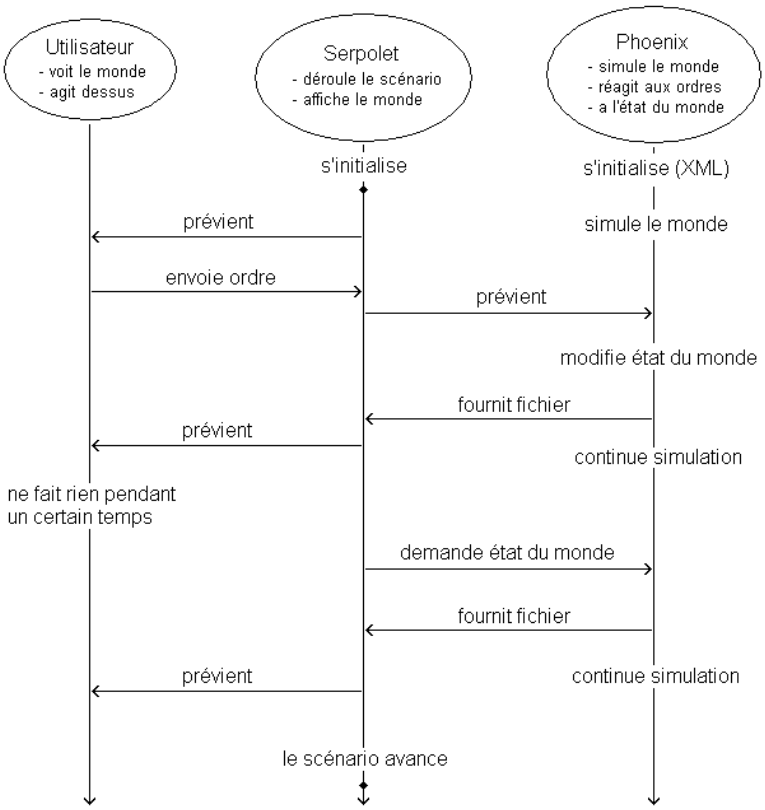
- en groupe, par la projection du scénario avec résolution du problème par une personne (la prise de décision individuelle fait partie de la formation du COS) et échanges du groupe ; Il peut être intéressant d'avoir des scénarios évolutifs qui permettent à plusieurs COS de se succéder.
- individuelle : elle peut constituer une étape de plus entre la formation théorique et la formation pratique mais ne les remplace pas. Elle peut également être utilisée comme entraînement pour les personnels déjà en place dans des fonctions de COS.

VIII.5.6. Architecture de communication entre Serpolet et Phoenix

Nous avons retenu une architecture MVC (Modèle-Vue-Contrôleur) et une communication par XML. Serpolet, en tant que « maître du temps », déroule le scénario. Lorsqu'il doit rafraîchir la Vue (*vue V*), il demande à Phoenix de lui fournir la représentation actuelle du monde (*modèle M*). De plus, il écoute les événements (*contrôleur C*) de l'utilisateur afin de passer les ordres à Phoenix. En fait, il gère, sur un modèle MVC, un ensemble d'instructions (une vingtaine au maximum) du type :

- « si l'utilisateur donne un ordre alors transmettre cet ordre à Phoenix. »
- « si le monde de Phoenix a changé alors réafficher »
- « si le sinistre est terminé alors la séance est terminée »

Phoenix, pour sa part, s'initialise à partir d'un fichier XML. Ce fichier est pour l'instant généré avant le début de la simulation et contient les renseignements nécessaires (nombre d'agents présents, nombre d'agents disponibles, …). Ensuite, il fait tourner le monde en permanence, seulement interrompu pour tenir compte des ordres de l'utilisateur et quand Serpolet demande l'état du monde. Il fournit alors une représentation de celui-ci dans un fichier lisible par Serpolet. Voici le schéma récapitulatif de cette architecture.



Phoenix offre de nombreuses possibilités techniques comme la distribution d'objets sur réseau via CORBA et l'utilisation des WebServices. Le projet ANTIC2 ne s'exécutait que sur une seule machine. Dans un souci de rapidité d'exécution, nous avons donc désactivé les appels à CORBA au profit d'appels par référence. De même, les Web-services servent à proposer le téléchargement de rôles, de tâches ou de compétences depuis le serveur sur la machine locale. Le principal intérêt de cette fonctionnalité est de permettre un développement réparti des classes Phoenix tout en offrant un système de diffusion du code source créé. Dans le cas de l'application ANTIC II, toutes les compétences sont déjà enregistrées sur la machine. Nous avons donc également désactivé la fonctionnalité WebServices.

VIII.5.7. Implémentation

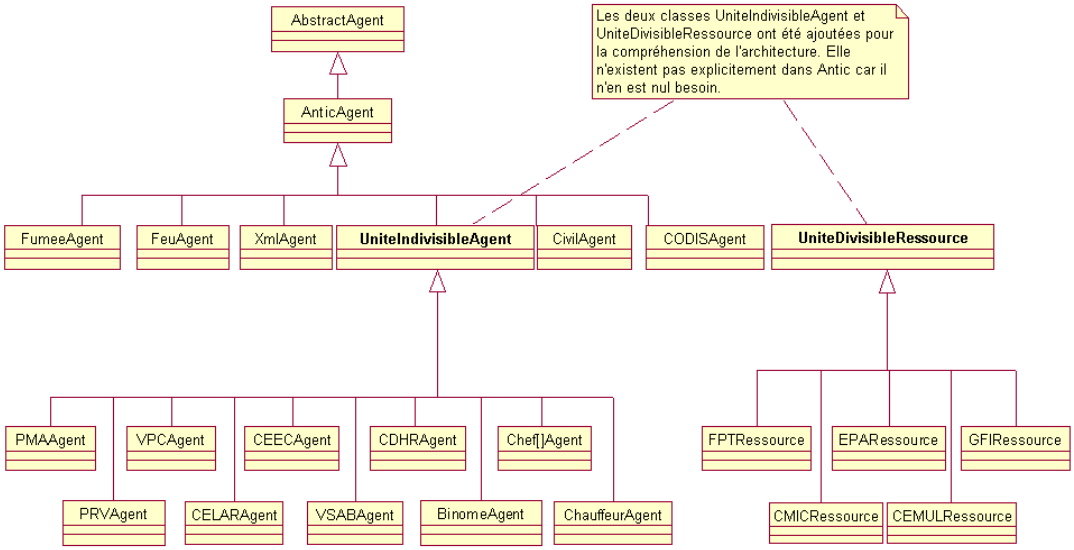
L'ontologie définie dans le projet nous a permis de réifier le modèle selon les règles suivantes :

- Toute entité devant agir (grâce à ses compétences) est un agent.
- Tous les comportements sont définis dans le code inhérent à une compétence.
- Toute entité contenant des renseignements déterminant la façon d'agir est une ressource.
- Les agents et les ressources se trouvent dans l'environnement s'ils représentent une entité physique du monde réel.

VIII.5.7.1. Les agents :

La classe *AnticAgent* hérite directement de la classe *AbstractAgent*. Nous avons créé plusieurs agents, dont le niveau de commandement est différent. Nous avons par exemple un agent « Chef de Groupe » (*ChefGroupeAgent*) qui commande 2 à 4 véhicules. Nous avons également un agent « Chef de Colonne » (*ChefColonneAgent*) qui gère 2 à 4 groupes, ainsi qu'un agent « Chef de site » (*ChefSiteAgent*) qui peut diriger le tout. *L*es agrès, représentés

par des agents, sont des unités indivisibles. Alors qu'un Fourgon Pompe Tonne (FPT) est divisé dans l'application en 2 binômes, les unités suivantes représentent les entités qui agiront effectivement. Lorsque l'utilisateur, qui joue le rôle du COS, veut lancer un ordre, c'est à ces types d'unités qu'il s'adresse.

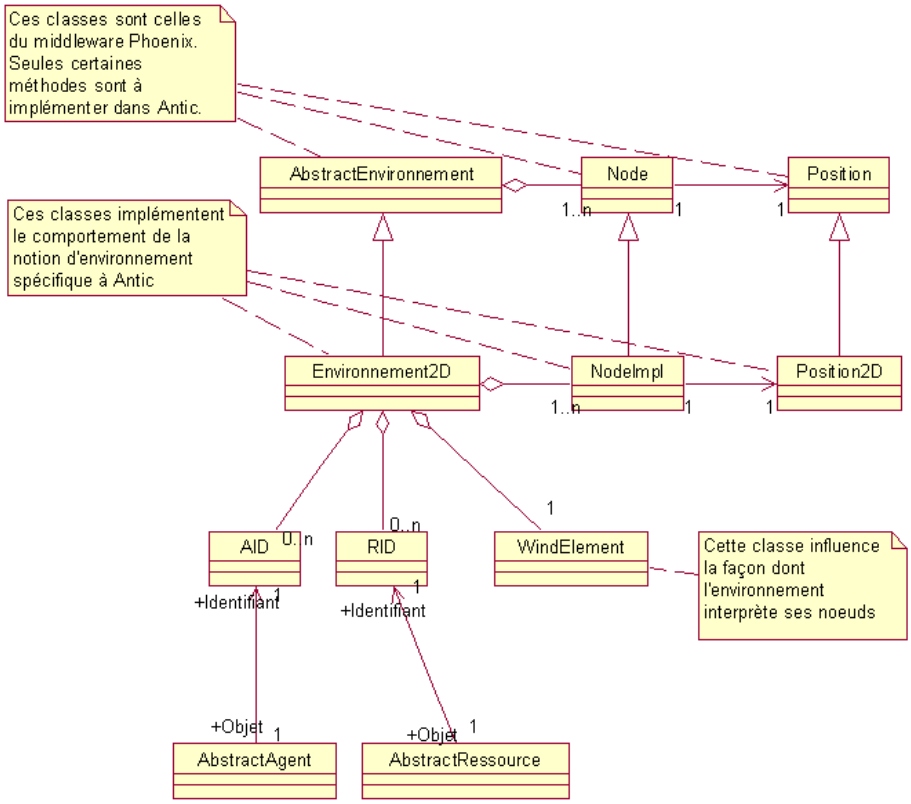


Nous avons ensuite utilisé d'autres agents gérant les civils, le CODIS, le feu, la fumée… Chacun utilise des compétences métiers. D'autres types d'agents sont dédiés à la gestion du système comme XmlAgent qui gère les transferts de messages XML entre Phoenix et Serpolet.

VIII.5.7.2. Description de l'environnement

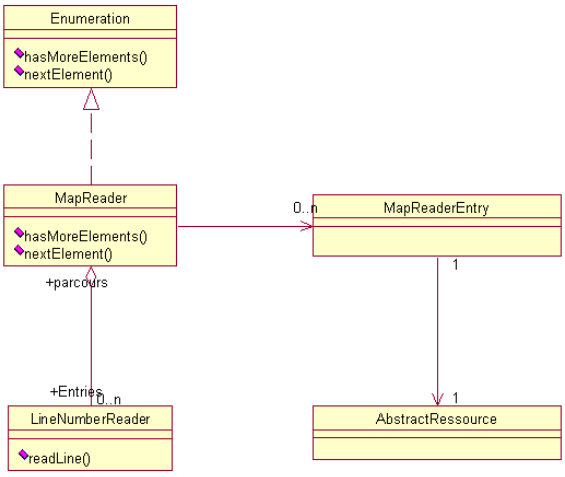
Les agents décrits ci-dessus ainsi que les ressources se situent dans un environnement. Phoenix propose un modèle d'environnement. C'est la classe abstraite *AbstractEnvironnement*. Une classe abstraite possède des méthodes qui doivent être implémentées dans la (les) classe(s) fille(s).

Nous avons choisi d'instancier la classe *Environnement2D* qui dérive de *AbstractEnvironnement*. Elle représente un environnement en deux dimensions. Nous pouvons toutefois imaginer la construction d'un environnement en trois dimensions. Pour l'instant, l'environnement est donc composé de cellules (c'est la classe « *NodeImpl* ») disposées en « l » colonnes sur « h » lignes. A chaque cellule est donc associée une position indiquant l'abscisse et l'ordonnée.



Cet environnement implémente différents mécanismes de gestion des agents et des ressources et peut être stocké en format XML :

- ajout d'un agent (d'une ressource) à une position aléatoire ou fixée
- déplacement d'un agent (d'une ressource) d'une position à une autre
- retrait d'un agent (d'une ressource) de l'environnement.
- gestion de la liste des agents (des ressources) présents
- gestion du paramètre vent (changement de direction ou de force)
- listage des nœuds contenant des informations devant être affichées.
- obtention d'un nœud à partir de sa position
- obtention des nœuds voisins d'un nœud donné (avec ou sans vent)
- obtention des nœuds visibles depuis un nœud donné sur une distance donnée
- obtention des agents visibles depuis un nœud donné sur une distance donnée
- obtention des ressources visibles depuis un nœud donné sur une distance donnée



Le découpage en lignes et colonnes est obtenu à partir d'une carte qui est fournie lors de la définition du scénario. Pour notre part, nous nous sommes basés sur la carte fournie par le SDIS27. Cette carte mesure 150m de largeur et 120m de hauteur. Notre environnement, découpé en cellules de 1m sur 1m, possède donc 150*120 nœuds (autre terme pour cellules), soit plus de 15.000 nœuds. Cette mesure de 1m a été choisie car elle permet au sinistre d'être simulé de façon réaliste sans pour autant être trop pesante pour les calculs.

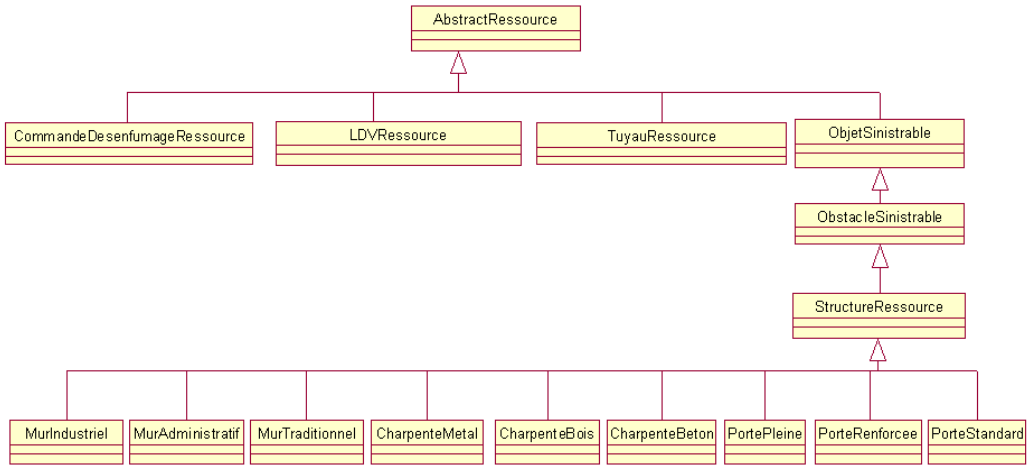
L'utilisateur peut modifier entièrement sa carte en mettant les ressources qu'il veut où il veut. La construction de la carte dans le simulateur se fait à partir d'un fichier texte. L'interpréteur, de la classe *MapReader*, retourne une à une chacune des ressources qu'il lit dans le fichier. Implémentant l'interface *Enumeration*, il permet une construction progressive de l'environnement. Cette carte peut contenir des cases vides, bien évidemment, et des cases contenant des ressources.

VIII.5.7.3. Description des ressources

Les ressources contiennent des informations. Il existe différents types d'informations, que nous avons classées en trois catégories :

❶ Les bâtiments

On décrit leur position, leur type et leur résistance au feu. *C*es ressources héritent de « *ObstacleSinistrableRessource* » car elles peuvent contribuer à la propagation du feu ou de la fumée.



❷ Les unités sapeurs pompiers

Ces ressources influent sur la propagation du sinistre. Le modèle de propagation défini est assez simple : pour commencer, l'agent feu possède une liste de nœuds enflammés, l'agent fumée possède une liste de nœuds enfumés. Pour l'un comme pour l'autre, la propagation consiste à parcourir cette liste, traiter chacun des nœuds jusqu'au dernier et recommencer. Nous avons introduit plusieurs variables telles qu'un « degré d'inflammabilité », une intensité, un pouvoir fumigène, …

Parmi les ressources utilisées, *notons CEMULRessource, CMICRessource, EPARessource, FPTRessource et GFIRessource*. Il s'agit de moyens de Sapeurs-Pompiers comme des Fourgons Pompe Tonne, des échelles EPA etc.

Toutes ces ressources, ainsi que les agents présentés ci-dessus, sont créés par une classe utilitaire appelée « *AgentInstancer* » qui agence les agents / Ressources comme il le faut. Il est important de noter que les agents sont des unités indivisibles alors que les ressources, au contraire, sont un regroupement d'unités indivisibles.

❸ Les autres

De même que certains agents avaient été créés individuellement, pour la diversité du logiciel, certaines ressources ont été implémentées pour le réalisme de la simulation comme des commandes de désenfumage, des lances à débit variable, des tuyaux…

Toutes ces ressources sont des informations manipulées par les agents lors de l'exécution d'un rôle. Une autre source d'information existe dans le modèle suivi par Phœnix : la mémoire des agents. Toutefois, cette mémoire est volatile. C'est à dire que lorsqu'un rôle est terminé, la mémoire est effacée (sauf dans certains cas). Il est donc possible de stocker des renseignements de façon éphémère dans la « mémoire » de l'agent. Ce mécanisme est utilisé pour la configuration d'un agent avant l'exécution d'un rôle afin de définir une coordonnée à atteindre, un civil à rechercher, …

Nous allons maintenant présenter l'ensemble des rôles qui ont été développés pour ANTIC. Nous présenterons ensuite la liste de compétences disponibles en indiquant quels sont les renseignements (stockés dans la mémoire) nécessaires pour leur exécution.

VIII.5.7.4. Description des rôles

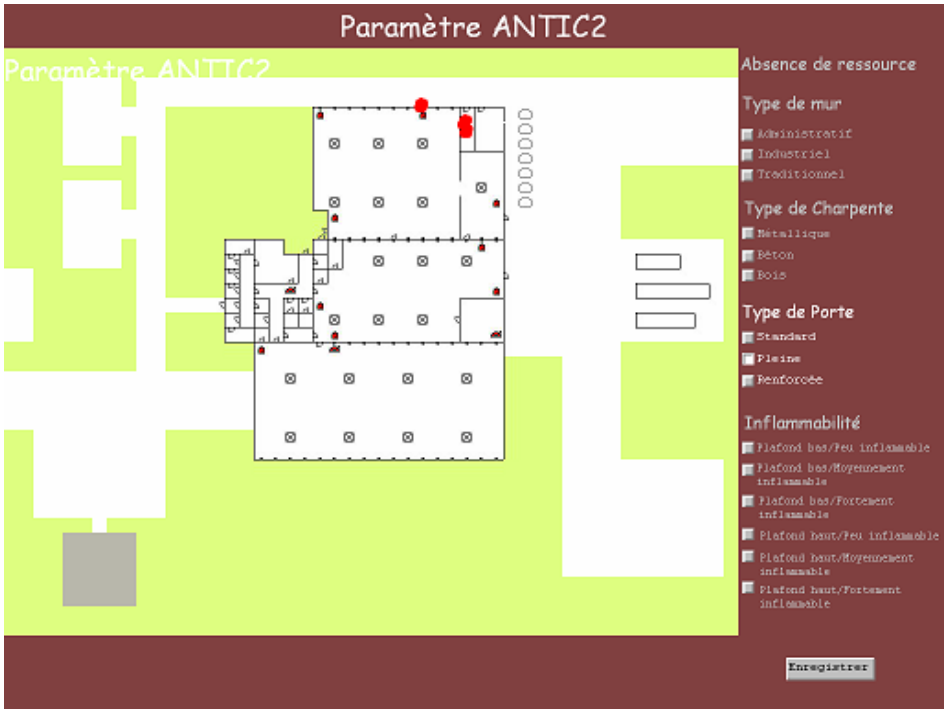
Les rôles disponibles sont de deux types : les premiers constituent l'ensemble des ordres que l'apprenant peut donner aux unités Sapeurs-Pompiers disponibles et les seconds sont les rôles non disponibles pour l'apprenant. Ils sont exécutés soit par l'agent représentant le CODIS, soit par les agents qui servent à faire tourner la plate forme. Parmi les rôles créés, notons *Propager_feu, Propager_fumée, Suivre_agent, Instancier_moyens, Communiquer, Photographier*…

Ces différents rôles sont composés de 1 à n tâches consécutives que nous n'allons pas décrire dans ce document.

VIII.5.8. Fonctionnement du logiciel

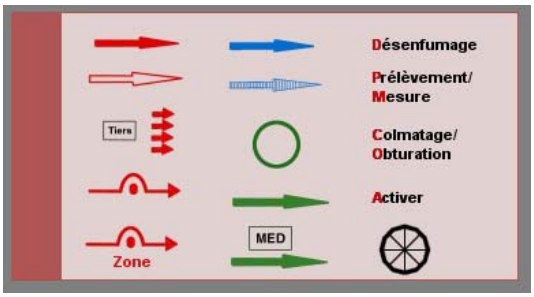
L'instructeur initialise le logiciel en spécifiant les conditions initiales de la simulation, la carte et le niveau de commandement spécifiques au stagiaire. Cette phase d'initialisation est composée de quelques écrans de paramétrages à la manière d'un « Wizard » sous Windows. On détermine ainsi des paramètres comme la force du vent, la pression à chaque poteau

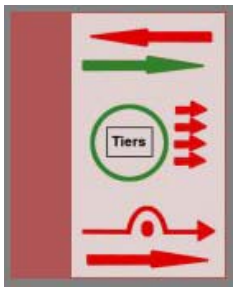
incendie, l'acheminement des renforts, les types de murs, de charpentes et de portes, le droit à l'aide contextuelle etc.



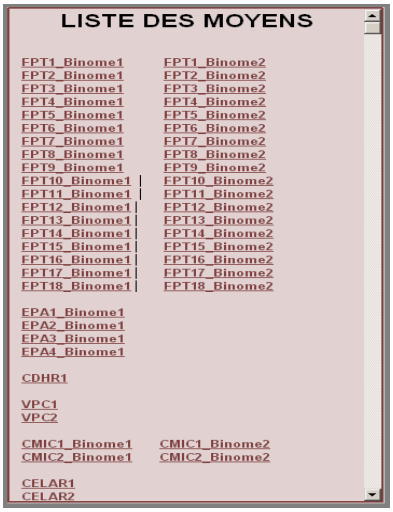
La phase de simulation débute par une demande de Login pour charger les fichiers de configuration adéquats. Le scénario par défaut se déroule jusqu'à la prise de commandement des opérations de secours par le stagiaire. Si l'utilisateur entre dans la simulation en tant que chef de groupe, le départ du sinistre et les premiers secours seront simulés normalement. S'il entre en tant que chef de colonne, le démarrage de la catastrophe, les premiers secours et le commandement par le chef de groupe sont simulés en accéléré afin de diminuer le temps d'attente du stagiaire. Une fois qu'il a pris le commandement, la simulation reprend une vitesse normale et c'est au stagiaire d'opérer. Ses actions possibles sont de lancer une action, choisir un moyen ou envoyer un message. Il peut néanmoins ne rien faire et regarder l'évolution de la situation.

Pour lancer une action, il doit choisir une « action » ou une « action combinée » parmi la liste des actions possibles. Ces choix sont accessibles via une palette d'outils graphiques.





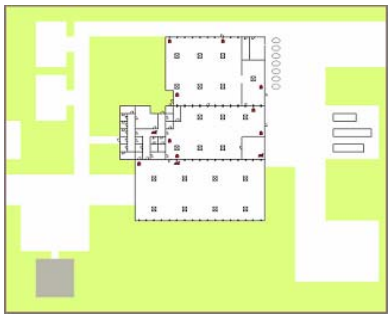
Le stagiaire peut sélectionner un « moyen » pour accomplir une action. Il spécifie ensuite les renseignements nécessaires à la bonne exécution du rôle. Ces moyens sont accessibles via une fenêtre déroulante (figure ci-dessous). Le stagiaire peut également envoyer un message au CODIS comme une demande de renfort, une aide EDF/GDF, une demande au préfet ou aux forces de l'ordre etc.

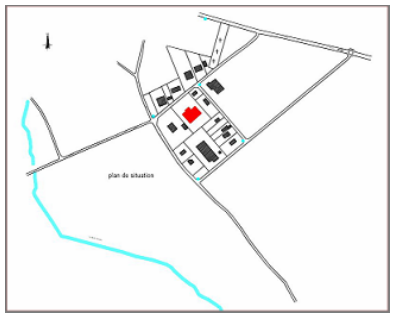


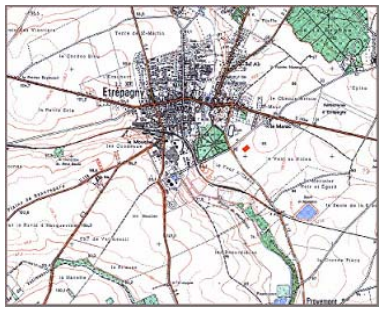


Les demandes de renforts permettent de demander tous types d'agrès de Sapeurs-Pompiers. Elles sont spécifiées dans une fenêtre graphique offrant un choix multiple dans une sélection comportant un chef de colonne, un chef de site, un FPT, un EPA, un CEMUL…

Si le stagiaire ne sélectionne aucun des choix précédents, il laisse la situation évoluer. Serpolet demande régulièrement à Phoenix l'état du monde afin de mettre à jour l'interface qui est composée de trois niveaux de précision accessible à l'apprenant. La première est un plan de masse de l'entreprise qui permet au stagiaire de lancer toutes ses actions. La deuxième est un plan de situation qui permet de spécifier que l'alimentation en eau provient de la rivière. Enfin la troisième est un plan général qui sert à surveiller les alentours de la zone d'intervention.







La simulation prend fin si l'apprenant passe le commandement à son supérieur (chef de colonne ou chef de site suivant qu'il soit chef de groupe ou chef de colonne) ou si le scénario se termine (le sinistre a tout brûlé ou est éteint).

VIII.6. Conclusion

Le projet ANTIC est un démonstrateur montrant la faisabilité technique d'un projet de simulation distribuée pour les officiers Sapeurs-Pompiers. Ce projet donnera probablement lieu à une suite, s'adressant plus particulièrement aux officiers terrain. Cette nouvelle phase devrait requérir des représentations graphiques en 3D, proches de celles utilisées à l'ENIB sur le projet AREVI/ORIS [Querrec & al., 2001]. Dans sa deuxième phase, ANTIC a aussi eu pour intérêt de tester notre middleware Phoenix et d'obtenir un premier retour d'expérience sur son exploitation dans un projet industriel. Ce projet nous a permis de vérifier sa scalabilité que ne pouvait mettre en exergue les quelques toy-problems que nous avons présentés dans le précédent chapitre.

La simulation consiste, d'une manière générale, à faire évoluer une abstraction d'un système au cours du temps afin d'aider à comprendre le fonctionnement et le comportement de ce système. Cela permet aussi d'appréhender certaines de ses caractéristiques dynamiques dans l'objectif d'évaluer certaines décisions [Hill, 2002]. La technologie agents se révèle souvent bien appropriée pour modéliser de telles applications.

Nous avons introduit dans ANTIC une réelle approche multi-agents qui tend à prouver, d'une manière générale, l'intérêt des SMA pour la simulation. Des recherches dans des domaines connexes ont aboutis aux mêmes conclusions.

Le domaine du transport possède aussi certaines problématiques auxquelles les SMA peuvent apporter des briques de réponses. L'institut National de Recherche sur les Transports et leur Sécurité (INRETS) travaille sur les systèmes d'aide à la décision et sur la gestion d'une ligne de bus, en particulier la détection d'incidents, le diagnostic et la prise de décision [Pinson & Balbo, 2001]. Un autre projet de l'INRETS est ArchiSim qui projette de modéliser la conduite automobile et de simuler le trafic routier [ElHadouaj & al., 2000].

Dans la simulation urbaine, le concept d'agent a été utilisé pour simuler des comportement interactifs des éléments de système urbain : maisons, piétons, véhicules… Une des applications du projet Nexsus[66] [Metcalfe & al., 2000] consiste à simuler la croissance d'une ville dans l'espace et dans le temps. De même, le centre CASA[67] (Centre for Advanced Spatial Analysis) est une initiative de l'Université de Londres pour développer des technologies informatiques dans plusieurs disciplines comme la géographie, l'espace, la construction d'environnement…

Le projet MACSI propose une approche par les SMA pour la simulation de réseaux de télécommunications [Merghem & Gaïti, 2002]. C'est typiquement un domaine dans lequel la complexité et la dynamicité ne cessent de croître, ce qui rend les techniques de modélisation traditionnelles de moins en moins appropriées. L'apport du SMA offre une amélioration des performances globales en garantissant un équilibre dynamique entre les nœuds du réseau.

[66] Le site du projet Nexus est accessible sur www.nexsus.org
[67] Le site du projet CASA est accessible sur http://www.casa.ucl.ac.uk/

Conclusion et perspectives

« Rien n'est plus dangereux qu'une idée, quand on n'a qu'une idée. » - Alain (Emile Chartier, dit)

Bilan

Les SMA ne sont pas seulement une nouvelle façon de concevoir et de développer des applications. Ils changent radicalement la manière d'appréhender l'informatique en ajoutant les dimensions d'autonomie et de robustesse. Cette nouvelle approche requiert des méthodologies d'analyse et de conception différentes de celles utilisées actuellement dans le monde de la programmation par objets. Toutes ces nouvelles méthodes en amont de la phase de programmation sont encore à l'état de recherche et aucune n'est encore « universelle ». Elles varient selon le chercheur, ses convictions, ses domaines d'études etc. Pour une même analyse et une même conception, l'implémentation d'un SMA n'aboutira pas nécessairement aux mêmes résultats selon l'outil utilisé (la plate-forme). Cela est du aux choix technologiques pris par le programmeur de la plate-forme, à sa manière de programmer, de synchroniser les processus, de communiquer etc. Toutes ces contraintes font des SMA un domaine difficile à étudier, d'autant que la formalisation est souvent délicate. Nous nous sommes beaucoup intéressés aux divers travaux de la communauté des SMA pour définir nos propres orientations de recherche et aboutir à la réalisation effective d'un logiciel.

La problématique initiale entrait dans le cadre de la conception d'un simulateur pour lequel une solution basée sur les SMA s'est révélée bien adaptée. Après un état de l'art sur les diverses solutions en matière de simulation, nous nous sommes orientés vers les SMA qui présentaient des caractéristiques telle que la dynamique d'autorégulation. Une étude des outils existants en matière de SMA nous a conduit à développer notre propre modèle d'organisation et à l'implémenter sous la forme d'une nouvelle plate-forme générique d'agents mobiles. L'outil étant créé, nous avons conduit une phase de test en mettant au point quelques petites applications montrant quelques fonctionnalités de la plate-forme. Cette phase achevée, nous avons pu concevoir et développer notre application de simulation, laquelle nous a permis de recueillir un premier retour d'expérience, à plus grande échelle, sur notre outil. Le projet ANTIC a donné lieu à des démonstrations à chacune de ses étapes. Les utilisateurs finaux semblent satisfaits par l'état actuel du prototype. Nous devrons cependant poursuivre nos efforts pour finaliser le projet avec une version plus aboutie qui ne nécessite pas par exemple de phase de configuration complexe.

La plate-forme constitue non seulement une réponse opérationnelle à notre problématique de simulation mais offre aussi des caractéristiques la rendant utilisable dans d'autres domaines. Nous avons ajouté des fonctionnalités telles que l'ouverture et la communication sécurisée vers d'autres logiciels, des fonctionnalités de distributions de tâches, d'équilibrage de charge etc. La migration d'agents, leur spécialisation et leur redondance permet à la plate-forme d'acquérir une certaine autonomie. Cette approche est également nouvelle dans la mesure où le programmeur SMA ne se soucie plus de « qui fait quoi » mais se contente de développer le savoir-faire des agents et de confier des missions à une des plates-formes du réseau et non à un agent en particulier. Les agents se chargent eux-mêmes du choix et de l'accomplissement des missions grâce à un système d'échange

d'énergie. Notre travail se distingue des autres travaux de la communauté des SMA par l'auto-organisation induite par le système d'échange d'énergie couplée à la grande modularité de notre modèle d'organisation.

D'un point de vue personnel, ces travaux de recherche m'ont permis d'acquérir une certaine rigueur méthodologique et m'ont permis aussi d'approfondir mes connaissances sur les SMA, la modélisation logicielle et les technologies réseau. C'était à la fois un approfondissement technique mais aussi une réflexion générale sur l'épistémologie, notamment les sciences de l'organisation, la phénoménologie et les nouvelles voies vers une intelligence artificielle.

Perspectives

Les diverses fonctions que nous avons conçues et implémentées dans cette étude offrent aussi de nouvelles perspectives. Nous avons entrepris non seulement la construction d'un middleware multi-agents, mais également proposé une des premières approches d'une informatique totalement distribuée et « auto-régulée[68] ».

Ces systèmes sont auto-adaptatifs et auto-immunes. Ils sont réactifs à la charge et effectuent le reroutage de flux de données et de traitements automatiquement synchronisés… Ces avancées majeures dans le domaine de l'autonomie préfigurent un avenir dans lequel les technologies SMA auront un grand rôle à jouer. Notre middleware pourrait, avec quelques évolutions, se positionner comme un outil pour les projets d'Autonomic Computing et de Grid Computing.

Certains modules de notre outil actuel peuvent donner lieu à des améliorations. Je pense notamment au découplage des rôles en tâches et à la recherche de nouvelles compétences d'agent avec un moteur ontologique. Cela permettrait à un agent de rechercher une nouvelle compétence sur le réseau, uniquement par une description et sans connaître au préalable son nom. La thèse de Samuel Morisset va précisément dans ce sens. La mémoire de l'agent et son système décisionnel est également un point délicat. L'adjonction d'un réseau neuronal ou d'un système heuristique plus perfectionné permettrait aux agents cognitifs de ne pas effectuer plusieurs fois les mêmes erreurs. Cette mémoire, sous quelque forme que ce soit, est en cours de recherche mais présente un désavantage qui est sa complexité de mise en œuvre et le taux d'occupation du processeur. Nous avons pensé intégrer prochainement un système proche des BDI dans le module décisionnel. En outre, le système global d'échange d'énergie est perfectible. Il pourrait être amélioré avec des techniques d'optimisation comme la chimiométrie[69]. Nous n'avons pas eu l'opportunité d'effectuer une batterie de tests pour prouver statistiquement l'efficacité du produit, cette phase sera probablement réalisée à l'heure où j'écris ces lignes. Nous pensons aussi qu'une version graphique du middleware permettrait de lancer et de gérer les agents plus facilement. Cela fait actuellement l'objet de plusieurs stages ingénieurs au laboratoire de recherche.

Le projet YAMAM/Phoenix est en état de prototype opérationnel et nous devrons poursuivre nos efforts pour le fiabiliser et le déployer à grande échelle. Nous avons présenté

[68] Traduction la plus proche de l'autonomic computing

[69] La chimiométrie peut être définie comme la branche de la chimie qui utilise les outils mathématiques et statistiques pour construire et sélectionner les expériences de manière à en extraire le maximum d'informations grâce à une analyse des données pertinente. Par extension ces techniques ne sont pas seulement utilisables dans la chimie mais également dans tous les domaines qui pourraient nécessiter une optimisation.

dans ce rapport la conception de la version JAVA. Nous travaillons aujourd'hui sur une évolution en langage C#.NET qui devrait nous offrir de meilleures performances.

Toutes ces extensions prévues pour perfectionner ce produit suivront aussi l'évolution de la technologie. L'utilisation de nouveaux outils de développement permettra d'intégrer les prochains standards de communication.

Sans nous projeter davantage dans le futur, les SMA trouvent déjà des applications dans les télécommunications, le travail collaboratif assisté par ordinateur, la robotique, la simulation de systèmes complexes, les bases de données réparties, les systèmes multi-capteurs, le commerce électronique, la bioinformatique, la planification et la supervision, l'optimisation des problèmes spatiaux etc.

Toutes ces applications montrent à quel point les SMA peuvent trouver une place dans l'industrie logicielle de demain. Ce nouveau concept de modélisation et de programmation devrait s'imposer progressivement mais des efforts sur des méthodologies de conception restent à faire afin de guider le développeur dans l'élaboration de ses futures applications distribuées.

Malgré l'intérêt grandissant de la communauté scientifique envers les SMA, il est raisonnable de prendre en compte également les problèmes engendrés par la distribution de l'informatique. Ce que l'on gagne par un traitement des tâches en parallèle assuré par des agents intelligents, on le perd par la somme considérable des informations échangées entre agents. En outre le traitement massivement parallèle induit par les SMA montre la faiblesse des architectures matérielles actuelles. Le nombre de processus sur une machine est naturellement limité. Cependant de nouvelles puces électroniques verront probablement le jour pour palier cette complexité de traitement. A chaque problème sa solution…

Bibliographie

[ACIBM] site IBM de l'Autonomic Computing. http://www.research.ibm.com/autonomic/
[AgentBuilder, 2002] site web http://www.agentbuilder.com/AgentTools/index.html
[AgentCities, 2002] site web http://www.agentcities.org/Resources/Software/
[AgentLink, 2002] site web http://www.agentlink.org/
[Allen, 1983] Allen J., Maintening knowledge about temporal intervals. Communications of the ACM, Volume 26, 832-843, 1983.
[Angin & al., 1998] Angin O., Campbell A.T., Kounavis M.E., and Liao R.R.-F., "The Mobiware Toolkit: Programmable Support for Adaptive Mobile Netwoking", IEEE Personal Communications Magazine, Special Issue on Adapting to Network and Client Variability, invited paper, August 1998, http://comet.columbia.edu/mobiware/
[Aramira, 2002] Aramira Corporation - JumpingBean, www.jumpingbeans.com
[ASA, 2002] Architectures de Systèmes d'Agents. Site web du groupe ASA : http://www.grp-asa.fr.st/
[Ashby, 1952] Ashby W., Design for a Brain.Chapman and Hall, 1952.
[Atlan, 1979] Atlan H., Entre le cristal et la fumée, Seuil, Paris 1979.
[Attoui, 1997] Attoui A., Les systèmes multi-agents et le temps réel, Eyrolles, ISBN 2-212-08935-X, 1997.
[Autebert, 1987] J-M. Autebert, Langages algébriques, collection ERI. Masson Ed. ISBN : 2-225-81087-7.
[Auveriot, 2002] Auveriot O., REBOL – Programmation, Eyrolles, ISBN 2-212-11017-0, 2002.
[Baeijs, 1998] Baeijs C., Fonctionnalité émergente dans une société d'agents autonomes, PhD Thesis, Institut National Polytechnique de Grenoble, France, Novembre 1998.
[Barkai, 2001] Barkai D., Peer-to-Peer Computing, Intel Press, octobre 2001.
[Barr & Feigenbaum, 1981] Barr A. and Feigenbaum E., The Handbook of Artificial Intelligence. Pitman, 1981.
[Bauer & al., 2001] Bauer B., Müller J.P., Odell J., Agent UML: A Formalism for Specifying Multiagent Interaction », Agent-Oriented Software Engineering, Paolo Ciancarini and Michael Wooldridge eds., Springer, Berlin, pp. 91-103, 2001.
[Baumol & Benhabib, 1989] Baumol W. and Benhabib J., "Chaos: Significance, Mechanism, and Economic Applications, in Journal of Economic Perspectives, vol. 3, issue 1, pages 77-105, 1989.
[Beaune, 2001] Beaune P., Apprentissage Automatique dans les SMA, Juin 2001.
[Bellifemine & al., 2001] Bellifemine F., Poggi A. and Rimassa G., Developing multi agent systems with a FIPA-compliant agent framework, in: Software - Practice And Experience, no. 31, page 103-128. 2001. http://sharon.cselt.it/projects/jade/
[Benard & al., 2002] Bénard J-L., Bossavit L., Médina R. et Wiliams D., Extreme programming – avec deux études de cas - collection objet de Eyrolles– ISBN 2-212-11051-0, 2002.
[Benhouhou & Mathieu, 1992] Benhouhou A, Mathieu J., Un générateur automatique de scenarii basé sur un simulateur. Une approche orientée objet, Actes des journées EC2 sur la représentation par objets, MontPellier, juin 1992.
[Benhouhou, 1993] Benhouhou A., Un système de formation basé sur un simulateur et un générateur de scenarios", colloque international en informatique cognitive des organisations (ICO93), 29-36, Montréal, 4-7 mai 1993.
[Benhouhou, 1996] Benhouhou A., Une base de connaissances événementielle pour le guidage interactif des situations de crise, Thèse de doctorat, Université Pierre & Marie Curie, Paris VI, 1996.
[Bensaid & Mathieu, 1996] Bensaid N. et Mathieu P., Intégration du raisonnement hypothétique dans une architecture multi-agents hiérarchique. Journées Jeunes Chercheurs, Ganymède, pp. 2-11, 1996.
[Bergenti & Poggi, 2000] Bergenti F. and Poggi A., Exploiting UML in the design of multi-agent systems - In A. Omicidi, R. Tolksdorf, F. Zambonelli, eds., Engineering Societies in the Agents World - LNAI 1972, pp 106-113, 2000.
[Bernon & al., 2001] Bernon C., Camps V., Gleizes M-P et Glize P., La conception de systèmes multi-agents adaptatifs : contraintes et spécificités. Atelier de Méthodologie et Environnements pour les Systèmes Multi-Agents (SMA 2001), Plate-forme AFIA, Grenoble du 25 au 28 juin 2001.
[Bernon & al., 2002] Bernon C., Gleizes M-P., Picard G., Glize P., The ADELFE Methodology For an Intranet System Design, - Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002), Toronto (Ontario, Canada) at CAiSE'02, 27-28 May 2002.
[Bertalanffy, 1973] Bertalanffy L.W., La théorie générale des systèmes, Dunod, Paris, 308 p. ISBN : 210-001841-8, 1973.
[Bestougeff & Ligozat, 1989] Bestougeff H. et Ligozat G.. Outils logiques pour le traitement du temps : de la linguistique à l'intelligence artificielle. Masson, 1989.

[Bigus, 1996] Bigus J., Data Mining with Neural Networks, New York, 1996.
[Bonabeau & al., 1999] Bonabeau E., Dorigo M., Theraulaz G., Swarm Intelligence, from natural to artificial systems, Oxford University Press, 1999. ISBN : 0-19-513158-4
[Booch, 1994] Booch G., Analyse et conception orientées objets – 2ième Ed. Addison-Wesley, ISBN : 2-87908-069-X, 1994.
[Bouchenak, 2001] Bouchenak S.. Making Java Applications Mobile or Persistent. 6th USENIX Conference on Object-Oriented Technologies and Systems, San Antonio (Texas - USA), 29th January - 2nd February 2001.
[Boucher & al., 1998] Boucher A., Doisy A., Ronot X., Garbay C. Cell migration analysis after in vitro wounding injury with a multi-agent approach, Artificial Intelligence Review 12: 137-162, 1998.
[Brazier & al., 1995] Brazier F.M.T., Dunin-Keplics B., Jennings N.R and Treur J., Formal specification of multi-agent systems : a real world case, in ICMAS'95, First International Conference on Multi-agent Systems, San Francisco, California, USA, 1995.
[Brazier & al., 1997] Brazier F.M.T., Dunin-Keplicz B., Jennings N.R. and Treur J., Desire : modelling multi-agent systems in a compositional formal framework, International Journal of Cooperative Information Systems, vol. 6, no. 1, Special Issue on Formal Methods in Cooperative Information Systems : Multiagent Systems, M. Huns, M. Singh (rds.), 1997.
[Brooks, 1989] Brooks R., A robot that walks : emergent behaviour from a carefully evolved network, Neural computation, 1:253-62, 1989.
[Bruneton, 2002] Bruneton E., Coupaye T. and Stefani J-B.. Recursive and Dynamic Software Composition with Sharing. In Proceedings of the 7th ECOOP International Workshop on Component-Oriented Programming (WCOP'02), Malaga (Spain), June 10th-14th, 2002.
[Buckle, 2000] Buckle P., Enabling the "Proactive Internet" with FIPA Agents. Invited key note speech at MATA, Paris, September, 2000.
[Buisson, 1990] Laurent Buisson, Le raisonnement spatial dans les systèmes à base de connaissances - application à l'analyse de sites avalancheux, Thèse d'informatique, université Joseph Fourier, Grenoble (FR), novembre 1990.
[Buschmann & al., 1996] Buschmann F., Meunier R., Rohnert H., Sommerlad P. and Stal M., Pattern-oriented software architecture – A system of patterns, John Wiley & Sons, Chichester, 1996.
[Caetano & Mathieu, 1997] Caetano H. et Mathieu J., Des systèmes de formation à la gestion de catastrophes fondés sur la coopération entre institutions intervenantes, Première Conférence Francophone de Modélisation et de Simulation, Systèmes de Production et de Logistique (MOSIM'97), pp. 135-146, Rouen (1997).
[Caetano, 1992] Caetano H., Un simulateur de catastrophes naturelles : cas des marées blanches », Rapport technique LAFORIA, 1992.
[Camazine & al., 2001] Camazine S., Deneubourg J-L., Franks N.R., Sneyd J., Theraulaz G., and Bonabeau E., Self-Organization in Biological Systems, Princeton University Press, 2001.
[Capera & al., 2002] Un défi pour les SMA: explorer l'émergence - D.Capera, J.-P.Georgé, M.-P.Gleizes, P.Glize, JFIADSMA 2002, Lille France.
[Cappello, 2002] Cappello F., "Calcul Global Pair a Pair : extension des systèmes Pair a Pair au calcul", Lettre de l'IDRIS, pages 14-25, Janvier 2002.
[Cardon & Durand, 1997] Cardon A. and Durand S., A model of crisis management system including mental representations. In Proceedings of the AAAI Spring Symposium, 1997.
[Cardon & Guessoum, 2000] Cardon A. & Guessoum Z., Systèmes multi-agents adaptatifs - Actes des Huitièmes Journées Francophones d'Intelligence Artificielle Distribuée et Systèmes Multi-Agents - Éditions Hermès, Novembre 2000.
[Cardon, 1996a] Cardon A., Le caractère fondamental des systèmes finalisés : la complexité profonde, Troisième Congrès Européen de Systémique, p. 951-956, Rome, 1-4 Octobre 1996.
[Cardon, 1996b] Cardon A., La complexité des systèmes d'expression du sens, Rapport Laforia Paris VI, 96-09, Février 1996.
[Cardon, 1997] Cardon A., A multi-agent model for co-operative communications in crisis management system: the act of communication. In Proceedings of the 7th European-Japanese Conference on Information Modelling and Knowledge Bases, pages 111-123, 1997.
[Carron & al., 1999] Carron, Proton H. and Boissier O., A Temporal Agent communication language for dynamic Multi-Agent Systems, MAAMAW'99 (Modelling Autonomous Agents in a Multi-Agent World), Valencia, Espagne, juillet 1999
[Castelfranchi, 1995] Castelfranchi C., Commitments : from individual intentions to groups and organizations. In Proceeding of International Conference on MultiAgent Systems (ICMAS95), 1995.
[Casteran & al., 2000] Casteran J-C., Gleizes M-P. & Glize P. Des méthodologies orientées multi-agents, Actes des Huitièmes Journées Francophones d'Intelligence Artificielle Distribuée et Systèmes Multi-Agents - Éditions Hermès, Novembre 2000.

[Castillo & al., 1998] Castillo A., Kawaguchi M., Paciorek N., and Wong D., Concordia as Enabling Technology for Cooperative Information Gathering. In Japanese Society for Artificial Intelligence Conference, 17-18 June 1998. http://www.meitca.com/HSL/Projects/Concordia/.

[Chambina & al., 1998] Chambina N., Wybo J-L, Guarnieri F., Acquisition de données météorologiques en temps réel, le système météo 06, rapport interne EMP Sophia-Antipolis, 1998.

[Charlet & al., 1994] Charlet J., Bachimont B., Bouaud J. et Zweigenhaum P., Ontologie et réutilisabilité: expérience et discussion, Actes JAC'94, Strasbourg, 1994.

[Chevrier, 1993] Chevrier V., Etude et mise en oeuvre du parasigme multi-agents : de l'atome à Gtmas, Thèse de l'Université de Nancy I, 1993.

[Cliff & al., 1994] Cliff D., Husbands P., Meyer J-A. and Wilson S. (1994). From Animals to Animats 3. Proceedings of theThird International Conference on Simulation of Adaptive Behavior. The MIT Press/Bradford Books.

[Collinot & al., 1996] Collinot A., Ploix L. et Drogoul A., Application de la méthode Cassiopée à l'organisation d'une équipe de robots, in IA distribuée et systèmes multi-agents (JFIADSMA'96), Hermès, 137-152, 1996.

[Conry & al., 1988] Conry S. Meyer R. A et Lesser V., Multistage negociation in distributed artificial intelligence, in readings in distributed artificial intelligence, A. Bond et L. Gesser (Ed), Morgan Kaufman, 1988.

[Cosnard & al., 2002] Cosnard M., Alessandrini V., Desprez F., Primet P., Priol T., Roucairol C., Vandromme D., Les réseaux à très hauts débits et le middleware pour les grilles de calcul, rapport de mission pour la science et la technologie pour l'ambassade de France aux Etats-unis, février 2002. www.france-science.org/photos/1015346325_rapport_mission_grid.pdf

[Cranefield & al., 2001] Cranefield S., Hausteiny S., Purvis M., UML-Based Ontology Modelling for Software Agents, Proceedings of Ontologies in Agent Systems Workshop, Agents 2001, Montreal, pp. 21-28.

[Criado & Cardonna, 1996] Criado et Cardona, Interface Wilfried-Bosque, Deliverable number D06.3 version 1.0n Project's internal reference WP06.DL.03.01, novembre 1996.

[Daniels, 1999] Daniels M., Integrating Simulation Technologies with Swarm, in Agent Simulation: Applications, Models and Tools, University of Chicago, 15-16 October 1999. www.swarm.org

[Darley, 1994] Darley V., Emergent Phenomena and complexity, Proceedings of Artificial life Workshop IV. Massachusetts Institute of Technology, Cambridge MA 02139, USA, MIT Press.

[Decker & Lesser, 1995] Decker K.S., Lesser V.R., Designing a Family of Coordination Algorithms, Proceedings of the First International Conference on Multi-Agent Systems ICMAS'95, 1995.

[Degirmenciyan & Marc, 2002] Degirmenciyan I. et Marc F., SCALA une approche multi-agents pour la conception de systèmes complexes application à la simulation de missions aériennes, JFIADSMA 2002, Lille, France.

[Delacour, 2001] Delacour J., Conscience et cerveau : La nouvelle frontière des neurosciences, De Boeck Ed. paris, 340p. ISBN : 2-8041-3766-X, 2001.

[DeLoach, 1999] Deloach S., A. Multiagent Systems Engineering : A Methodology and Language for Designing Agent Systems. In Proc. of Agent Oriented Information Systems, pages 45–57, 1999.

[Demazeau, 1995] Demazeau Y., From interactions to collective behaviour in agent-based systems, Proceedings of the first European Conference on Cognitive Science'95:117-132, 1995

[Demazeau, 1997] Y. Demazeau, «Steps toward Multi-Agent Oriented Programming », In First International Workshop on Multi-Agent Systems, IWMAS'97, Boston, 1997.

[Deneubourg & al., 1990a] Deneubourg J-L., Aron S., Goss S. and Pasteels J.M., J. Insect Behav. 32, 15, 1990.

[Deneubourg & al., 1990b] Deneubourg J-L., Goss S., Francks N.R., Sendova-Franks A., Detrain C. et Chretien L., The dynamics of collective sorting : robot-like ants and ant-like robots, in Meyer J-A et Wilson S. Eds, Simulation of Adaptive Behaviour : from animals to animats, Cambridge, Mass. : MIT Press, 356-65.

[Doncieux, 1999] Doncieux, S., Evolution d'architectures de contrôle pour robots volants. In Drogoul et Meyer (Eds.). Intelligence Artificielle Située. Hermes, 1999.

[Dorigo & Gambardella, 1997] Dorigo M., Gambardella L.M., Ant colonies for the traveling salesman problem, BioSystems, 43, 73-81, 1997.

[Dreyfus & al., 2002] Dreyfus G., Martinez J-M., Samuelides M., Gordon M.B., Badran F., Thiria S., Herault L., Réseaux de neurones – méthodologie et applications, Eyrolles 2002, ISBN 2-212-11019-7

[Dreyfus, 1984] Dreyfus H., Intelligence artificielle – mythes et limites, Flammarion, 1984.

[Drogoul, 1993] Drogoul A., De la simulation multi-agent à la résolution collective de problèmes, Thèse de l'Université P et M Curie, 1993.

[Drogoul, 1998] A. Drogoul, J.-D. Zucker, LIP6 1998/041: Rapport de Recherche LIP6 - 30 pages - Octobre/October 1998

[Durand & al., 1999] Durand S., Lesage F., Cardon A., et Tranouez P., Représentation par des organisations d'agents des connaissances échangées dans un système de gestion des situations d'urgence. In Actes des JFIADSMA'99, la Réunion, 1999.

[Durand, 1996] Durand B. Simulation multiagents et épidémiologie opérationnelle : étude d'épizooties de fièvre aphteuse. Thèse de doctorat, Université de Caen, 1996.
[Durfee & al., 1989] Durfee E.H, Lesser V.R et Corkill D.D, Coorperative distributed problem solving, in the handbook of artificial intelligence, vol IV, A Barr, P.R Cohen et E.A Feigenbaum (Ed), p.83-148, Addison-Wesley, 1989.
[Durfee & Montgomery, 1989] Durfee E.H. and Montgomery T.A., MICE: A Flexible Testbed for Intelligent Coordination Experiments. Artificial Intelligence Laboratory, Proceedings of the Ninth Workshop on Distributed AI, 1989.
[Dutech, 1999] Dutech A., Apprentisage d'environnement : Approches cognitives et comportementales, Thèse de doctorat Soutenue à l'ENSAE, Toulouse, le 1er février 1999.
[ElFallah & Haddad, 1996] El Fallah-Seghrouchni A., Haddad S., A Coordination Algorithm for Multi-Agent Planning. MAAMAW, 86-99, 1996.
[ElHadouaj & al., 2000] El Hadouaj S., Drogoul A. and Espie S., How to Combine Reactivity and Anticipation: the Case of Conflicts Resolution in a Simulated Road Traffic, MABS'2000 workshop, Springer Verlag LNAI series, Boston, USA.
[Erceau, 1993] Erceau J., Intelligence Artificielle Distribuée et Systèmes Multi-Agents — de la théorie aux applications. 23ème Ecole Internationale d'Informatique de l'AFCET, Neuchâtel, 1993.
[Erman & al., 1980] Erman L.D., Hayes-Roth F., Lesser V.R. and Raj Reddy D.,The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty. Computing Surveys, 12(2):213-253, 1980.
[Erman & al., 1988] Erman, L.D. and Hayes-Roth F. ABE : An environment for Engineering Intelligent Systems. IEEE Transacions on Software Engineering vol.14(12), p.1758, Décembre 1988.
[Ermine & al., 1996] Ermine J-L, Chaillot M., Bigeon P., Charreton B. and Malavieille D., MKSM, a method for knowledge management, dans "Knowledge Management, Organization, Competence and Methodology", Advances in Knowledge Management, Volume 1, Jos F. Schreinmakers Ed., pp. 288-302, Ergon Verlag, 1996.
[Esmahi, 2001] Esmahi L., Coopération dans les systèmes multi-agents : une analyse et un point de vue, JFIADSMA'01, p31-53, Montréal, Canada, 2001.
[Fenet & Hassas, 1998] Fenet S., Hassas S. : An ant-like multi-agents system on the network: a dynamic repartition guided by interaction In: Multi-Agent systems Models, Architectures and Applications, proceedings of II Iberoamerican Ws on Distributed Artificial Intelligence & Multi-Agents Systems, Toledo, 1-2 Oct. 1998.Madrid: Iberamia, 1998. pp. 1-6.
[Ferber & Ghallab, 1988] Ferber F. et Ghallab M., Problématique des univers multi-agents intelligents. In Actes des 2èmes Journées du PRC-GRECO IA, éd. D. Pastre, p. 295-320, Toulouse, 14-15 mars 1988.
[Ferber, 1995] Ferber J., Les systèmes multi-agents : vers une intelligence collective, InterEditions, ISBN : 2-72-96-0572-X, 1995.
[Ferber, 1999] Ferber J., Multi-Agent Systems - An introduction to distributed artificial intelligence, Addison-Wesley, ISBN 0-201-36048-9, 1999.
[Finin & al., 1997] Finin T., Labrou Y. and Mayfield J., KQML as an agent communication language, in Jeff Bradshaw (Ed.), ``Software Agents", MIT Press, Cambridge, 1997.
[FIPA, 2002] Foundation for Intelligent Physical Agent. Site web www.fipa.org
[Flacher & al., 2002] Flacher, F. et Sigaud, O. (submitted): Tuning Social Potential Field with Genetic Algorithms, Proceedings of the International Conference of Artficial Intelligent Systems (ICAIS 2002), Australia.
[Flake & al., 2002] Flake G.W., Lawrence S., Giles C.L., Coetzee F., Self-Organization and Identification of Web Communities, Journal IEEE Computer, Vol.35, Numéro 3, March 2002, 66-71.
[Foisel, 1996] Foisel R., Chevrier V. et Haton J-P., De l'organisation d'une société à sa réorganisation, Actes de la journée "Systèmes Multi-Agents", Toulouse, P.R.C.-G.D.R Intelligence Artificielle, pp. 121-128, 1996.
[Foster & Kesselman, 1998] Foster I. and Kesselman C., The Grid : blueprint for a new computing infrastructure, Morgan-Kauffmann, 1998.
[Fowler & al., 1999] Fowler J., Perry B., Nodine M., and Bargmeyer B., Agent-Based Semantic Interoperability in InfoSleuth, SIGMOD Record 28:1, pp. 60-67, March, 1999.
[Franklin & Gasser, 1996] Fraklin S., Graesser A., Is it an agent or just a program ? : a taxonomy for autonomous agents, in proceedings of the 3rd international workshop on agent theories, Architecture and languages, Springer Verlag, 1996.
[Franklin, 1997] Franklin S., Coordination without Communication, Institute or Intelligent Systems, University of Memphis. http://www.msci.memphis.edu/~franklin/coord.html
[Friedman, 1968] Friedman M., Inflation et systèmes monétaires, 1968, Calmann-Lévy, 1976.
[Friedman, 1992] Friedman M., « free to choose », 1981, cité par J-p Dupuy in Introduction aux sciences sociales, logiques des phénomènes collectifs », Edition Ellipse, 1992.
[Friedman, 1995] Friedman M., Essais d'économie positive, Litec, L.I.B.E.R.A.L.I.A., 1995.

[Gallone & al., 1994] Gallone J-M., Charpillet F. et Chevrier V., Un modèle d'agent pour un raisonnement contraint par le temps, - In: Actes Journée PRC-IA sur les Systèmes Multi-Agents, Paris, Décembre 1994.
[Gasser & al., 1987] Gasser L., Braganza C., and Herman N., "MACE: A flexible testbed for distributed AI Research," in Distributed Artificial lntelligence, Michael N. Huhns, Ed. San Mateo, CA: Morgan aufmann Publishers, pp. 119-152, 1987.
[Gasser, 1992] Gasser L., An overview of DAI, Kluwer Academic Publisher, Boston, 1992.
[Gaussier & al., 1998] Gaussier P., Moga S., and Quoy M., From perception-action loops to imitation processes: A bottom-up approach of learning by imitation, Robotics and autonomous systems, 16:333--356. Applied Artificial intelligence (AAI), 12, 1998.
[Geib & al., 1999] Geib J-M., Gransart C. et Merle P., Corba. Des concepts à la pratique, 2eme Ed. Dunod, ISBN : 2100048066, 1999.
[Gérard & al., 2001] Gérard, P. and Sigaud, O. (2001) Généralisation et Apprentissage Latent dans les Systèmes de Classeurs Extraction des connaissances et apprentissage : Apprentissage automatique et évolution artificielle. 3(1), 87-114. Hermès.
[Germond & al., 1999] Germond L., Dojat M., Taylor C., Garbay C., A Multi-Agent System for MRI Brain Segmentation, Proc. of the 7th European Conference on Artificial Intelligence in Medicine. In Lecture Notes in Artificial Intelligence, (W. Horn, Y. Shahar, G. Lindberg, S. Andreassen & J. Wyatt, eds), Springer-Verlag, pp. 423-432, 1999.
[Glaser, 1997] Glaser N., "The CoMoMAS Methodology and Environment for Multi-Agent System Development" in: C. Zhang, D. Lukose (eds.), Multi-Agent Systems - Methodologies and Applications, LNAI 1286, Springer-Verlag, Berlin, Germany, 1997, pp. 1 - 16.
[Gleick, 1999] Gleick J., La théorie du chaos, ISBN: 2-08-081219-x, 432 pages. Champs Flammarion, 1999.
[Gleizes & al., 1994] Gleizes M-P., Glize P., Trouilhet S., Etude des lois de la conversation entre agents autonomes, Revue internationale de systémique, 8(1), p.39-50., 1994.
[Gleizes & al., 1999] Gleizes M-P., Camps V., Glize P. , A theory of emergent computation based on cooperative self-organization for adaptive artificial systems - Fourth European Congress of Systems Science - Valencia, 1999
[Goldberg & al., 1996] Goldberg I., Wagner D., Thomas R. and Brewer E.A., A secure environnemnt for untrusted helper applications, dans le recuil d'articles de Usenix'96, 1996.
[Granet, 1988] La pensée chinoise de Marcel Granet, édité à La Renaissance du Livre en 1934, puis réédité aux Éditions Albin Michel en 1988, coll. "Bibliothèque de l'Evolution de l'Humanité. ISBN : 2226104747
[Griffin, 1984] Donald Griffin, Animal Thinking, Cambridge, Mass. : Harvard University Press, 1984.
[Guarnieri & Wybo, 1995] Guarnieri F. and Wybo J-L., Spatial Decision Support and Information managment, Application to Wildland fire prevention : the Wieldfried system, Safety Science 20, pp.3-12, Elseiver, 1995.
[Guarnieri, 1994] Guarnieri F., Aide à la décision à référence spatiale dans le domaine de la prévention des incendies de forets : le système WilFried, Hermes, Volume 4.n°3-4/1994, Revue Internationale de géomatique, 1994.
[Guessoum & Briot, 1999] Guessoum Z. and Briot J-P., From Active Objects to Autonomous Agents, IEEE Concurrency. vol. 7 N° 3, pp. 68-78, July/september, 1999.
[Guessoum & Durand, 1996] Guessoum Z. et Durand R., Des agents intelligents pour modéliser l'évolution des entreprises. In Intelligence Artificielle Distribuée, Hermès, Paris, pp. 37-58, Avril 1996.
[Guessoum, 1996] Guessoum Z., Environnement de développement et de conception de systèmes multi-agents. Thèse, Université Paris 6, 1996.
[Guessoum, 1998] Guessoum Z., DIMA : Une plate-forme multi-agents en Smalltalk. Revue Objet, Volume 3 No 4, pp.393-410, 1998.
[Gutknecht & al., 2000] Gutknecht O., Ferber J. et Michel F., MadKit: Une expérience d'architecture de plate-forme multi-agent générique, JFIADSMA'2000, Octobre 2000.
[Gutknecht & Ferber, 1998a] Gutknecht O. and Ferber J., Aalaadin : a meta-model for the analysis and design of organizations in multi-agent systems, ICMASí98, Paris, 1998.
[Gutknecht & Ferber, 1998b] Gutknecht O. et Ferber J., Un méta-modèle organisationnel pour l'analyse, la conception et l'exécution de systèmes multi-agents, JFIADSMA1998.
[Gutknecht & Ferber, 2000] Gutknecht O. and Ferber J., The MADKIT Agent Platform Architecture, 1st Workshop on Infrastructure for Scalable Multi-Agent Systems, Barcelona, June 2000. http://www.madkit.org
[Hachicha & al., 2002] Hachicha H., Loukil A. et Ghedira K., MA-UML : une extension de A-UML aux agents mobiles, JFIADSMA 2002, France, Lille.
[Hannoun & al., 1999] Hannoun M., Boissier O., Sichman J.S., Sayettat C., MOISE : un modèle organisationnel pour la conception de systèmes multi-agents, Actes des Journées Francophones IAD et SMA, La réunion, Novembre 1999.

[Harrouet, 2000] Harrouet H., oRis : s'immerger par le langage pour le prototypage d'univers virtuels à base d'entités autonomes, Thèse de doctorat en Informatique, Université de Bretagne Occidentale, Brest (France), 8 décembre 2000.
[Hayes-Roth, 1985] Hayes-Roth B., A Blackboard Architecture for Control. Artificial Intelligence 26(3):251-321, 1985.
[Hempel & Oppenheim, 1948]
[Herzip & al., 2000] Herzig A., Longin D. and Virbel J. Towards an analysis of dialogue acts and indirect speech acts in a BDI framework. In Massimo Poesio et David Traum, éditeurs, Proc. Fourth Int. Workshop on the Semantics and Pragmatics of Dialogue (Götalog-2000), Göteborg University, Sweden, 2000.
[Hewitt, 1977] Hewitt C., "Viewing Control Structures as Patterns of Passing Messages", Artificial Intelligence 8(3), 323-364. 1977.
[Hill, 2002] Hill D.R.C, Nouveaux defies pour les simulations et les SMA, Conférence invitee, JFIADSMA02, 2002.
[Hirano & al., 1998] Hirano S., Yasu Y., Igarashi H., Performance evaluation of popular distributed objet technologies for Java, Workshop on Java for High-Performance Network Computing, 1998.
[Hirano, 1997] Hirano S., HORB : Distributed execution of Java Programs, WorldWide computing and its application, springer lecture notes in computer science, pp.29-42, 1997.
[Hochmann & Jeannerod, 1996] Hochmann J., Jeannerod M., Esprit, où es-tu ?, Ed. Odile Jacob, 1996.
[Hodges, 1988] Hodges A., Alan Turing ou l'énigme de l'intelligence, Bibliothèque scientifique, Payot, 1988.
[Homere, ...] Homere, L'Iliade, Garnier-Flammarion.
[Horn & al., 1998] Horn E., Kupries M., and Reinke T., Properties and Models of Software Agents and Prefabrication for Agent Application Systems. Software Technology Track of the Thirty-Second Hawaii International Conference on System Sciences (HICSS-32), 1998.
[Hubert, 1999] Huber M.J., JAM: a BDI-theoretic mobile agent architecture, Proceedings of the third annual conference on Autonomous Agents, Seattle, Washington, United States, 1999. http://members-http-2.rwc1.sfba.home.net/marcush/IRS/
[Hubert, 1999] Marcus J. Huber, JAM: A BDI-theoretic Mobile Agent ArchitectureProceedings of the Third International Conference on Autonomous Agents (Agents'99), pgs 236-243, Seattle, WA, May, 1999, http://www.marcush.net/
[Huguet, 2002] Huget M-P., Une application d'Agent UML au Supply Chain Management, JFIADSMA 2002, France, Lille.
[Iglesias & al., 1998] Iglesias A., Garijo M. and Gonzalez C. A survey of agent-oriented methodologies, In Proceedings of the 5th International Workshop on Agent Theories, Architectures and Languages (ATAL'98), LNAI n1555 - Springer Verlag, Paris, France, pp:317-330, July 1998.
[IKV, 1999] IKV++ GmbH Informations und Kommunikationssystem. Grasshopper : the agent platform – Technical Overview. IKV++ GmbH Informations-und Kommunikationssystem, February 1999. http://www.ikv.de/products/grasshopper/
[Jaber & al., 1998] Jaber A., Guarnieri F. et Wybo J-L., "Un système d'agents logiciels intelligents pour favoriser la coopération entre des systčmes d'aide à la décision dédiés à la gestion de crise", cole des mines de Paris, Pôle Cindyniques, JFIADSMA'98, actes des 6émes journées francophones, éditions HERMES, Paris, 1998.
[James, 1890] James W., Psychology (briefer course). Holt., 1890.
[Jemam, 1993] Jemam D., "L'économie découvre le chaos", Alternatives économiques, 103, pp. 59 – 61, 1993.
[Jennings & al., 1998] Jennings N.R., Sycara, and Wooldridge M., Autonomous Agents and Multi-Agent Systems 1(1), 1998.
[Jennings & Wooldridge, 2000] Jennings N. R and Wooldridge M., Agent-oriented software engineering - In J Bradshaw (Ed.), Handbook of Agent technology, AAAI/MIT Press 2000.
[Jensen, 2001] Jensen F.V., Bayesian Networks and Decision Graphs (Statistics for Engineering and Information Science), Springer-Verlag Telos; ISBN : 0387952594, 2001
[Johnson, 1975] Johnson S. C., Yacc: Yet Another Compiler Compiler, Computing Science Technical Report No. 32, 1975, Bell Laboratories, Murray Hill, NJ 07974.
[Jouet, 2001] Jouet C., Ontologie du projet ANTIC, Rapport de DEA, INSA de Rouen, 2001.
[Kann, 2001] Kann G., Gnutella, dans peer to peer : harnessing the power of disruptive technologies, A.Oram éditeur, edition O'Reilly, mars 2001.
[Karjoth & al., 1997] Karjoth G., Lange D.B. and Oshima M., A Security Model for Aglets, in IEEE Internet Computing, Vol. 1, No. 4, July/August 1997.
[Kinny & al., 1996] Kinny D., Georgeff M., and Rao A. A Methodology and Modeling Technique for Systems of BDI Agents. In W. van der Velde and J. Perram, editors, Agents Breaking Away. Proceedings of the 7' European Workshop on Modelling Autonomous Agents in a Multi-Agent World MAAMAW'96, Springer-Verlag, 1996.
[Koblick, 1999] Koblick R., Concordia. Communications of the ACM, 42(3):96-99, March 1999.

[Labbani, 1998] Labbani-Igbida O., Müller J-P., and Bourjault A., Cirta: an emergentist methodology to design and evaluate collective behaviours in robots colonies dans Collective Robotics Workshop'98, Paris, Agent World 1998.
[Lagadec, 1991] Lagadec P. La gestion des crises. McGraw-Hill. 1991.
[LaMettrie, 1974] La Mettrie, Textes choisis,ed. par Marcelle Bottigelli-Tisserand, Paris 1974
[Lange & al., 1998] Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, J. White , MASIF: The OMG Mobile Agent System Interoperability Facility, 2de MA Workshop, Stuttgart, Allemagne, 9-11 septembre, 1998.
[Lange & Oshima, 1998] Lange D. and Oshima M., Programming and Deploying Java Mobile Agents with Aglets, Addison Wesley, ISBN: 0-201-32582-9, 1998.
[Langley, 2001] Langley A., Freenet, dans dans peer to peer : harnessing the power of disruptive technologies, A.Oram éditeur, edition O'Reilly, mars 2001.
[Laszlo & Laugel, 1998] Laszlo C., Laugel J-F., L'économie du chaos , Editions d'Organisation, 1998
[Lebaron, 1999] LeBaron B., "Evolution and Time Horizons in an Agent-Based Stock Market", Computing in Economics and Finance '99 1342, Society for Computational Economics, 1999.
[Lee & Durgee, 1994 Jaeho Lee and Edmund H. Durfee, Structured circuit semantics for reactive plan execution systems, AAAI'94, p.1232-1237.
[LeMoigne, 1990] Le Moigne J-L., La modélisation des systèmes complexes. Paris : Dunod, 1990.
[Lesage, 2000] Lesage F., Interprétation adaptative du discours dans une situation multiparticipants : modélisation par agents, Thèse de doctorat, Université du Havre – LIH, 2000
[Lesourne, 1991] Lesourne J., Economie de l'ordre et du désordre, Economica, 1991
[Lesser & Corkill, 1983] Lesser V.R. and Corkill D.D, The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks », AI Magazine, 4(3), pp.15-33, 1983.
[LeStrugeon, 1995] Le Strugeon E., Une méthodologie d'auto-adaptation d'un système multi-agents cognitifs, Thèse de doctorat, Université de Valenciennes. 1995.
[Li & Vitanyi, 1997] Li M. and Vitanyi P., An Introduction to Kolmogorov Complexity and Its Applications, Second Edition Springer Verlag, ISBN : 0-387-94868-6, 1997.
[Logan, 1998] Logan B., Classifying agent systems, in Software Tools for Developing Agents: Papers from the 1998 Workshop, J. Baxter and B. Logan Cochairs, Technical Report WS-98-10, AAAI Press, pp 11-21, 1998. http://www.cs.bham.ac.uk/~bsl/aaai-98/agent-classification.html
[Logan, 1999] Logan B., Software Tools for Developing Agents (with J. Baxter), Report on the AAAI-98 Workshop, AI Magazine, Spring 1999/Vol. 20, No. 1, pp 124-125, 1999.
[M.R. Jean, 1997] M.R.Jean (Collective name), Emergence et SMA, dans JFIADSMA'97, Hermès, Nice, pp. 323-342, 2-4 Avril 1997.
[Maes & al., 1996] Maes P., Mataric M., Pollack J., Meyer J-A. and Wilson S. From Animals to Animats 4. Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior. The MIT Press/Bradford Books, 1996.
[Maes & Brooks, 1990] Maes P. et Brooks R., Learning to coordinate behaviours, in proceedings of the eight national conference on artificial intelligence, Menlo Park, Calif., AAAI 796-802, 1990.
[Maes, 1995] Maes P., Intelligent Software, Scientific American, volume 273, numéro 3, septembre 1995, pp. 84-86.
[Magnin & Alikacem, 1999] Magnin L., Hachemi Alikacem A., GenA: multiplatform generic agents, Mata '99, Ottawa (Canada), October 6-8, 1999.
[Malavieille & al., 1995] Malavieille D., Ermine J-L., Cdt Ribot, Cpt Lions, Lt Condamin, Étude et réalisation d'un système à base de connaissances d'aide à la décision en cas d'intervention sur des accidents majeurs, IA'95, 26-30, Montpellier, juin 1995.

[Malone, 1988] Malone T.W., What is coordination theory », in national science foundation coordination theory workshop, MIT, 1988.
[MaSE, 2000] MASE Project Web Page. http://www.cis.ksu.edu/~sdeloach/ai/projects.htm
[Mathieu & al., 2002a] Mathieu P., Routier J-C., Yann Secq, Dynamic organization of multi-agent systems. AAMAS'02, Bologne, 2002, p.451-452
[Mathieu & al., 2002b] Mathieu P., Routier J-C., Yann Secq: Principles for Dynamic Multi-agent Organizations. PRIMA 2002: 109-122
[McFarlard, 1990] McFarlard D., dictionnaire du comportement animal, Robert Laffont, 1990.
[Merghem & Gaïti, 2002] Merghem L. et Gaïti D., Une approche multi-agents pour la simulation de réseaux de télécommunications, JFIADSMA02, 2002.
[Mesle, 1994] Mesle, R.. ICHTYUS : architecture d'un système multi-agents pour l'étude de structures agrégatives, Rapport de DEA, LAFORIA, Université Paris 6, 1994.
[Message, 2001] Rapport Eurescom P907, MESSAGE: Methodology for Engineering Systems of Software Agents, Septembre 2001, 73p. http://www.eurescom.de/public/projectresults/P900-series/907ti1.asp

[Metcalfe & al., 2000] Metcalfe J.S., Fonseca M.D. and Ramoglan R., Innovation, Growth and Competition: Evolving Complexity or Complex Evolution, Complexity and Complex Systems in Industry. Sept 2000.
[Meyer & al., 1993] Meyer J-A., Roitblat H., and Wilson S. (Eds.). From Animals to Animats 2. Proceedings of the Second International Conference on Simulation of Adaptive Behavior. The MIT Press/Bradford Books, 1993.
[Meyer & Wilson, 1991] Meyer J-A., and Wilson S., "Simulation of Adaptive Behavior: From Animals to Animats." Complex Adaptive Systems, MIT Press, Cambridge, 551, 1991.
[Meyer, 1998] Meyer, J-A. The Animat Approach : Simulation of Adaptive Behavior in Animals and Robots. In Alexandre, F. and Kant, J.D. (Eds.). Actes de la conférence Neurosciences Pour l'Ingénieur. Publication Loria. 1998.
[Michaels, 1995] Michaels M., Seven fundamentals of Complexity, in Chaos and society, Ed. Alain Albert, Montréal, Presse de l'Université du Québec & IOS Press, 1995.
[Milner, 1999] Milner R., Communicating and Mobile Systems: the Pi-Calculus. Cambridge University Press, ISBN: 0521658691, 1999.
[Milojicic & al., 1998] Milojicic D., Breugst M., Busse I., Campbell J., Covaci S., Friedman B., Kosaka K., Lange D.B., Ono K., Oshima M., Tham C., Virdhagriswaran S. and White J., MASIF: The OMG Mobile Agent System Interoperability Facility. Mobile Agents, p.50-67, 1998.
[Minar & al, 1999] Minar N., Gray M., Roup O., Krikorian R., and Maes P., Hive: Distributed Agents for Networking Things, ASA/MA'99, August 1999. http://www.hivecell.net/
[Mingers, 1994] Mingers J., "Self-Producing Systems: Implications and Applications of Autopoiesis", Plenum Publishing, New York, 1994.
[Minsky, 1988] Minsky M., La société de l'esprit, Intereditions, 1988.
[Morin, 1977] Edgar Morin, La méthode (1) : la nature de la nature, Le Seuil, 1977.
[Müller, 1998] Jôrg P. Müller, "The right agent (architecture) to do the right thing", ATAL'98, AgentWorld98, pp.1-16, Paris, Juillet 1998.
[Muller, 1998] Müller J-P., Méthodologie de conception de systèmes multi-agents de résolution de problèmes par émergence, dans JFIADSMA'98, Hermès, novembre 1998.
[Nader & Thompson, 2000] Azarmi N. and Thompson S., ZEUS: A Toolkit for Building Multi-Agent Systems, Proceedings of fifth annual Embracing Complexity conference, Paris April 2000.
[Newell, 1990] Newell A., Unified Theories of Cognition, Harvard Univ. Press, Cambridge, Mass., 1990.
[Nino, 1995] Nino S., Réaliser un compilateur ; Les outils Lex et Yacc, 2ème édition, Eyrolles, 1995.
[Nourredine & Mathieu, 1997a] Nourredine Bensaid, Philippe Mathieu. A Hybrid Architecture for Hierarchical Agents. in Proceedings of the International Conference on Computational Intelligence and Multimedia Applications, ICCIMA'97. GRIFFITH UNIVERSITY.
[Nourredine & Mathieu, 1997b] Bensaid N., Mathieu P., A Framework for Cooperation in Hierarchical Multi-Agent Systems. in Mathematical Modelling and Scientific Computing. vol 8. 1997.
[Nwana & al., 1999] Nwana, H. S., Ndumu, D. T., Lee, L. C., and Collis, J. C., "ZEUS: A Toolkit for Building Distributed Multi-Agent Systems," Applied Artificial Intelligence Journal, vol. 1, no. 13, pp. 129-185, 1999,
[Occello & al., 2002] Occello M., Baeijs C., Demazeau Y. and Koning J-L.. MASK : An AEIO Toolbox to Develop Multi-Agent Systems. in Cuena et al. (eds), Knowledge Engineering and Agent Technology, IOS Series on Frontiers in AI and Applications, Amsterdam, The Netherlands. 2002.
[Odell & al., 2000] Odell J., Parunak V.D.H. and Bauer B., Extending UML for Agents, AOIS Workshop at AAAI 2000.
[Odell, 1998] Odell J., Advanced Object-Oriented Analysis and Design using UML, Cambridge University Press, 1998.
[Paciorek & Wong, 1998] Paciorek N. and Wong D., "Concordia: A Mobile Agent Platfom for Complex Distributed Systems", Workshop on Agent Technologies (AT), 17-18 November 1998 .
[Parunak & Brueckner, 2001] Parunak H.V.D. and Brueckner S., Entropy and self-organization in multi-agent systems – In Proc. Of the International Conference on Autonomous Agents, 2001.
[Parunak, 1998] Parunak V.D., Agent-based emulation vs. Numerical Evaluation : A users'guide. Multi-agent Based Simulations, 1998.
[Pfeifer & al., 1998] Pfeifer, R.., Blumberg, B., Meyer, J.A. and Wilson, S.W. (Eds.). From Animals to Animats 5. Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior. The MIT Press/Bradford Books. 1998.
[Pinson & Balbo, 2001] Pinson S. et Balbo F., "Toward a Multi-Agent Modelling Approach for Urban Public Transportation Systems", in Omicini A., Petta P. et Tolksdorf R. (eds), Engineering Societies in the Agent World II, LNAI 2203, Springer Verlag, 2001, pp160-174 et 2nd International Workshop, ESAW01, Prague, July 2001.
[Pnueli, 1986] Pnueli A., Specification and development of reactive systems. In Information Processing 86. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1986.
[Poslad & al., 2000] Poslad S., Buckle P., Hadingham R. - The FIPA-OS agent platform: Open Source for Open Standards published at PAAM2000, Manchester, UK, April 2000.

[Pree, 1998] Pree W., Design Patterns et architectures logicielles, Vuiberts, ISBN 2-7117-8611-0, 1998.
[Querrec & al., 2001] Querrec R., Deloor P. et Chevaillier P. "Environnement Virtuel pour la formation des officiers sapeurs-pompiers". JFIADSMA 2001 Journées francophones d'intelligence artificielle distribuée et systèmes multi-agents, pp 311 - 314, Montréal, Canada, 12 - 14 Novembre 2001.
[Quinqueton & Hamadi, 1999] Quinqueton J. et Hamadi Y., "Communication et Emergence : une épidémie chez les termites.", in actes des JFIADSMA'99 à Saint-Gilles, île de la Réunion, Gleizes J.-P., Marcenac P., Ed. Hermès, 1999.
[Redfield, 1994] Redfield J., La prophétie des Andes, édition Robert Laffont, ISBN 2-221087-24-0, 1994.
[Regis & al., 2002] STAFF, un système multi-agents adaptatif en prévision de crues - C.Régis, T.Sontheimer, P.Glize, M.-P.Gleizes – JFIADSMA 2002, France, Lille.
[Reinke & al., 1999] Reinke T., Glode D., Horn E., Kupries M., An Architecture Type-Based Development Environment for Agent Application Systems, 5th international Technion Symposium, April 1999.
[Rogalski & al., 1989] Rogalski J., Samurcay R., Baron M., Elaboration d'aides informatiques à la formation opérationnelle de responsables de la Sécurité Civile, Rapport PIRTTEM (87-89).
[Rosnay, 2000] Rosnay J.de, L'homme symbiotique – regards sur le troisième millénaire, nouvelle edition, Editions du Seuil, Paris, ISBN 202041399X, 2000
[Rougegrez, 1992] Rougegrez S., Raisonnement par cas appliqué aux feux de forets, Colloque jeunes chercheurs en IA, 1992.
[Rougegrez, 1994] Rougegrez S. Prédiction de processus à partir de phénomènes observés – le système REBECAS, Thèse de doctorat de l'université Paris 6, 1994.
[Routier & al., 2000] Routier J-C., Mathieu P. and Secq Y., Dynamic Skills Learning, rapport interne IT-2000-06, mai 2000. www.lifl.fr/SMAC
[Routier & al., 2001] Routier J-C., Mathieu P., and Secq Y., Dynamic skill learning: A support to agent evolution. In Proceedings of the AISB'01 SYmposium on Adaptive Agents and Multi-Agent Systems, pages 25--32, 2001.
[Sabas & al., 2002] Sabas A., Delisle S. and Badri M., A Comparative Analysis of Multiagent System Development Methodologies: Towards a Unified Approach, 3' International Symposium "From Agent Theory to Agent Implementation" (AT2AI-3), 6' European Meeting on Cybernetics and Systems Research, Vienne, Volume 2, 599-604,april 2002.
[Savall & al., 2001] Savall M., Itmi M. and Pecuchet J-P. : YAMAM – a new organization model for multi-agent systems and its platform named Phœnix – Conference SCSC 2001 – Orlando (USA) – 2001.
[Savall, 2001] Savall M., Comparaison de plates-formes SMA, rapport interne, laboratoire PSI, INSA de Rouen, Septembre 2001.
[Savall, 2002] Savall M., Etude comparative des méthodologies d'analyse des SMA, Rapport interne laboratoire PSI, INSA Rouen, août 2002.
[Schmidt, 1995] Schmidt D., Florinus : an Emergency Management, Information and Communication System for fire brigades, Police Forces and Civil protection. In proceedings of TIEMEC'95, Nice, May 1995.
[Schumacher & al., 1988] Schumacher M., Chantemargue F., Krone O. and Hirsbrunner B., STL ++: A Coordination Language for Autonomy-based Multi-Agent Systems, Internal working paper no 98-07 , IIUF-PAI, University of Fribourg (Switzerland), march 1998. http://www-iiuf.unifr.ch/pai/axe/AmocPapers/rep-march98/rep.html
[Searle, 1992] Searle J. La redécouverte de l'esprit, NRF Essais, Gallimard, 1992.
[Shoham, 1993] Shoham Y., Agent-oriented programming. Artificial Intelligence, 60(1):51--92, 1993.
[Silva & al., 2000] Silva L.M., Soares G., Martins P., Batista V. and Santos L., Comparing the Performance of Mobile Agent Systems: A Study of Benchmarking, the Journal of Computer Communications, Special Issue on Mobile Agents for Telecommunications, publication in January 2000.
[Smarr & Catlett, 1992] Smarr L. and Catlett C.E., "Metacomputing", communication of the ACM, vol.35, n°6, pages 45-52, juin 1992.
[Smith, 1980] Smith R., The Contract Net Protocol : High-level communication and control in a distributed problem solver, IEEE Transations on Computers, vol.C-29, no.12, pp.1104-1113, Décember 1980.
[Steels, 1994] Steels, L. Building agents with autonomous behavior systems, in L. Steels & R. Brooks, eds, `The `artificial life' route to `artificial intelligence'. Building situated embodied agents.', Lawrence Erlbaum Associates, New Haven, 1994.
[Sycara, 1989] Sycara K., Multiagent compromise via negociation, in distributed artificail intelligence, L.Gasser et M.Huhns (Ed), Pitman, 1989.
[Tardieu & al., 2000] Tardieu H., Rochfeld A. et Colletti R., La Méthode Merise : Principes et outils Organisation (Ed. organisation), ISBN : 2708124730, 2000.
[Tavares & Demazeau, 2002] Tavares da Silva J-L. and Demazeau Y., Vowels Coordination Model. In Cristiano Castelfranchi and W. Lewis Johnson, editors, 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS`02), pages 1129-1136, Bologna, Italy, ACM Press. 2002.

[Teller, 1992] Teller P., « A contemporary look at emergence », dans A.Beckermann, H.Flohr et J.Kim. « Emergence or reduction ? » Walter de Gruyter 1992, pp.139-153.
[Tisseau, 1998] Ballet P., Rodin V., Tisseau J., A multi-agent system to model and simulate in-vitro experimentations, Proceedings IIIS ISAS'98, Vol. 2, pages 1-7, Orlando (USA), July 12-16, 1998.
[Tranouez & al., 1999] Tranouez P., Durand S., Lesage F. et Cardon A., Représentation par des organisations d'agents des connaissances échangées dans un système d'information, JFIADSMA'99.
[Turing, 1950] Turing A.M., Computing Machinery and Intelligence, Mind 49, pp 433-460, 1950.
[Turing, 1983] Turing A.M., Les ordinateurs et l'intelligence, Pensées et Machines, coll. Milieux, Champ-Vallon, 1983.
[Tveit, 2001] Tveit A., A survey of agent-oriented software engineering - NTNU Computer Science Graduate Student Conference. Norwegian University of Science and Technology, May 2001.
[Tzafestas, 1995] Tzafestas E., Vers une systémique des agents autonomes : des cellules, les motivations et des perturbations, Thèse de doctorat, Université de Paris VI, 1995.
[VanAeken & Demazeau, 1998] VanAeken F. et Demazeau Y., Minimal Multi-Agent Systems, in ICMAS'98, Third International Conference on Multi-Agent Systems, Paris, France, 1998.
[Varela & Bourgine, 1992] Bourgine P. and Varela F. J., Introduction: Towards a practice of autonomous systems. In Varela and Bourgine, editors, Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life, pages xi--xvii, Cambridge, USA, 1992. MIT Press/Bradford Books. 112
[Varela & Maturana, 1994] Varela F., Maturana H. R., L'arbre de la connaissance: Racines biologiques de la cognition, 256 p, Addison-Wesley, 1994.
[Varela, 1989a] Varela F., Connaître les sciences cognitives : tendances et perspectives. Paris : Seuil, ISBN : 2-02-010474-1, 126 p, 1989.
[Varela, 1989b] Varela F., Autonomie et connaissance : essai sur le vivant, ISBN 2-02-010030-4, Seuil, 1989.
[Vauche & Guarnieri, 1997] Vauche D. et Guarnieri F., MCI : la main courante informatisée du CODIS06, rapport interne EMP Sophia-Antipolis, 1997.
[Versteeg, 1998] Steven Versteeg, Comparaison of Mobile Agent Toolkits for Java (draft), June 19, 1998.
[Walsh & al., 1999] Walsh, T.; Paciorek, N.; Wong, D., "Security and Reliability in Concordia", Mobility: Processes, Computers and Agents, ISBN: 0-201-37928-7, pp 525-534, Addison Wesley, April 1999.
[Weil-Barais, 2001] Weil-Barais A., L'homme cognitif, Presses universitaires de France, ISBN : 2 13 052383 8, 6ième édition refondue, 2001.
[Wong & al., 1997] D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M. Young, B. Peet, Concordia: An Infrastructure for Collaborating Mobile Agents, First International Workshop on Mobile Agents 97 (MA'97), Berlin, April 7-8, 1997. http://www.meitca.com/HSL/Projects/Concordia/
[Wood & DeLoach, 2000] Wood M. and Deloach S. An overview of the multiagent systems engineering methodology – AOSE'00 – LNCS n° 1957 – Springer Verlag, Berlin – January 2000.
[Wooldridge & al., 2000] Wooldridge M. Jennings N.R. and Kinny D., The Gaia Methodology for Agent-Oriented Analysis and Design, Journal of Autonomous Agents and Multi-Agent Systems 3(3), p.285-312, 2000
[Wooldridge & al., 2001] Wooldridge M., Weib G. and Ciancarini P., Agent-Oriented Software Engineering II, Second International Workshop, AOSE 2001, Montreal, Canada, May 29, 2001. Revised Papers and Invited Contributions, Springer Eds, 2001.
[Wooldridge & Ciancarini, 1999] Wooldridge M. and Ciancarini P., Agent-Oriented Software Engineering : the state of the art, Multi-Agent System Engineering (MAAMAW-99), 1999.
[Wooldridge & Ciancarini, 2000] Wooldridge M. and Ciancarini P., Agent-oriented software engineering : the state of the art – AOSE'00 - LNCS n° 1957 – Springer Verlag, Berlin – January 2000.
[Worms, 1992] Worms P., Sagesse – Feux d'Hydrocarbure – Acquisition de connaissance pour un système d'E.I.A.O, rapport de DEA, I.A.R.F.A, 1992.
[Wybo & al., 1998] Wybo J.L., DEDICS : a general framework for supporting management of forest fires, in proceedings of 3rd International Conference on Forest Fire Research, Coimbra, Portugal, 1998.
[Yang & al., 1998] Yang J., Havaldar R., Honavar V., Miller L. and Wong J. Coordination of Distributed Knowledge Networks Using Contract Net Protocol. In: Proceedings of the IEEE Information Technology Conference. Syracuse, New York. pp. 71-74., 1998.
[Yim & al., 2000] Yim M.H., Cho K., Kim J. and Park S. Architecture-centric object-oriented design method for multi-agent systems - ICMAS 2000.
[Zachary & al., 2001] Zachary W., Santarelli T., Ryder J., Stokes J., and Scolaro D. Developing a multi-tasking cognitive agent using the COGNET/iGEN integrative architecture. In 10th Conference on Computer Generated Forces and Behavioral Representation, (pp. 79-90). Norfolk, 2001. http://www.cognitiveagent.com/
[Zeghal, 1993] Zeghal, K. Un modèle de coordination d'actions réactives appliqué au trafic aérien, in journées francophones JFIADSMA, Toulouse, AFCET, 1993.

Glossaire

ACL (Action Communication Language) :
Un langage de description de message utilisé en particulier dans les spécifications FIPA.

CORBA (Communication Objet Request Broker) :
Architecture et technologie permettant l'échange d'objets et l'appel de méthodes distantes dans des environnements hétérogènes.

CPU (Central Processor Unit)
Puce électronique constituant le cœur de l'ordinateur.

DOM (Document Object Model) :
Interface indépendante d'un language de programmation ou d'une plateforme particulière, permettant aux scripts et aux programmes d'accéder dynamiquement au contenu d'un document (généralement XML). C'est une norme du W3C.

DTD (Document Type Definition) :
Description formelle du contenu d'un document XML. Les DTD sont utilisées pour valider le contenu et la forme des instances XML.

FIPA (Foundation for Intelligent and Physical Agent) :
Un groupe de travail spécialisé dans la spécification d'un modéle portable et complet de SMA.

GPRS (General Packet Radio Service) :
Ce service permet aux réseaux d'offrir une plus grande capacité, et d'utiliser des services comme la navigation internet en couleur, les emails, des communications visuelles, des messages multimédia…

IAD (Intelligence Artificielle Distribuée) :
Domaine de l'intelligence artificielle s'intéressant à la distribution des unités de connaissance. Les SMA sont une branche de l'IAD.

INRIA (Institut National de Recherche en Informatique et en Automatique) :
Etablissement public à caractère scientifique et technologique placé sous la double tutelle du ministère de la recherche et du ministère de l'économie, des finances et de l'industrie

INSA (Institut National des Sciences Appliquées) :
Ecole d'ingénieurs française. L'INSA héberge le laboratoire PSI qui a travaillé sur le modèle YAMAM.

JDK (Java Development Kit) :
Ensemble d'outils permettant le développement d'applications Java.

JVM (Java Virtual Machine) :

Machine virtuelle permettant d'exécuter des applications développées à l'aide de Java.

KQML (Knowledge Query Modelling Language) :
Langage d'interrogation de bases de connaissances.

ONTOLOGIE :
Du grec ôn, ontos, participe présent de einai, «être», et logos, «discours», «étude», «science». Autre nom de la métaphysique - entendue comme la discipline qui étudie l'Être en tant qu'être, indépendamment de ses déterminations particulières. Sens étendu par les bibliothécaires et les documentalistes, avec la construction des index, thesaurus, glossaires etc. Le vieux sens métaphysique du mot a été rafraîchi pour les besoins des systèmes informatiques complexes en représentation des connaissances et définition de catégories

PSI (Perception Systèmes Informations) :
C'est un des laboratoires informatiques hébergés par l'INSA de Rouen. Il est spécialisé dans la recherche appliquée sur les thèmes de la reconnaissance d'image, ou encore des interfaces hommes/bases de données.

RMI (Remote Management Interface) :
Technologie objet permettant l'appel de méthodes distantes entre des objets placés sur des plates-formes et des environnements hétérogènes. La technologie RMI permet au programmeur de faire abstraction des problèmes liés à la communication entres objets distants.

SAX (Simple API for XML) :
API originellement développée pour Java mais étendue à d'autres langages permettant de lire ou d'écrire un document XML. C'est devenu un standard depuis la version 2.0.

SL0 :
Langage de contenu de messages basé sur la sémantique des langages. Recommandation du FIPA.

SMA (Système Multi-Agents) :
Système composé d'un ensemble d'agents dont l'activité doit permettre l'émergence d'un résultat défini par l'objectif du système.

SOAP (Simple Object Access Protocol) :
Protocole d'échange d'informations dans un environnement décentralisé, distribué. C'est une norme du W3C et est utilisé par les Web Services.

TOY-PROBLEM :
Petit programme développé à l'aide de Phœnix destiné à mettre en évidence certaines particularités de la plate-forme. Certains toy-problems, tels celui des fourmis, sont devenus des références pour la comparaison de SMA.

UDDI (Universal Description, Discovery, and Integration)
Annuaire basé sur XML permettant de référencer des services. Utilisé notamment par les Web-Services.

UML (Unified Modeling Language) :
Formalisme de description de spécifications.

WEB-SERVICES :
Norme du W3C (World Wide Web Consortium) définissant des interfaces de programmation pour les communications inter-applications par Internet.

WSDL (Web Service Definition Language) :
Format XML décrivant des services réseaux offerts par les Web Services. C'est une norme du W3C.

XML (eXtended Modeling Language) :
Langage de descritpion de données basé sur un système de balises. C'est une norme du W3C. XPath, XSL et XPointer sont d'autres aspects liés à cette norme.

XML Schéma :
Modélisation à l'aide de XML d'une DTD d'instances de documents XML. En fait, métalanguage de XML. C'est une norme du W3C.

YAMAM (Yet Another Multi-Agent Model) :
Modèle théorique d'un SMA incluant entre autre la notion d'énergie. Phœnix est basée sur ce modèle.

Annexes

L’Extreme Programming

L’Extreme Programming (XP) est un ensemble de pratiques qui couvre une grande partie des activités de la réalisation d’un logiciel, planification, organisation d’équipe, échange avec le client, programmation… Cela n’a rien de révolutionnaire car la technique ne propose que des pratiques de « bon sens » comme la relecture de code, les livraisons fréquentes, la mise en place de tests automatiques… La nouveauté introduite par XP consiste simplement à pousser ces pratiques à l’extrême, comme son nom l‘indique, et à les organiser en un tout cohérent et parfaitement défini. XP s’adresse à des petites équipes de moins de dix personnes.

Le code est maintenu toujours aussi clair et simple que possible par une activité de remaniement permanente, soutenue par une batterie de tests automatiques ce qui permet d’y introduire de nouvelles fonctionnalités sans craindre des régression cachées.

L’équipe de développement comprend des membres qui travaillent toujours en commun et qui sont capables d’intervenir sur toutes les parties de l’application.

La gestion du projet a une démarche itérative qui permet à tous les intervenants de projet aussi bien l’équipe de développement que le client, d’améliorer continuellement son pilotage en s’appuyant sur l’expérience acquise en cours de route.

Ces trois points ne suffisent pas à expliquer l’intérêt croissant que suscite XP. Au-delà de la recherche purement cartésienne de l’efficacité, XP est guidé par un ensemble de valeurs humanistes, une vision positive du développement officiel fondés sur une conviction que l’efficacité s’obtient en prenant le temps de faire un travail de qualité et en donnant une phase de premier plan aux contacts humain – au sein de l’équipe de développement et entre l’équipe et le client.

Cette nouvelle approche XP est partie de la constatation des schismes entre les méthodes classiques et théoriques de conception de projet comme le Cycle en V ou les nombreuses variantes plus légères qui ont ensuite vues le jour incorporant des cycles itératifs. Dans toutes ces méthodes, le canevas d’un projet reste globalement le même puisqu’on cherche à définir très précisément ce que l’on souhaite avant de se lancer dans la réalisation proprement dite. Cette démarche a fait ses preuves dans le BTP mais pas vraiment dans l’industrie logicielle. Une étude américaine menée en 1994 sur un échantillon de 8000 projets a conclu sur le faire que seulement 16% d’entre eux ont été finalisés dans e temps et le budget initialement prévus et 32% de ces projets ont été interrompus en cours de route ! Le diagnostique de ces dérapages est souvent le même : après des spécifications et une conception poussée, le développement s’enlise par une remise en question tardive des choix initiaux qui oblige à revoir des pans entiers du modèle de conception. Il ressort de cette étude que les méthodes dites « classiques » sont en réalité tout à fait utopiques. Il est « normal » que des changements, des oublis ou des erreurs interviennent en fin de projet au moment de la phase de validation et de recette. Le développement logiciel un tant soit peu complexe est un exercice difficile dans la mesure ou les phases d’analyse et de conception constituent les premières étapes de a création d’un logiciel. C’est dans ces phases déterminantes que les différents intervenants en savent le moins sur le contexte précis du projet, y compris le client

(ou le donneur d'ordre) qui a souvent bien du mal à se faire une idée précise de ce à quoi pourra ressembler l'application quelques mois plus tard. Il est très difficile pour un client de spécifier ce qu'il souhaite. C'est souvent l'équipe de développement qui spécifie pour lui et lui fait valider les spécifications fonctionnelles et techniques écrites. Malgré cela, il y a parfois une différence entre ce que le client croit être une bonne spécification et ses besoins réels. C'est une fois que le client possède une première version du logiciel qu'il peut dire que cela lui convient ou pas. C'est aussi à ce moment qu'il peut remettre en question les spécifications initiales. Dans un cycle de vie classique en V, ces remises en question sont très coûteuses puisqu'une partie des efforts fondés sur les spécifications initiales est perdue. De plus ces remises en question représentent un risque pour la conception logicielle mise en place qui absorbera plus ou moins bien les adaptations nécessaires. Bien que les principes de développement des logiciels évoluent grâce à des méthodes et des outils comme UML, aucun changement significatif n'est apporté sur le déroulement du projet lui-même. L'XP est partie de la question suivante : la solution à ces problèmes de spécification et de conception passe-t-elle nécessairement par un effort supplémentaire de modélisation ou faut-il envisager d'adopter une approche radicalement différente qui accorde une importance moindre à cette phase initiale, dans de but de coller le plus près possible aux attentes du donneur d'ordre ?

Bibiographie

« extreme programming – avec deux études de cas » - collection objet de Eyrolles par J-L Bénard, L.Bossavit, R.Médina et D.Wiliams – ISBN 2-212-11051-0.

Le Format d'Echange de Connaissance

Le KIF (Knowledge Interchange Format) est un langage conçu pour l'échange de connaissance sur des systèmes hétérogènes. C'est une sorte de format de fichier pour faire des passerelles d'un format interne à un autre dans l'idée du format PostScript ou Acrobat. Il peut être utilisé pour passer une base de connaissance d'un système expert à un autre qui n'ont pas la même structure interne de représentation des données. Dans le cadre d'un système multi-agents, dont les agents sont cognitifs et utilisent des règles de productions sur un ensemble de connaissance, le KIF peut être intéressant pour échanger des données d'un agent à un autre.

Les automates cellulaires

La classe Environnement2D est découpée en cellules sur lesquelles, nous pourrons appliquer quelques travaux sur les automates cellulaires. Typiquement un réseau se compose

- De cellules (automates)
- De connexions d'activation
- De connexions d'inhibition

Pour une cellule i dont a_i représente le niveau d'activité à un instant précis, on a une connexion à ses voisins j par les valeurs c_{ij}. Les coefficients c_{ij} sont spécifiques à chaque relation. Si la relation est activatrice, on a $c_{ij} \geq 0$, si la relation est inhibitrice, on a $c_{ij} < 0$.

c_{ij} est nul pour deux cellules non interconnectées. On calcule le nouvel état de l'automate par

$$f_i(a) = \sum_j \left(c_{ij}.a_j\right) \text{ si } \left(a_{\min}, a_{\max}\right) \subset \sum_j \left(c_{ij}.a_j\right)$$

$$f_i(a) = a_{\min} \text{ si } \sum_j \left(c_{ij}.a_j\right) \leq a_{\min}$$

$$f_i(a) = a_{\max} \text{ si } \sum_j \left(c_{ij}.a_j\right) \leq a_{\max}$$

La stabilisation de ce réseau est réellement déterministe. Plus le nombre de neurones croit, plus le temps de calcul peut être grand. On est encore loin du neurone humain qui peut être connecté à 50.000 synapses. On peut aussi envisager l'emploi d'une méthode de stabilisation probabiliste (peut être un dérive de la machine de Boltzmann) ce qui introduit une forme de hasard dans le processus.

Dans tous les cas, l'existence de contraintes de micro-organisation au niveau local cumulée à une certaine stabilité de message d'entrée de chaque nœud du réseau provoque l'auto-organisation globale du réseau et sa stabilisation autour d'un attracteur stable grâce à une formidable intrication des interactions du réseau.

IL est vain de chercher à fiabiliser un système en évitant au maximum les bruits. Au contraire, la perturbation est une partie intégrante d'un système robuste et adaptatif.

Piaget définit l'intelligence et l'activité de la pensée comme la recherche permanente d'un équilibre. En outre, l'homme apprend plus aisément de manière harmonique, c'est à dire en assimilant la perturbation que de manière formelle.

Attracteur récursif de Varela : (mu est la perturbation et X est l'état du système)

$$X(n+1) = 1 - \mu.X(n)^2$$

Les systèmes connexionnistes

Les systèmes connectionnistes n'utilisent pas l'approche symbolique pour la résolution de problèmes. On suppose ici que l'intelligence peut émerger dans des systèmes composés d'entités simples et interactives à travers un processus d'apprentissage ou d'adaptation au cours duquel les connections entre les composants sont ajustées. L'approche conceptuelle n'est pas fondamentalement différente des SMA, mis à part que le traitement sera distribué au travers de couches de neurones.

La résolution de problème est parallèle dans le sens où tous les neurones des différentes couches traitent leurs entrées simultanément et indépendamment. Et c'est là que le bas blesse dans la mesure où cette notion de parallélisme n'est que théorique et quantitativement dans la limite du nombre de processeurs de la machine et de la qualité de l'OS ainsi que du quanta de chaque processus. Il ne peut s'agir que d'une simulation d'un environnement parallèle. Il est probable qu'on perde cette notion d'effet de bord présente dans le système réel. Puisque le

système informatique va traiter les neurones l'un après l'autre, et de ce fait, la distribution s'en trouve changée. Il est possible de sélectionner aléatoirement le traitement de chaque neurone mais on ne se rapproche pas pour autant du fonctionnement du système réel. La première question est donc de savoir si la simulation peut fournir un résultat suffisamment réaliste. L'effet de bord peut être nuisible dans certains cas et bénéfique dans d'autres. Je pense qu'il n'est pas nécessairement souhaitable d'éliminer toute trace de désordre apparent. Au contraire peut être est ce la présence du chaos qui permet au système de tendre vers un certain point d'équilibre, ou au contraire de le déséquilibrer pour réorganiser sa morphologie. La présence du chaos fractal est alors souhaitable.

Dans le modèle connexionniste, on ne manipule pas les symboles comme des symboles. En réalité, on va construire une architecture qui va, lors de son apprentissage (et même lors de son fonctionnement), capturer les invariants de l'objet analysé. Le réseau se structure donc autour d'un attracteur sans avoir été explicitement programmé pour reconnaître l'objet en question. L'avantage de ce type de système est sa plasticité et sa robustesse. En effet, si le réseau est très largement attiré vers un point d'équilibre, on considère que le réseau est stable et de ce fait, ne sera pas facilement déstabilisé par un nouvel élément perturbateur.

Les tâches pour lesquelles l'usage d'un réseau neuro-mimétique est approprié sont :

- la classification
- la prédiction
- le filtrage de bruit
- la reconnaissance de forme
- l'optimisation

C'est précisément dans ce genre de problème que les modèles symboliques sont peu efficients. Bien souvent le problème n'est pas séparable linéairement, pour cela on peut utiliser l'algorithme de rétro-propagation qui permet de répartir « la responsabilité » d'une erreur dans les différents nœuds des systèmes à couches cachées.

Dans les réseaux de Kohonen ou de Hecht-Nielsen, des vecteurs de « poids de réseau » sont utilisés plutôt que des forces de connections pour représenter des formes. A noter que ces formes ne nous sont pas toujours intelligibles. L'algorithme d'apprentissage non-supervisé « winner-take-all » sélectionne le nœud pour lequel la forme des poids est la plus proche du vecteur d'entrée et l'ajuste de façon à ce qu'il soit encore plus proche. La combinaison des couches de Kohonen et de Grossberg (1982) forme un réseau nommé apprentissage par contage de propagation ; il s'agit d'un modèle d'apprentissage par stimulus-réponse.

Il en existe deux formes : l'apprentissage supervisé et non supervisé.

L'apprentissage supervisé est caractéristique de tâches telles que la mémorisation. Il consiste à utiliser un certains nombres de couples d'entrée-sortie et un algorithme forçant la convergence des poids de telle sorte que la présentation de chacun des objets d'entrée de l'apprentissage donne en sortie l'objet qui lui était associé. Ce type de réseau est utilisé pour la reconnaissance de formes complètes ou brouillées. Les algorithmes utilisés pour déterminer les poids W_i suivent souvent la règle de Hebb. Le principe est simple : chaque fois qu'un neurone contribue au déclenchement d'un autre neurone, le force du chemin entre ces deux

neurones est augmentée. L'apprentissage de Hebb est un algorithme assez simple pour ajuster le poids des connexions.

Il existe un autre type de réseau intéressant nommé le réseau par attracteur. Ce type de réseau utilise un feedback sur les connexions pour faire circuler un signal dans le réseau. Tout objet « appris » est un point d'attraction de la dynamique du cerveau. La sortie du réseau est prise en considération lorsque le réseau s'est stabilisé autour d'un point d'équilibre.

Les poids du réseau sont construits afin qu'un ensemble d'attracteurs soit créé. La forme en entrée passe dans un bassin d'attraction et l'équilibre est atteint véritablement sur cet attracteur. Les attracteurs de ce type de réseaux neuronaux sont utilisés pour stocker des formes en mémoire. Les informations sont souvent codées de façon redondante, ce qui rend le réseau relativement peu sensible aux lésions. Le physicien John Hopfield (1982) définit une classe de réseaux d'attracteurs pour lesquels la convergence peut être représentée en terme de minimisation d'énergie. Il est possible de résoudre le problème célèbre du voyageur de commerce avec cette méthode.

L'apprentissage supervisé est plutôt employé pour les tâches de classement, c'est à dire le tri d'objets et leur répartition en classes. Chaque cellule du réseau va encoder pour une classe et est en compétition avec les autres. La cellule dite gagnante détermine la classe de l'objet en entrée. L'information n'intervenant pas dans le classement est éliminée pour discriminer au mieux les objets présentés au réseau. Il n'y a donc pas de notion de redondance dans ce type d'apprentissage. Ce type de réseau utilise peu ou prou la règle de Hebb (voir algorithme de Oja) ou alors d'un point de vue globale l'utilise entre l'entrée et la sortie de façon à corréler chaque sortie avec le contenu du signal. Entre les neurones de sortie, il y a au contraire des modifications anti-hebbiennes de façon à imposer un codage pour différentes caractéristiques du signal. Ce type de réseau est adéquat pour le codage et la compression de données.

Un neurone artificiel est :

- Un signal d'entrée x_i qui peut provenir de l'environnement. Cette entrée peut aussi venir de l'activation d'un autre neurone. Souvent l'entrée sera discrète, d'un ensemble {0,1} ou {-1,1}, ou un nombre réel.
- Un ensemble de nombres réels qui constituent les poids w_i. Ces poids représentant la force de la connexion.
- Un niveau d'activation $\sum w_i.x_i$. Le niveau d'activation est déterminé par le cumul des forces du signal d'entrée x_i pondéré par son poids w_i.
- Une fonction de seuil f(net) qui détermine l'état de sortie du neurone en fonction du niveau d'activation . L'état de sortie est déterminé selon que le niveau d'activation soit au dessus ou au dessous d'un certain seuil. Cette fonction seuil est sensée produire l'état de sortie on/off du neurone. On note

$$net_i(t) = \sum_{j=1}^{n} w_{ij}(t).o_j(t)$$

$net_i(t)$ représente le signal à l'entrée du neurone i et $o_j(t)$ représente la sortie du j ^{ième} neurone du réseau. Le terme $w_{ij}(t)$ représente la valeur pondérée associée à la connexion qui se trouve entre le neurone n°j et le neurone n°i. La valeur n correspond au nombre d'autres neurones connectés à l'entrée du neurone n°i.

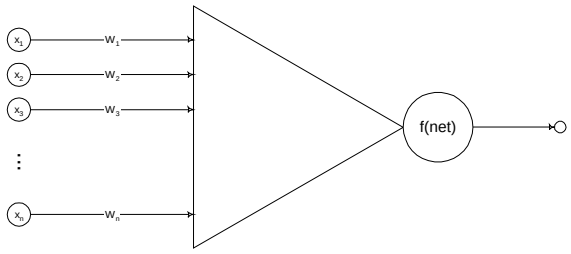


Figure 4 - Schéma d'un neurone artificiel appelé également *nœud* ou *unité*

Un réseau de neurones est donc un ensemble de neurones basés sur ce modèle. A cela s'ajoute un certain nombre de propriété telles que :

- La topologie du réseau, qui est la forme des connexions entre ces neurones individuels.
- L'algorithme d'apprentissage.
- L'interprétation des données.

Plusieurs principes de fonctionnement existent. Hebb pensait que des stimuli répétés sur une synapse augmentaient sa sensibilité et sa probabilité future de déclenchement. Un stimulus répété provoque une activité dans un groupe de cellules, ces cellules deviennent de plus en plus fortement associées, ce qui fait qu'un stimulus aura tendance à exciter plus ou moins tous ces neurones associés et de ce fait, assurerait la reconnaissance du stimulus. Des psychologues des années 90 ont tentés de recréer le modèle de Hebb mais n'ont pas obtenu de résultats satisfaisants sans l'ajout de mécanismes d'activation et d'inhibition.

Le perceptron :

Vers 1958, Frank Rosenblatt proposa un algorithme d'apprentissage pour un réseau à simple couche nommé le « perceptron ». Les valeurs d'entrée et les niveaux d'activation étaient soit –1 ou 1. Les poids étaient des nombres réels. Le perceptron calculait sa valeur de sortie avec les règles : 1 si $\sum w_i.x_i \geq \theta$ et -1 si $\sum w_i.x_i < \theta$.

Entraînement du perceptron
Le perceptron utilise une méthode d'apprentissage supervisé très simple. Le perceptron essaie de résoudre le problème, puis « l'instructeur » lui donne la bonne réponse. Le perceptron change alors les poids de façon à réduire l'erreur. Appelons C la constante qui représente le taux d'apprentissage et d la valeur de sortie désirée. L'ajustement du poids du composant i du vecteur d'entrée se fait est réalisé par le calcul suivant :

$$\Delta Wi = c\left(d - sign\left(\sum x_i.w_i\right)\right).x_i$$

$sign\left(\sum x_i.w_i\right)$ étant la valeur de sortie du perceptron (+1 ou –1). Ainsi la différence entre la sortie désirée et la sortie actuelle sera 0, +2 ou –2.

3 cas se présentent :
- si la sortie désirée est égale à la valeur de sortie actuelle : ne rien faire
- si la sortie actuelle est –1 et devrait être 1 alors incrémenter le poids de la ligne i de $2.c.x_i$.
- Si la sortie actuelle est 1 et devrait être –1, décrémenter le poids de la ligne i de $2.c.x_i$.

Nils Nilson en 1965 critiqua le perceptron en arguant qu'il ne pouvait résoudre certains classes de problèmes, ceux ou les données n'étaient pas linéairement séparables. Plusieurs améliorations ont été apportées au perceptron tel que le multi-couches mais Seymour papert et Marvin Minsky en 1969 prouvaient que le problème de la séparation linéaire ne pouvait pas être surmonté quelque soit le type du perceptron. Un exemple de problème non linéairement séparable est la table de vérité du ou-exclusif.

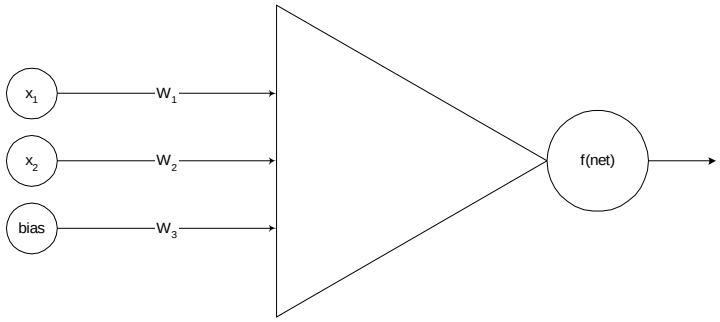


Figure 5 - Schéma d'un neurone artificiel avec biais

Rappelons que pour un espace à n dimensions, une classification est linéairement séparable si ses classes peuvent être séparées par un hyperplan à n-1 dimensions. Après cette constatation, les recherches vers les méthodes connectionnistes ont très largement diminué en se redirigeant vers une approche symbolique.

La loi Delta

Les fonctions de seuil.
On peut utiliser une fonction de seuil très simple comme une limite bipolaire, ou utiliser d'autres fonctions d'activation comme une sigmoïde, ce qui permet des algorithmes d'apprentissage plus fin en réduisant l'erreur possible.

La fonction de seuil sigmoïdale s'écrit : $f(net) = \frac{1}{\left(1 + e^{-\lambda * net}\right)}$ où $net = \sum x_i . w_i$

λ est la pente i.e. le paramètre « d'écrasement » de la sigmoïde. Plus λ est grand, plus la sigmoïde se rapproche de la fonction de seuil linéaire bipolaire. Cette fonction est utilisée car elle présente un certain nombre d'avantages : de forme sigmoïdale, elle est continue et donc dérivable en tout point. En outre, sa dérivée est plus grande là où la fonction sigmoïdale change rapidement, ce qui fait qu'il est plus facile de localiser l'erreur sur les nœuds où l'activation est la moins certaine. Sa dérivée se calcule assez rapidement :

$$f'(net) = f(net).(1 - f(net))$$

Si on introduit le biais dans cette fonction (le biais θ est le déplacement de la courbe le long de l'axe des x, c'est donc le point de transition entre deux états), l'équation ressemble à :

$$f(net_i(t)) = \frac{1}{\left(1 + e^{-(net_i(t)-\theta)/\lambda}\right)}$$

Mais dans la plupart des réseaux de neurones, ce paramètre λ est fixé à 0. Dans la fonction ci-dessus, plus θ est petit, plus la fonction ressemble à une fonction d'activation bipolaire. Pour certaines applications de réseau de neurones, il se peut qu'une autre fonction soit utilisée comme fonction de seuil telle que :

$f(net_i(t)) = \frac{1}{2}.(1 + \tanh(\mu.net_i(t))$ μ étant le gain (souvent déterminé empiriquement).

Parfois même, dans des systèmes de classification on peut utiliser une fonction gaussienne :

$f(net_i) = e^{\frac{net_i - 1}{\sigma^2}}$, ici σ est le « smoothing parameter ».

La règle Delta est basée sur la notion d'erreur de surface. L'erreur de surface représentant l'erreur cumulée d'un ensemble de données comme une fonction des poids du réseau. Chaque configuration possible des poids est représentée par un point sur cette « surface d'erreur ». L'objectif alors de l'apprentissage est de trouver la direction sur cette surface qui réduit l'erreur le plus rapidement. Cette méthode est appelée *descente du gradient* car le gradient est une mesure de la pente, représenté par une fonction de direction d'un point sur une surface. L'utilisation de la méthode de descente du gradient est tout à fait analogue à une technique de minimisation d'erreur. Afin d'utiliser cette règle Delta, le réseau doit utiliser une fonction d'activation qui soit continue et donc dérivable.

$$c(d_i - O_j) f'(net_i).x_j$$

C est une constante permettant de paramétrer le taux d'apprentissage. d_i et O_i sont la valeurs de sortie désirée et réelle pour le nœud i. xj est l'entrée j du nœud i. f' est la dérivée de la fonction d'activation pour le nœud i. L'erreur est donc donnée par la formule :

$$Error = \frac{1}{2}.\left(\sum_i (di - Oi)^2\right)$$

L'intérêt de mettre au carré consiste à ce que la somme de valeurs d'erreur individuelles, négatives et positives, ne s'annulent pas.

Calcul de l'erreur globale pour chaque nœud

Sur la couche de sortie, il est intéressant de mesurer le taux de changement de l'erreur du réseau en respectant la sortie de chaque nœud. Pour cela, il nous faut regarder le taux de changement d'une variable particulière d'une fonction à plusieurs variables. La dérivée partielle de l'erreur totale est :

$$\frac{\delta Error}{\delta O_i} = \frac{\delta\left(\frac{1}{2}\right).\sum (d_i _ O_i)^2}{\delta O_i} = \frac{\delta\left(\frac{1}{2}\right).(d_i _ O_i)^2}{\delta O_i}$$

Il est en effet possible de simplifier le Sigma dans la mesure où on considère un nœud sur la dernière couche du réseau ; l'erreur ne va donc pas affecter d'autres nœuds. La dérivée de cette quantité nous donne :

$$\frac{\delta\left(\frac{1}{2}\right).(d_i _ O_i)^2}{\delta O_i} = -(d_i - O_i)$$

Calcul de l'erreur sur le poids du nœud
Ce qui nous intéresse maintenant est de rapporter le taux de changement de l'erreur du réseau au changement de poids W_k du nœud i.

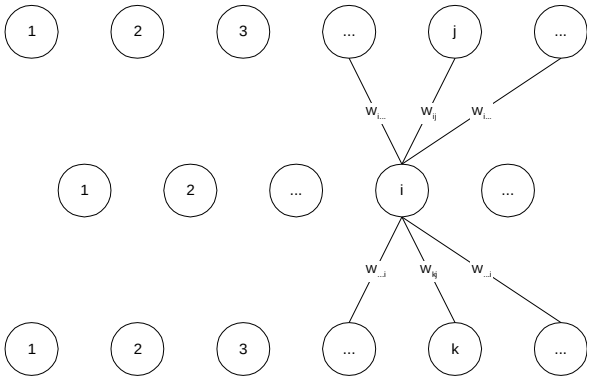
$$\frac{\delta Error}{\delta W_k} = \frac{\delta Error}{\delta O_i}.\frac{\delta O_i}{\delta W_k} = -(d_i - O_i).\frac{\delta O_i}{\delta W_k}$$

or $\frac{\delta O_i}{\delta W_k} = x_k.f'(W_i X_i)$ donc $\frac{\delta Error}{\delta W_k} = -(d_i - O_i) f'(net_i) * x_k$

La minimisation de l'erreur nécessite que les changements de poids soit dans la direction de la partie négative du gradient, donc il vient :

$$\Delta W_k = -c.\frac{\delta Error}{\delta W_k} = c(d_i - O_i) f'(net_i).x_k$$

Le problème de l'apprentissage Delta est qu'il risque de tomber dans un minimum local. La constante c paramétrant le degré d'apprentissage est importante pour la performance de la règle delta. Plus la valeur de c est grande plus rapidement les poids vont tendre vers la valeur optimale. Mais si c est trop grande, l'algorithme va dépasser le minimum et osciller entre des valeurs de poids optimum. Des valeurs de c plus petites ne sont pas sujettes à ce problème mais ne permettent pas à l'algorithme d'apprendre rapidement. Il faut donc paramétrer c en fonction de l'application pour laquelle le réseau de neurones est implémenté.



L'apprentissage par rétro-propagation

Dans un réseau multi-couches, les neurones sont connectés en couches, les unités de la couche k transmettant aux neurones de la couches k+1. Ainsi, un signal d'erreur se propage dans le réseau et évolue dans une voie qui peut être extrêmement complexe et souvent inattendue, ce qui fait que l'analyse d'une source d'erreur sur la couche de sortie est rendue très difficile. L'intérêt de l'algorithme de rétro-propagation est de répartir cette responsabilité et d'ajuster les poids en conséquence. L'approche est simple : on démarre sur la couche de sortie et on propage l'erreur au travers des couches cachées. Cette technique est certainement la plus utilisée dans le monde du réseau neuronal.

La rétro-propagation utilise la généralisation de la loi Delta et utilise aussi l'approche de la descente du gradient mais nous allons regarder la contribution des nœuds des couches cachées pour l'erreur à la couche de sortie.

Afin de calculer l'ajustement du poids k du nœud i, nous utilisons les formules :

$\Delta W_{ik} = -c(d_i - O_i).O_i(1-O_i).x_{ik}$ pour les nœuds de la couche de sortie

$\Delta W_{ik} = -c.O_i(1-O_i).\sum(-Delta_j.w_{ij}).x_{ik}$ pour les nœuds des couches cachées.

j est l'index du nœud dans la couche suivante pour lequel le signal du nœud i de la couche courante se déploie.

$$Delta_j = -\frac{\delta Error}{\delta net_j} = (d_i - O_i).O_i(1-O_i)$$

Taux de changement de l'erreur réseau pour le nœud i en fonction de W_k.

$$\frac{\delta Error}{\delta W_k} = -(d_i - O_i)f'(net_i) * x_k = -(d_i - O_i).O_i(1-O_i).x_k$$

$$\frac{\delta Error}{\delta O_i} = -Delta_j.\frac{\delta net_j}{\delta O_i}$$

La rétro-propagation présente bon nombre de défaut, le plus grand étant la possibilité de tomber dans un minimum local. De plus le processus est assez long à calculer surtout si le réseau converge lentement.

Représentation d'un système visuel :

Il peut être représenté par n récepteurs (n entrées) dont le stimulus d'entrée est filtré pour fournir en sortie une information codée, c'est à dire une représentation neuronale de l'objet d'entrée. Voir les travaux de Barlow.

Une tentative simple due à Oja propose un algorithme non supervisé pour un réseau simple à un neurone de la forme :

$$S(t) = \sum w_i(t).x_i(t) \quad \text{et} \quad w_i(t+1) = w_i(t) + S(t).x_i(t) - S^2(t).x_i(t)$$

le premier terme est du type Hebbien, le second borne la norme du vecteur d'efficacité synaptique (vecteur W). Cet algorithme détermine l'axe de la première composante principale de l'ensemble des données d'entrée.

La fidélité du codage est décrite par la dépendance statistique entre l'entrée et la sortie du réseau. Cette dépendance est appelée information mutuelle.

Le terme entropie désigne « la quantité de désordre », c'est à dire la quantité d'information. L'information mutuelle est l'expression d'une différence d'entropie I=S-S1-S2. Par exemple dans un milieu contenant x_1 objets de type 1 et x_2 objets de type 2, l'entropie est :

$$S = \ln\left(\frac{(x_1 + x_2)!}{x_1!x_2!}\right)$$

Dans le cadre d'un réseau de neurones en charge de classer p objets répartis en 2 classes : si le tri est parfait, l'entropie finale est nulle, sinon l'entropie finale est non nulle avec I<S. Plus l'information mutuelle est importante, plus l'efficacité du réseau est bonne. Si on a une distribution continue des objets d'entrée, l'information mutuelle s'exprime avec :

$$I(E,S) = \sum p(E,S).\ln\left(\frac{p(E,S)}{P(E).q(S)}\right)$$

Une technique nouvelle consiste à faire évoluer les poids à l'aide d'algorithmes génétiques.

Apprentissage par renforcement : la voie vers la spécialisation

Technique classique en réseaux de neurones, elles est aussi utilisée pour les agents adaptatifs dans les SMA. Cette technique est basée sur l'idée que la tendance à produire une action doit être renforcée si elle produit des résultats favorables et fragilisée dans le cas contraire. Ainsi des agents initialement totipotents peuvent se spécialiser progressivement dans l'accomplissement d'une tâche déterminée. L'agent devient donc de plus en plus performant pour résoudre une tâche très particulière. Ce principe a été démontré par A.

Drogoul qui montra qu'une société spécialisée tendait à devenir plus efficace qu'un ensemble d'agent totipotents.

On la nomme apprentissage de stratégies par interactions. Ceci permet d'optimiser de façon extraordinaire une future simulation. C'est aussi un agent qui sera chargé d'observer le comportement des autres agents et de dégager une stratégie. L'apprentissage peut être long. Il permet en quelque sorte d'extraire des attracteurs issus d'une simulation qui peut être chaotique.

Un joueur améliore sa technique en jouant énormément et si possible avec des adversaires différents et toujours meilleurs. L'idée n'est pas de trouver une technique de résolution efficace pour un type particulier de problème mais de construire une architecture auto-adaptative permettant de se perfectionner elle-même en observant de façon continue les actions de son ou ses adversaire(s) qui possèdent eux-mêmes un objectif de victoire. Ce comportement typiquement cognitif est donc fondamentalement axé sur l'étude de son « collègue » et néanmoins ennemi !.

Les agents doivent donc choisir entre exploiter leur connaissance ou d'explorer d'autres alternatives afin d'accroître leur connaissance pour une situation future. Ce « tradeoff » est connu dans la théorie de la décision comme le dilemme de l'exploration contre l'exploitation. Bien sur, l'exploration de nouvelles possibilités et donc l'augmentation potentielle de connaissance a un coup non négligeable. Et réciproquement les agents qui s'engagent dans une exploitation des résultats précédents risque d'appliquer une solution non-optimale. Plusieurs méthodes furent développées pour incorporer des stratégies d'exploration dans les agents RL (Reinforcement Learning). Les méthodes indirectes qui incorporent le hasard dans la prise de décision en assignant à chaque action une probabilité positive d'être sélectionnée et exécutée. Les méthodes directes, qui utilisent l'historique d'interaction pour évaluer la contribution attendue de chaque action de l'exploration. Les méthodes d'exploration actuelles ne prennent pas en considération le risque impliqué dans l'exploration. C'est un problème non négligeable qui méritera d'être soulevé.

Dans le cadre d'une simulation, il est possible d'implémenter des mécanismes de renforcements de sorte qu'une agent ayant accompli une tâche ait tendance à choisir une tâche du même type ultérieurement. Pour cela un coefficient pondérateur peut être utilisé de façon à ce que l'agent semble plus spécialisé. Il pourra ainsi se diriger et s'intégrer à un groupe et y jouer un rôle. Cette forme de spécialisation typiquement artificielle n'est cependant pas une spécialisation en tant que telle puisque changer la valeur d'un attribut d'un objet n'a jamais permis à cet objet d'acquérir de la connaissance ou d'affiner son état réel. La spécialisation implique une certaine notion de cognition que nous traduiront dans notre application par un « réseau de neurone dans la tête de l'agent. On ne parlera pas bien entendu de puissants algorithmes de conscience permettant de prendre une décision. Ce réseau est uniquement comportemental, et devrait lui permettre d'affiner un choix originellement réactif.

Optimisation théorique du système

A partir de règles simples, et un certain phénomène d'émergence, un SMA peut manifester un fonctionnement étonnant et trouver une solution à un problème. Cette solution n'est pas toujours optimale et on observe souvent un temps d'exécution et/ou de communication prohibitif. Le problème se pose mois dans un système réel que dans un système simulé puisque le premier fonctionne de façon massivement parallèle.

Il est intéressant de réaliser un apprentissage, d'observer et de profiter des résultats de l'émergence et d'en tirer des lois de fonctionnement statistiques ou une fonction mathématique plus optimisée pour réduire les erreurs d'aléa du système. Afin d'améliorer les

performances globale du système, des recherches devraient être menées afin de tenter de déterminer une fonction dynamique permettant de :

maximiser la multiplication générale des agents ce qui tend à améliorer la robustesse

maximiser la spécialisation et donc favoriser le regroupement ce qui tend à améliorer la performance

minimiser les conflits et le flux de messages de négociation échangé ce qui permet d'accroître la rapidité de la simulation.

Cette fonction devrait être évaluée en temps réel et pouvoir échapper à des minima locaux. Pour cela, devra-t-on sans doute envisager des techniques du type du recuit simulé.

Quelques exemples d'algorithmes

L'observation du comportement des fourmis a permet d'écrire des algorithmes simples mettant en exergue des propriétés émergentes.

Algorithme de fourragement et de récupération de nourriture

Comportement A :
Si pas de nourriture trouvée alors
    Choisir une direction au hasard
    Avancer pendant quelques temps dans cette direction
    Changer de direction

Comportement B :
Si forte odeur trouvée au sol pendant le déplacement alors
    Suivre la trace jusqu'à la nourriture
        Ou jusqu'à la fin de la trace (plus de nourriture)

Comportement C :
Si nourriture trouvée alors
    Emettre une quantité de phéromones à l'endroit de la nourriture
    S'agiter un peu sur place pour être vue par d'autres fourmis à proximité
    Prendre un morceau de nourriture et aller au nid en laissant une trace de phéromone au sol
    Revenir sur ses pas (mémorisation du chemin) un fois ressortie du nid

Comportement D :
Si on a passé trop de temps dehors sans rien trouver, alors revenir au nid.

Algorithme de récupération de nourriture

Si la fourmi n'est pas chargée de nourriture
    Faire la détection d'odeur 6 pixels à la ronde,
    Voir s'il y a trace de phéromones
    Ou s'il y a une fourmi qui s'agite
    Ou une odeur de nourriture
Effacer toutes les fourmis de l'écran
Faire avancer toutes les fourmis d'un pixel
Si une fourmi est chargée alors
    Déposer phéromones
Si nouveau pixel = phéromone et si fourmi non chargée alors

```
        Vider pixel
        Charger fourmi
        Tracer en mémoire le chemin nourriture-nid
Si la fourmi n'a pas trouvé de nourriture et est arrivée à la fin de son segment initial alors
        Tirer au hasard un nouveau segment que la fourmi devra parcourir
Dessiner toutes les fourmis
```

Algorithme de regroupement des miettes

```
Effacer toutes les fourmis
Pour chaque fourmi tester si la présence d'une miette
Si présence d'une miette alors
        Si la fourmi ne transporte pas déjà une miette alors ramasser la miette
Continuer à avancer jusqu'au bout du segment tiré au hasard
Si la fourmi est chargée alors
        Déposer sa charge
        Changer de direction
Sinon
        Avancer
Lorsqu'on est au bout du segment
        Tirer au hasard un autre segment
Une fois qu'on a étudié l'environnement de chaque fourmi
        Redessiner les miettes et les fourmis
```

Le Chaos

Le chaos peut être présent à plusieurs niveaux. Voici quelques exemples :

Croissance d'une population

On peut observer la croissance d'une population comme un système chaotique. Les écologistes ont proposé différents modèles pour estimer le comportement d'une population dans le temps. Un des plus simples est le suivant : on calcule la population de l'espèce avec la formule $P_{n+1}=k.P_n(1-P_n)$ où k est une constante qui dépend des conditions écologiques telles que la quantité de nourriture et P_n est le pourcentage de population après n générations ($0 \leq P_n \leq 1$). On initialise P_0 à 0.5. Quand k est petit, le destin de la population semble tragique ; pour k=0.5 la population meurt très vite tandis que pour k=1.2, 2, 2.7 les chiffres tendent à se stabiliser vers une valeur limite. Au dessus de 3, les différentes valeurs de k entraînent des résultats étonnants : pour k=3.1 les valeurs limites oscillent entre deux valeurs distinctes, pour k=3.4, les valeurs limites oscillent entre 4 valeurs. Si k=4, il n'y a rien de notable. Une valeur initiale fixée à 0.5 conduit à la disparition de l'espèce en seulement deux générations, tandis qu'une valeur initiale fixée à 0.4 produit une variation extrêmement aléatoire de la population.

Pn+1=kPn(1-Pn)							
k							
0,5	**1,2**	**2**	**2,7**	**3,1**	**3,4**	**4**	**4**
0,500	0,500	0,5	0,500	0,500	0,500	0,400	0,5
0,125	0,300	0,5	0,750	0,775	0,850	0,960	1
0,055	0,252	0,5	0,563	0,540	0,434	0,154	0
0,026	0,226	0,5	0,738	0,770	0,835	0,520	0
0,013	0,210	0,5	0,580	0,549	0,469	0,998	0
0,006	0,199	0,5	0,731	0,768	0,847	0,060	0
0,003	0,191	0,5	0,590	0,553	0,441	0,025	0
0,002	0,186	0,5	0,726	0,766	0,838	0,099	0
0,001	0,181	0,5	0,597	0,555	0,461	0,358	0
0,000	0,178	0,5	0,722	0,766	0,845	0,919	0
0,000	0,176	0,5	0,603	0,556	0,446	0,298	0
0,000	0,174	0,5	0,718	0,765	0,840	0,837	0
0,000	0,172	0,5	0,607	0,557	0,457	0,547	0
0,000	0,171	0,5	0,716	0,765	0,844	0,991	0
0,000	0,170	0,5	0,610	0,557	0,448	0,035	0
0,000	0,170	0,5	0,713	0,765	0,841	0,135	0
0,000	0,169	0,5	0,613	0,557	0,455	0,466	0
0,000	0,168	0,5	0,711	0,765	0,843	0,996	0
0,000	0,168	0,5	0,616	0,557	0,450	0,018	0
0,000	0,168	0,5	0,710	0,765	0,851	0,070	0
0,000	0,168	0,5	0,618	0,557	0,455	0,261	0
0,000	0,168	0,5	0,708	0,765	0,843	0,773	0

Question : Le modèle n'est il pas adapté à ce type de problème ou le comportement de la population tendrait-il vers celui modélisé ?

Une liste d'itérés successifs d'un point ou d'un nombre est appelée « Orbite ». Prenons un exemple : Soit S(x)=sin(x). En suivant la notation $F^2(x)=F(F(x))$ et non $(F(x))^2$.

$S(123)=-0.459...$
$S^2(123)=-0.443...$
$S^3(123)=-0.429...$
...
$S^{299}(123)=-0.097892...$
$S^{300}(123)=-0.097580...$

Finalement, de manière plus générale $S^n(x) \rightarrow 0$ quand $n \rightarrow \infty$ ($\forall x$). L'orbite de 123 est une séquence de nombres qui tendent vers 0. En fait, quelque soit la valeur initiale, l'orbite tendra vers 0.

Supposons que notre étude porte sur un système dynamique représentant un processus physique dont nous devons prévoir la phase terminale. Une fois votre système initialisé, vous constaterez quelques petites erreurs de votre modèle par rapport aux observations du système réel. Si l'orbite résultante est stable, il y a de fortes chances que les petites erreurs rencontrées n'altèrent pas le comportement global de votre système. Si pour une valeur d'initialisation x de votre système très proche d'une autre valeur d'initialisation y, on obtient une orbite sensiblement différente, on parlera d'orbite instable. La plupart des systèmes dynamiques possèdent un vaste ensemble de conditions initiales pour les lesquelles les orbites sont instables. L'ensemble de ces points pour lesquels les orbites sont instables est appelé l'ensemble chaotique. C'est ici que les fractales entrent dans le champ des systèmes dynamiques : les ensembles de points pour lesquels les orbites sont instables forment une fractale. On peut donc donner une définition d'une fractale : c'est un ensemble chaotique d'un système dynamique.

Attracteur de Héron

Les systèmes dynamiques possèdent des attracteurs. Considérons le cas d'un système possédant deux variables. En 1974 l'astronome Héron rencontre l'objet mathématique suivant :

$$x_{n+1}=1+y_n-Ax_n^2 \text{ et } y_{n+1}=Bx_n.$$

Les paramètres A et B seront fixés respectivement à 1.4 et 0.3. Selon les valeurs qu'on donnera à x0 et à y0, lorsqu'on lance l'itération, Héron démontra qu'on ne peut obtenir que deux résultats possibles ; soit l'orbite tend vers l'infini ou tend vers une structure très particulière appelée l'attracteur de Héron.

Attracteur de Gingerbreadman

C'est un autre exemple d'ensemble chaotique dans le plan. Le système dynamique est donné par les relations :

$$x_{n+1}=1-y_n+|x_n| \text{ et } y_{n+1}=x_n$$

C'est chaotique dans certaines régions et stable dans d'autres.

La théorie du chaos fractal consiste à montrer que l'ordre peut émerger à partir de situations semblant désordonnées et loin de l'équilibre. Le biologiste F. Varela a travaillé sur une recherche d'attracteur fractal (l'attracteur qui porte son nom : attracteur de Varela), qui, modélisé par d'autres chercheurs présentait à partir de conditions initiales aléatoires une auto-organisation formée de grandes cellules polygonales (phénomène apparemment analogue à celui des tourbillons de Benard).

Les automates cellulaires

Lorsqu'il s'agit de simuler le comportement d'un système, il existe plusieurs techniques dont les automates cellulaires. Selon Fredkin [Toffoli et Margolis, 1987] et [Wolframm, 1984], « tout processus peut être décrit comme une structure d'automates cellulaires ».

Les modèles de la vie artificielle tentent de reproduire les hypothèses et les conditions d'émergence de certains phénomènes en simulation, ce qui permet en retour une meilleure compréhension de ces phénomènes. Cela permettra à terme de connaître des orientations d'émergence en créant des conditions initialement favorables. On saura ainsi que placer telles conditions initiales dans tel environnement avec tels agents produit tel résultat émergent après un certain nombre d'interactions et d'itérations.

Partant du principe que la vie est une propriété de l'organisation de la matière et pas de la matière elle-même, le mathématicien John Horton Conway [Conway, 1982] est à l'origine du célèbre « Jeu de la vie », popularisé en grande partie par les nombreuses discussions dans Scientific American en 1970 et 1971. Il s'agit d'un exemple simple d'automate cellulaire. Les automates cellulaires sont une famille de simples modèles à états finis qui présentent d'intéressantes propriétés d'émergence de comportements au travers des nombreuses interactions dans les populations qu'ils réifient. La plupart des jeux de simulation actuels du type wargame sont basés sur ce principe.

Un automate cellulaire est défini par :

- Un ensemble I appelé alphabet d'entrée.
- Un ensemble d'états S dans lequel évolue l'automate.

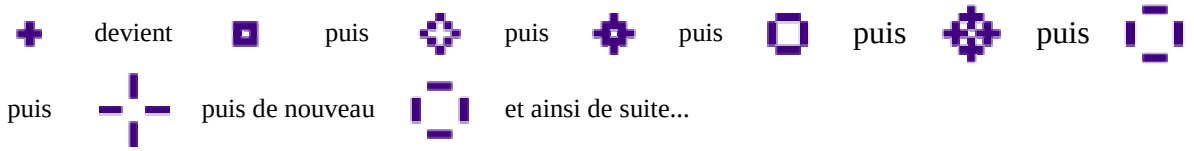
- Un état initial S_0.
- Une fonction N : S * I → S qui calcule le prochain état de l'automate.

Dans le cas des automates cellulaire, Wolfram [Wolfram, 1984] a proposé une classification en quatre classes de comportement :

1. homogène
2. périodique
3. chaotique
4. complexe

Dans un automate cellulaire, l'état à un temps t+1 est fonction de l'état actuel est de l'état des voisins directs au temps t. C'est grâce à la richesse de ces interactions que l'automate cellulaire peut présenter des propriétés émergentes de complexité très intéressantes. En tout état de cause, il s'agit d'un comportement qui se stabilise et souvent se répète autour d'un point de convergence souvent appelé attracteur. On ne peut pas parler d'apprentissage dans la mesure où les règles de fonctionnement ne sont pas mémorisées mais réévaluées en temps réel. On pourra observer l'émergence d'invariances ce qui est d'ailleurs un aspect important des recherches de Crutchfield-Mitchell.

Règles de fonctionnement du Jeu de la Vie : Sur un espace à 2 dimensions : Si une cellule, occupée ou non, possède exactement trois de ses cellules voisines directes occupées, elle sera occupée à la prochaine période. Si toute cellule occupée possède exactement deux de ses cellules voisines directes occupées, elle sera occupée à la prochaine période. Pour toutes les autres situations, la cellule ne sera pas occupée à la prochaine période.



On retrouve ce principe expliqué dans l'article "Le Jeu de la Vie toujours vivant" de Jean-Paul Delahaye publié dans Pour la science N°221 mars 1996 qui est une excellente introduction (et qui va même au-delà) au Jeu de la Vie. Il présente des exemples de configurations stables ou périodiques. Certains motifs peuvent se reproduire. Par exemple, un vaisseau (ou navire) est une structure finie (de cellules vivantes) qui, après un certain nombre de générations réapparaît, mais translatée dans une certaine direction (et avec une distance non nulle). Le plus petit nombre de générations au bout desquelles elle réapparaît translatée est appelé sa période. On définit la vitesse d'un vaisseau par la distance de translation (mesurée par la longueur du plus court chemin entre deux cellules : deux cellules adjacentes (cote à cote ou en diagonale) étant distantes de 1) divisée par la période.

L'exemple ci-dessous montre un cas typique de « clignotement », c'est à dire d'oscillation entre deux états dans un automate cellulaire.

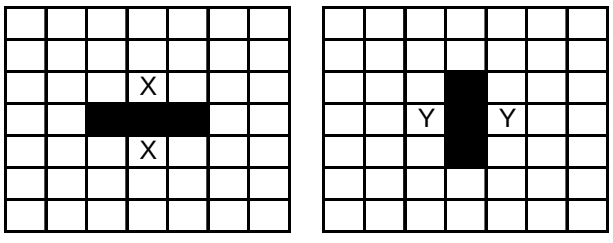


Figure 6 - Exemple d'oscillation dans un automate cellulaire

Les automates cellulaires ont quelques propriétés remarquables :

La reproduction

Le but de Von Neumann, à l'origine, lorsqu'il a conçu le principe des automates cellulaires, était de concevoir un mécanisme copiant la vie dans le sens où il pourrait se reproduire de lui même. Ainsi certains automates cellulaires sont capables de produire des copies d'eux-mêmes, propriété particulièrement intéressante lorsqu'il s'agit de modéliser la vie d'êtres vivants. Ce point de vue de la reproduction permet de classer les automates cellulaires en deux grandes catégories :

- les automates cellulaires actifs, c'est à dire auto-reproducteurs.
- les automates cellulaires passifs.

Les automates cellulaires actifs sont auto-reproducteurs dans le sens où ils contiennent une sous-configuration qui se comporte en « copieur universel » dirigeant activement la réplication via une fonction de transition assurant la réplication. Les automates cellulaires passifs sont ainsi nommés car leur reproduction est provoquée par la règle de transition et non pas par les caractéristiques de la configuration initiale. Le Jeu de la Vie est un exemple d'automate cellulaire passif.

L'inversibilité

On appelle automate inverse l'automate qui permet de revenir aux états précédents. On a démontré que tous les automates n'ont pas nécessairement un inverse. Pour démontrer qu'un automate n'a pas d'inverse il suffit de trouver deux configurations différentes qui aboutissent à la même configuration. On dit que l'automate de Conway est irréversible. Cela signifie qu'il n'est pas possible de proposer des règles permettant de déduire d'une configuration son état à la génération précédente. En effet, d'une part, il existe des structures ayant plusieurs antécédents possibles, d'autre part il en existe aussi n'ayant pas d'antécédent. Il est donc impossible de « repasser le film en arrière ». Par exemple :

 et  sont deux états antérieurs possibles de 

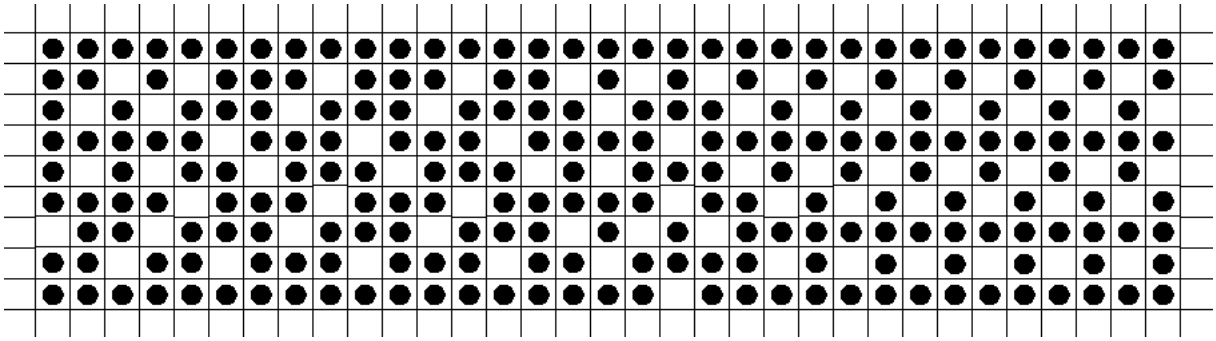
L'indécidabilité

L'une des caractéristiques importantes des automates cellulaires est le caractère d'indécidabilité qui touche nombre de leurs propriétés. Ainsi, déterminer si un automate cellulaire possède un inverse est indécidable : il ne sera jamais possible d'écrire un programme prenant en paramètres un automate quelconque et pouvant décider si oui ou non cet automate possède un inverse. De la même façon, l'« avenir » d'un automate est indécidable. On n'a pas de méthode générale permettant de déterminer si un automate ne va pas s'éteindre au bout d'un certain nombre de générations ou s'il va se stabiliser.

Les Jardins d'Eden

Les Jardins d'Eden sont des structures pour lesquelles il n'existe pas d'antécédent. Cela ne se produit pas avec les automates inversibles. De même, par exemple, une configuration Jardin d'Eden ne peut être un attracteur car elle ne peut apparaître que comme première configuration d'une suite de configurations. L'aspect remarquable de ce concept est que la question de l'existence de Jardins d'Eden est indécidable. Cela a été démontré par J. Kari en

1990. La figure suivante montre une configuration qui n’a pas de prédécesseur : c’est un jardin d’Eden.



Un autre résultat intéressant avait déjà été trouvé en 1962 par E. Moore et J. Myhill : un automate possède des Jardins d’Eden si et seulement si deux configurations donnent le même résultat. Cette précision pour souligner le fait que cette question a su tenir en haleine les informaticiens, et peut-être plus généralement, les logiciens pendant longtemps.

Les attracteurs

Les attracteurs des automates sont des configurations qui reviennent indéfiniment. Ils permettent des études intéressantes sur les automates et la démonstration de propriétés. Ainsi, J. Kari a démontré que toute propriété de l’ensemble limite (l’ensemble des attracteurs) est vraie pour certains automates et fausses pour d’autres est indécidable. Ce résultat est finalement le plus extraordinaire de tous, car il nous montre que nous ne saurons jamais rien à l’infini des réseaux d’automates cellulaires.

Nous verrons dans le chapitre consacré aux applications de notre travail, que nous avons rencontré le problème de l’inversibilité des automates et que certaines caractéristiques que nous souhaitions développer dans notre logiciel n’ont pas pu être implémentées.

Bibliographie

[Conway, 1982] J. H. Conway, E. R. Berlekamps et R. K. Guy. Winning, Ways for yourmathematical plays, Academic Press, 1982.
[Toffoli et Margolis, 1987] Toffoli T et Margolis N., Cellular Automata Machines : A New Environnement for Modeling, Cambridge MIT Press, 1987.
[Wolfram, 1984] Wolfram S., Universality and complexity in cellular automata, Physica D, 10:1-35, 1984.

Hasard et simulation

L'idée d'utiliser le hasard pour résoudre des problèmes de physique est née lors de la seconde guerre mondiale. Pendant le projet Manhattan, qui allait conduire à la réalisation de la première bombe atomique, John von Neumann et Stanisalw Ulam devaient étudier le problème de la diffusion des neutrons dans les matériaux. Pris par le temps et le peu d'accès possibles à un réacteur nucléaire, ils n'avaient pas la possibilité de réaliser des expériences sur ce sujet. Par contre ils disposaient du premier calculateur généraliste existant, l'ENIAC.

Aussi reprirent-ils une idée d'Enrico Fermi, qui datait des années 1930 : à chaque collision d'un neutron avec un atome ils laissèrent au hasard le soin de décider s'il était

absorbé ou s'il rebondissait ainsi que la détermination de la valeur de son énergie après le choc. Il était totalement impossible à l'époque, compte tenu de l'avancement des théories et de la faiblesse des moyens de calcul, de déterminer exactement ce qui se passait au cours de la collision. Décider aléatoirement de ce qui allait se passer était par contre simple et ne nécessitait que la mise au point d'un générateur aléatoire, à condition bien sûr, que ce tirage puisse simuler avec assez de précision la vraie loi physique ignorée. Nous laisserons à la physique quantique le soin de justifier cette technique qui se révéla satisfaisante. Le projet réussit comme nous le savons tous.

Cette technique fut baptisée Monte-Carlo par référence au célèbre casino. Le nom est resté. Elle fut reprise en 1953 par des chercheurs de Los Alamos pour étudier le passage de l'état amorphe à l'état cristallin d'un ensemble de billes. Encore une fois la raison en était qu'on ne connaissait pas les lois physiques régissant cette transformation et qu'on n'aurait pas su les simuler en calculant toutes les interactions entre les sphères. L'idée de la simulation par les méthodes de Monte-Carlo était née. Depuis, cette technique de simulation s'est généralisée. Grossièrement on peut dire que son usage se justifie dans les domaines suivants:

- la physique recourt à une hypothèse probabiliste et le générateur de nombres aléatoires s'efforce de reproduire les caractéristiques de cette loi.
- le nombre d'objets à considérer dépasse largement les possibilités de calcul des ordinateurs. On s'efforce de pallier cette insuffisance par les jeux de hasard: c'est ce qu'on fait par exemple lorsqu'on veut étudier le comportement d'un gaz.

Le problème, pour le numéricien, est de réaliser un générateur de hasard de bonne qualité. A priori un ordinateur ne se prête pas à ceci: c'est une machine entièrement déterministe dont les états sont parfaitement reproductibles et, en théorie, prévisibles. L'idée est donc de calculer des suites de nombres qui présentent le minimum de corrélation entre eux, dont la séquence périodique soit la plus longue possible. On choisit un nombre appelé germe auquel on fait subir un ensemble d'opérations simples (additions, multiplications, décalage des chiffres, troncatures...) au moyen d'un algorithme non linéaire. Le résultat est une suite pseudo-aléatoire. Le calcul est cependant déterministe car un même germe conduit toujours à une même suite. Trop souvent le physicien prête peu d'attention au germe (il utilise souvent toujours le même), ce qui dégrade la qualité du générateur qu'il a choisi, aussi sophistiqué soit-il. La technique de choix du germe peut introduire un biais. Ainsi il est assez classique de choisir la date et l'heure courantes, exprimée dans les unités de la machine. Deux suites successives présentent alors une relation d'ordre puisque le germe de la seconde est forcément plus grand que celui de la première.

Les générateurs de nombres aléatoires se distinguent par leur qualité à générer de longues suites. Comme ils sont, le plus souvent, utilisés des millions de fois au cours d'un même problème, il faut faire un compromis entre qualité et rapidité du calcul. Tous les générateurs présentent des limitations quant à la longueur de la séquence générée et les corrélations qui existent entre les nombres de la suite. Le biais numérique caractéristique de chaque algorithme est délicat à évaluer. Il peut être satisfaisant pour un problème mais se révéler catastrophique pour un autre. C'est là une des plus grandes difficultés des méthodes de Monte-Carlo qui justifie que les nombreux ouvrages sur ce sujet consacrent une part importante à la génération de ces nombres. Le physicien doit donc accorder la plus grande importance à son générateur, chaque fois qu'il envisage de recourir à des techniques de Monte-Carlo. Il devrait, du moins en théorie, commencer par étudier le générateur lui-même et tester la sensibilité de ses résultats au changement d'algorithme.

Quelques applications des SMA

Il est vain de vouloir recenser tous les domaines pouvant recourir aux SMA mais parmi les plus connus, on trouve le datamining [Fayyad & al., 1996]. Cela consiste à extraire des formes ou des règles de grandes collections de données. Cette technique utilise aussi parfois des SMA [Kargupta & al., 1997] et [Botia & al., 1998], bien que le rôle de l'analyste – encore humain – soit toujours prépondérant pour déterminer l'intérêt de la forme ou de la règle. Le Web étant aujourd'hui la plus grande banque d'informations connue, on évoque le terme de WebMining [Spiliopoulou & al., 1999] et [Cooley & al., 1997] comme spécialisation du datamining.

Le domaine de la médecine propose aussi beaucoup de projet recherche en SMA. La Fondation Health On the Net et le Laboratoire d'Imagerie Moléculaire et Bio-Informatique de l'Hôpital Universitaire de Genève ont développé MARVIN -Multi-Agent Retrieval Vagabond on Information Networks-, un robot qui recherche les sites web et les documents se rapportant à un domaine d'intérêt spécifique. Un tel robot a déjà été développé pour la santé et la médecine ainsi que pour d'autres domaines tels que la biologie moléculaire. Il s'agit uniquement d'un logiciel multi-agents recherchant de l'information [Marvin, 2002]. D'une manière générale, la recherche médicale a depuis longtemps été très portée sur l'intelligence artificielle. Les SMA ont naturellement apporté leur contribution au domaine. Les applications les plus connues sont la simulation de cellules [Boucher & al., 1988] ou l'étude du cerveau [Germond & al., 1999].

L'agent trouve naturellement sa place dans les applications de groupware, soulageant l'utilisateur de nombreuses tâches telles que le classement du courrier et autres documents, l'envoi d'e-mails, la recherche de documents présents dans différentes boîtes à lettres ou archivés sur un thème donné, etc. Un certain nombre d'agents de base sont intégrés dans des solutions comme Lotus Notes ou Teamware Office mais les utilisateurs peuvent définir leurs propres agents, à l'aide des modules Notes Agents et Teamware Agents, respectivement. l'Agent Knowledge Server est un système qui, au fur et à mesure que l'utilisateur travaille et se connecte sur tel ou tel site Web, construit automatiquement des liens de connaissance [Teamware, 2002]. Dans le même ordre d'idée, le projet InfoSleuth [Fowler & al., 1999] utilise la technologie agent pour tenter de reconstituer la sémantique des données sur un réseau et donc retrouver plus efficacement de l'information. Les principales applications sont les moteurs de recherche sur Internet.

La thèse de L.Rousseau, co-encadré par Suzanne Pinson (LAMSADE) propose une approche multi-agents dans les démarches participatives en aménagement dans le cadre d'une décision spatiale collaborative. Cette thèse en cours de réalisation implique le CEMAGREF. Toujours dans le domaine géographique, le projet STAFF qui consiste à construire une SMA pour prévoir les crues [Regis & al., 2002].

La thèse de [Boucher, 1999] propose, quant à elle, une nouvelle approche de la vision par ordinateur, insistant sur le rôle joué par les différentes informations et connaissances présentes dans un système intégré de vision. Une approche distribuée est proposée, sous la forme d'un système multi-agents, où chaque agent est spécialisé pour la reconnaissance d'un concept de l'image. Des capacités d'adaptation et de coopération sont nécessaires aux agents pour mener à bien leurs tâches, et ce tant au niveau local qu'au niveau des groupes d'agents (travaillant sur un même groupement perceptuel). Ce système est appliqué pour la reconnaissance des cellules en mouvement. Chaque cellule est divisée en plusieurs

composantes (noyau, pseudopode, cytoplasme, ...) qui servent de concepts de base pour la spécialisation des agents.

Nous nous intéressons, de notre coté, au partage des ressources réseau qui peut requérir aussi des systèmes multi-agents [Fenet & Hassas, 1998], ainsi qu'à la gestion de production en temps réel [Attoui, 1997]. Nous projetons de développer prochainement une application de notre middleware Phoenix au carrefour de ces deux domaines et présentant des caractéristiques de grid computing, d'ASP et d'Autonomic Computing. L'approche globale consiste à considérer le réseau local de l'entreprise comme un « super SMA ». Par extension il est possible de sortir du réseau privé pour faire de même sur un WAN ou sur le réseau Internet global. Des projets de recherche des grands constructeurs informatiques, comme IBM, s'orientent sur des concepts proches des SMA adaptatifs au niveau hardware et software. On parle d'Autonomic Computing [ACIBM, 2003] et d'ordinateur planétaire…

TOY-PROBLEM

Un « toy-problem » est un petit exemple didactique permettant de mettre en exergue une fonctionnalité d'un programme. Ils permettent aussi de comparer le nombre de lignes de code nécessaires pour l'exécution de chacun d'entre eux sur les différentes plates-formes SMA et comparer ainsi le niveau d'abstraction du langage. Le groupe de travail ASA [ASA, 2002] a défini quelques toy-problems parmi lesquels :

- les proies-prédateurs
- le ping-pong
- calcul d'une factorielle
- le jeu Pierre-Papier-Ciseaux
- calcul d'une suite de Fibonacci

Un certain nombre de critères ont été retenu pour classifier et évaluer les plate-formes. Ces critères sont les suivants :

- Caractéristiques générales, dont l'objectif est de fournir une fiche d'identité de la plate-forme.
- Modèles multi-agents: description des concepts multi-agents disponibles sur la plate-forme.
- Caractéristiques physiques: description des caractéristiques physiques des systèmes multi-agents implantables au travers de la plate-forme.
- Environnement de développement.
- Environnement d'exécution.

Nous avons développé ces Toy-Problems ainsi que d'autres exemples pour montrer le fonctionnement de Phoenix, notamment pour le système d'échange d'énergie global et le découpage de rôles en tâches. Nous n'allons présenter ici qu'un des toy-problems : le jeu du « Pierre-Papier-Ciseaux ». La simplicité de celui-ci ne requiert pas de système d'énergie.

C'est un jeu bien connu à deux joueurs qui jouent simultanément et ont trois coups possibles : Pierre, Papier ou Ciseaux. Le résultat de chaque coup est :

- Les deux joueurs jouent la même chose : pas de point
- Pierre contre Papier : Papier gagne (et marque 1 point)

- Pierre contre Ciseaux : Pierre gagne (et marque 1 point)
- Ciseaux contre Papier : Ciseaux gagne (et marque 1 point)

Une partie dure 10 coups. Le but de ce problème est de jouer à P-P-C avec un système multi-agents. Un agent arbitre est nécessaire pour coordonner la partie puisque les deux coups des joueurs doivent être simultanés. Un des joueurs doit pouvoir être humain. Cet exemple permet de tester la communication entre agents et le système de synchronisation.

Nous avons choisi de découper autant que possible les compétences pour montrer la modularité maximale. A noter qu'il est aussi possible de mettre plusieurs fonctions dans un même fichier skill pour gagner de la place. Le code présenté est en langage C#.NET, dernière évolution de la plate-forme Phoenix à ce jour.

// Voici le code de la classe PPCAgent dérivant d'un agent YAMAM. Cette classe instancie un agent et lui affecte un rôle.

```
using System;
using Message;
using Identifiers;
using System.Windows.Forms;
using Yamam;

namespace PPC
{
	/// <summary>
	/// Description résumée de la classe.
	/// </summary>
	[Serializable()]
	public class PPCAgent : Yamam.AbstractAgent
	{

		public PPCAgent(string name):base(name)
		{
		}
		public PPCAgent(string name,string rolename):base(name)
		{
			this.AddRole(rolename);
		}
	}
}
```

// Nous présentons à présent les classes compétences (Skill)
// Voici le code de la skill PPCAffecter. Elle dérive de la classe Skill

```
using System;
using Yamam;
using Resources;
using Utils;

namespace PPC
{
	/// <summary>
	/// Description résumée de SkillAdd.
	/// </summary>
	public class PPCAffecter : AbstractSkill
	{
		public PPCAffecter(String name) : base(name)
		{
```

```
                }

                public override Object run()
                {
                        Double a = Convert.ToDouble(resourcesIn[0]);
                        return (a);
                }
        }
}
```

// Voici le code de la skill PPCProposer. Elle dérive de la classe Skill

```
using System;
using Yamam;
using Resources;
using Utils;

namespace PPC
{
        /// <summary>
        /// Description résumée de SkillAdd.
        /// </summary>
        public class PPCProposer : AbstractSkill
        {
                private Random r ;

                public PPCProposer(String name) : base(name)
                {
                        r = new Random();
                }

                //on génère un nombre aléatoire : 1 = Pierre, 2 = Papier, 3 = Ciseaux
                public override Object run()
                {
                        Double a = Convert.ToDouble(r.Next(1,4));
                        return (a);
                }
        }
}
```

// Voici le code de la skill PPCTesterLes2A1. Elle dérive de la classe Skill

```
using System;
using Yamam;
using Resources;
using Utils;

namespace PPC
{
        /// <summary>
        /// Description résumée de SkillAdd.
        /// </summary>
        public class PPCTesterLes2A1 : AbstractSkill
        {
                public PPCTesterLes2A1(String name) : base(name)
                {
                }
```

```
                public override Object run()
                {
                        Double a = Convert.ToDouble(resourcesIn[0]);
                        Double b = Convert.ToDouble(resourcesIn[1]);
                        if(a==0 && b==0)
                                return(Convert.ToDouble(1));
                        return(Convert.ToDouble(0));
                }
        }
}
```

// Voici le code de la skill PPCDemanderHumain qui s'occupe de l'interface de dialogue avec l'utilisateur

```
using System;
using Yamam;
using Resources;
using Utils;
using System.Windows.Forms;

namespace PPC
{
        /// <summary>
        /// Description résumée de SkillAdd.
        /// </summary>
        public class PPCDemanderHumain : AbstractSkill
        {
                public PPCDemanderHumain(String name) : base(name)
                {

                }

                //on demande à l'utilisateur une proposition : 1 = Pierre, 2 = Papier, 3 = Ciseaux
                public override Object run()
                {
                        DialogResult result=MessageBox.Show("Cliquez sur OUI pour jouer PIERRE, NON
pour PAPIER, ANNULER pour CISEAUX","Votre choix",

        MessageBoxButtons.YesNoCancel,MessageBoxIcon.Question,MessageBoxDefaultButton.Button1,Mes
sageBoxOptions.ServiceNotification);
                        if(result==DialogResult.Yes)
                                return (Convert.ToDouble(1));
                        if(result==DialogResult.No)
                                return (Convert.ToDouble(2));
                        if(result==DialogResult.Cancel)
                                return (Convert.ToDouble(3));
                        return(null);
                }
        }
}
```

// Voici le code de la skill PPCTrouverGagnantManche

```
using System;
using Yamam;
using Resources;
using Utils;

namespace PPC
```

```
{
	public class PPCTrouverGagnantManche : AbstractSkill
	{
		public PPCTrouverGagnantManche(String name) : base(name)
		{
		}

		// 1 = Pierre, 2 = Papier, 3 = Ciseaux
		public override Object run()
		{
			Double joueur1 = Convert.ToDouble(resourcesIn[0]);
			Double joueur2 = Convert.ToDouble(resourcesIn[1]);
			//si joueur 1 gagne
			if((joueur1==2 && joueur2==1) || (joueur1==1 && joueur2==3) || (joueur1==3 &&
joueur2==2))
				return(Convert.ToDouble(1));
			//si joueur 2 gagne
			if((joueur2==2 && joueur1==1) || (joueur2==1 && joueur1==3) || (joueur2==3 &&
joueur1==2))
				return(Convert.ToDouble(2));
			//égalité
			return(Convert.ToDouble(0));
		}
	}
}
```

// Voici le code de la skill PPCTrouverGagnantMatch

```
using System;
using Yamam;
using Resources;
using Utils;
using System.Collections;

namespace PPC
{
	public class PPCTrouverGagnantMatch : AbstractSkill
	{
		public PPCTrouverGagnantMatch(String name) : base(name)
		{
		}

		// 1 = Pierre, 2 = Papier, 3 = Ciseaux
		public override Object run()
		{
			ArrayList manches=(ArrayList)(resourcesIn[0]);
			for(int i=0;i<manches.Count;i++)
				Log.Write(" indice "+i+" : "+manches[i],name,3);
			int victoires1=0, victoires2=0;
			for(int i=0;i<manches.Count;i++)
			{
				if((double)(manches[i])==1)
					victoires1++;
				if((double)(manches[i])==2)
					victoires2++;
			}

			if(victoires1>victoires2)
				return(Convert.ToDouble(1));
```

```
                        if(victoires2>victoires1)
                                return(Convert.ToDouble(2));
                        return(Convert.ToDouble(0));
                }
        }
}
```

// Voici le code du programme principal

```
using System;
using Identifiers;
using Message;
using WebServiceUtils;
using System.Net;
using Utils;
using System.Threading;
using System.Collections;

namespace PPCToy
{

        public class PPCTest
        {
                [STAThread]
                static void Main(string[] args)
                {
                        Console.WriteLine("\n====================================");
                        Console.WriteLine("| Toy-Problem Pierre/Papier/Ciseaux |");
                        Console.WriteLine("====================================\n");

                //=== Création du proxy sur le Web Service ===
                        Constant.Init();
                        Console.WriteLine(" + Création du proxy du Web Service, veuillez patienter...");
                        Plateforme proxy = new Plateforme("http://" +   Constant.Get("localplatform")
                        +"/Plateforme/Plateforme.asmx");
                        proxy.init();
                        Console.WriteLine(" | OK\n");
                //=== Fin de la création du proxy ===

                //=== Création des agents nécessaires pour l'exemple Pierre-Papier-Ciseaux ===
                        Console.WriteLine(" + Création des agents nécessaires...");
                        //les deux agents joueurs Joueur1 et Joueur2
                        String agentName1=(String)(proxy.CreateAgent("PPC","PPCAgent",null));
                        String agentName2=(String)(proxy.CreateAgent("PPC","PPCAgent",null));
                        Console.WriteLine(" | Agents joueurs 1 et 2 : OK");

                        //Et l'agent Arbitre
                        String agentName3=(String)(proxy.CreateAgent("PPC","PPCAgent",null));
                        Console.WriteLine(" | Agents arbitre : OK\n");
                //=== Fin de la création des agents ===

                //=== Création des rôles ===
                        //création du rôle ARBITRE
                        proxy.CreateRole("ppc.arbitre");

                        //création du rôle JOUEUR
                        proxy.CreateRole("ppc.joueur");
                //=== Fin de la création des rôles ===

                //=== Affectation des compétences aux agents ===
```

```
		//affectation à l'Arbitre des compétences nécessaires à son fonctionnement
		String[]
		RefereeSkills={"PPCDemanderSiJoueurHumain","PPCAffecter","PPCTrouverGagna
		ntManche","PPCTrouverGagnantMatch"};
		proxy.GiveSkillsToAgent(agentName3,RefereeSkills);

		//affectation aux Joueurs des compétences nécessaires à leur fonctionnement
		//affectation au Joueur1
		String[] PlayerSkills={"PPCAffecter","PPCProposer","PPCDemanderHumain"};
		proxy.GiveSkillsToAgent(agentName1,PlayerSkills);

		//affectation au Joueur2
		proxy.GiveSkillsToAgent(agentName2,PlayerSkills);
	//=== Fin de l'affectation des compétences aux agents ===


		//=== Lancement de l'arbitre ===
		Console.WriteLine("L'agent arbitre a commencé son exécution");
		proxy.GiveRoleToAgent(agentName3,"ppc.arbitre");

		//On attend que l'arbitre ait fini d'initialiser les variables...
		bool initDone=false;
		do
		{
			Object rsrcobj = proxy.GetResource("i","ppc");
			if(rsrcobj.ToString()!="null")
			{
				Object resource = XMLSerializer.DeSerializeSOAPString(
				rsrcobj.ToString());
				initDone=(resource!=null);
			}
			Thread.Sleep(50);
		}
		while(!initDone);
		//Il a fini

		//=== Lancement des joueurs ===
		Console.WriteLine("L'agent joueur 1 a commencé son exécution");
		proxy.GiveRoleToAgent(agentName1,"ppc.joueur");

		Console.WriteLine("L'agent joueur 2 a commencé son exécution");
		proxy.GiveRoleToAgent(agentName2,"ppc.joueur");


		//=== On attend la fin du match ===
		Double i=0;
		do
		{
			Object rsrcobj = proxy.GetResource("i","ppc");
			if(rsrcobj.ToString()!="null")
			{
				Object resource = XMLSerializer.DeSerializeSOAPString(
				rsrcobj.ToString());
				i= Convert.ToDouble(resource);
			}
			Thread.Sleep(50);
			Console.Write(".");
		}
		while(i<11);
		// Fin du jeu
```

```
                        //Récupération de toutes les ressources
                        Object rsrcobj2 = proxy.GetApplicationResources("ppc");
                        Hashtable rsrcList = (Hashtable)(XMLSerializer.DeSerializeSOAPString(
                        rsrcobj2.ToString()));

                        // Affichage du vainqueur
                        if(Convert.ToDouble(rsrcList["vainqueur"])==0)
                                Console.WriteLine("Le match s'est soldé par un match nul\n");
                        else
                                Console.WriteLine("\n\nLe match est terminé, le gagnant est : Joueur
                                "+rsrcList["vainqueur"]);

                        // Affichage des résultats des 10 tours
                        ArrayList results = (ArrayList)(rsrcList["manches"]);
                        int j=1;
                        foreach(double result in results)
                        {
                                if(result!=0)
                                        Console.WriteLine("Manche "+j+": Joueur "+result+" gagnant");
                                else
                                        Console.WriteLine("Manche "+j+": match nul");
                                j++;
                        }
                }
        }
}
```

// Passons à présent à la description des rôles. Un rôle est affecté à un agent et décrit les tâches qui le composent
// Les rôles sont décrit dans des fichiers XML. Voici le rôle PPC.arbitre.XML

```
<?xml version="1.0"?>
<role>
        <resources>
                <!-- pour dire aux joueurs de jouer : 0 = attends, 1 = joue -->
                <resource name="$joueurGO1" value="0"/>
                <resource name="$joueurGO2" value="0"/>
                <!-- drapeau pour savoir que les 2 joueurs ont joue -->
                <resource name="$joueursOntJoue" value="0"/>
                <!-- résultat des 2 joueurs pour une manche : 1 = pierre, 2 = papier, 3 = ciseaux -->
                <resource name="$result1" value="0"/>
                <resource name="$result2" value="0"/>
                <!-- tableau utilise pour stocker les résultats des 10 manches -->
                <resource name="$manches" value="|0|0|0|0|0|0|0|0|0|0"/>
                <!-- vainqueur du match (1 ou 2) -->
                <resource name="$vainqueur" value="0"/>
                <!-- compteur de boucles -->
                <resource name="$i" value="0"/>
        </resources>
        <structure>
                <for variable="$i" start="1" end="10" step="1">
                        <!-- on dit aux joueurs qu'ils peuvent jouer -->
                        <task name="PPCDebutManche" in="" out="$joueurGO1, $joueurGO2,
$joueursOntJoue"/>
                        <!-- on attend que les 2 joueurs aient joue -->
                        <while term1="$joueursOntJoue" term2="1" sign="!=">
                                <task name="PPCTestJoueursOntJoue" in="$joueurGO1, $joueurGO2" out=
"$joueursOntJoue"/>
                        </while>
```

```
                        <!-- on calcule le vainqueur de la manche -->
                        <task name="PPCDeterminerVainqueurManche" in= "$result1, $result2" out= "$manches[$i]"/>
                </for>
                <!-- on calcule le vainqueur global -->
                <task name="PPCDeterminerVainqueurMatch" in="$manches" out="$vainqueur,$i"/>
        </structure>
</role>
```

// Voici le rôle (toujours en XML) de PPC.joueur.XML

```
<?xml version="1.0"?>
<role>
        <resources>
                <resource name="$joueurGO#" value="0"/>
                <!-- resultat des 2 joueurs pour une manche : 1 = pierre, 2 = papier, 3 = ciseaux -->
                <resource name="$result#" value="0"/>
                <!-- compteur de boucles -->
                <resource name="$bidon" value="4012"/>
        </resources>
        <structure>
                <while term1="$i" sign="&lt;=" term2="10">
                        <if term1="$joueurGO#" term2="1" sign="==">
                                <task name="PPCJouer" in="" out="$result#,$joueurGO#"/>
                        </if>
                </while>
                <task name="PPCBidon" in="" out="$bidon"/>
        </structure>
</role>
```

// Voici le rôle XML de PPC.joueurhumain.XML

```
<?xml version="1.0"?>
<role>
        <resources>
                <resource name="$joueurGO2" value="0"/>
                <!-- resultat des 2 joueurs pour une manche : 1 = pierre, 2 = papier, 3 = ciseaux -->
                <resource name="$result2" value="0"/>
                <!-- compteur de boucles -->
                <resource name="$bidon" value="4012"/>
        </resources>
        <structure>
                <while term1="$i" sign="&lt;=" term2="10">
                        <if term1="$joueurGO2" term2="1" sign="==">
                                <task name="PPCJouerHumain" in="" out="$result2,$joueurGO2"/>
                        </if>
                </while>
                <task name="PPCBidon" in="" out="$bidon"/>
        </structure>
</role>
```

// Description des tâches. Les tâches sont appelées par les rôles et sont aussi au format XML
// Voici le code de la tâche PPCdebutmanche.xml

```
<task>
        <!-- On signale aux joueurs le début d'une manche : on leur dit qu'ils peuvent jouer -->
        <skill name="PPCAffecter" in="1" out="$peutjoueur1"/>
        <skill name="PPCAffecter" in="1" out="$peutjoueur2"/>
```

```
        <skill name="PPCAffecter" in="0" out="$ontjoue"/>
        <return vars="$peutjoueur1,$peutjoueur2,$ontjoue"/>
</task>
```

// Voici le code de la tâche PPCDeterminerVainqueurManche.xml

```
<task>
        <skill name="PPCTrouverGagnantManche" in="$0,$1" out="$gagnant"/>
        <return vars="$gagnant"/>
</task>
```

// Voici le code de la tâche PPCDeterminerVainqueurMatch.xml

```
 <task>
        <skill name="PPCTrouverGagnantMatch" in="$0" out="$gagnant"/>
        <skill name="PPCAffecter" in="11" out="$finpartie"/>
        <return vars="$gagnant,$finpartie"/>
</task>
```

// Voici le code de la tâche PPCJouer.xml

```
<task>
        <skill name="PPCProposer" in="" out="$proposition"/>
        <skill name="PPCAffecter" in="0" out="$ajoue"/>
        <return vars="$proposition,$ajoue"/>
</task>
```

// Voici le code de la tâche PPCJouerHumain.xml

```
<task>
        <skill name="PPCDemanderHumain" in="" out="$proposition"/>
        <skill name="PPCAffecter" in="0" out="$ajoue"/>
        <return vars="$proposition,$ajoue"/>
</task>
```

// Voici le code de la tâche PPCTestJoueursOntJoue.xml

```
<task>
        <skill name="PPCTesterLes2A1" in="$0,$1" out="$les2ontjoue"/>
        <return vars="$les2ontjoue"/>
</task>
```

// Voici le code de la tâche taskAdd.xml

```
<task>
        <skill name="SkillAdd" in="$0,$1" out="task_out"/>
        <return vars="task_out"/>
</task>
```

// Voici le code de la tâche taskDiv.xml

```
<task>
        <skill name="Skilldivq" in="$0,$1" out="quotient"/>
```

```
        <skill name="SkillAdd" in="$0,quotient" out="somme"/>
        <return vars="quotient,somme"/>
</task>
```

// Voici le code de la tâche taskSub.xml

```
<task>
        <skill name="SkillSub" in="$0,$1" out="diff"/>
        <return vars="diff"/>
</task>
```