UNIVERSITÉ DE
SHERBROOKE
Faculté de génie
Génie électrique et génie informatique

GEMAS: UN ENVIRONNEMENT DE DÉVELOPPEMENT D'APPLICATIONS BASÉES SUR LES SYSTÈMES MULTI-AGENTS HÉTÉROGÈNES

Thèse de doctorat
Spécialité : génie électrique

Charles-Antoine BRUNET

Sherbrooke (Québec) Canada Décembre 2000

RÉSUMÉ

Les agents et les systèmes multi-agents hétérogènes (SMA) représentent l'approche logicielle du futur pour une grande partie de la communauté informatique. Ils sont bien adaptés aux applications d'aujourd'hui qui sont de plus en plus complexes, distribuées et hétérogènes. Avec ces caractéristiques, ils représentent une architecture intéressante pour des systèmes intelligents. Les SMA sont un domaine de recherche très actif.

Bien que les agents et les SMA représentent une approche très prometteuse, leurs caractéristiques intrinsèques comme la complexité, l'hétérogénéité et la distribution, en font des systèmes difficiles à analyser, à concevoir et à réaliser. Plusieurs défis de taille restent à résoudre, comme l'intégration systématique d'agents hétérogènes, le développement d'outils génériques réutilisables, la définition de méthodologies de conception et le raisonnement avec des connaissances incomplètes, incertaines et contradictoires.

D'un autre côté, les véhicules autonomes sont un domaine très populaire en intelligence artificielle, car ils représentent un défi tant au niveau du matériel qu'au niveau du système logiciel qui les contrôle. L'architecture logicielle des SMA est une approche intéressante pour ce type d'application.

Les travaux de recherche présentés dans cette thèse proposent, conçoivent et réalisent un modèle générique d'agent nommé GAM (*Generic Agent Model*) et un environnement générique de développement de SMA hétérogènes nommé GEMAS (*Generic Environment for Multi-Agent Systems*). Ils solutionnent les problèmes d'intégration d'agents hétérogènes et de manque d'outils génériques et réutilisables de développement de SMA.

Afin de valider le modèle GAM et l'environnement GEMAS, un nouveau modèle de pilote pour véhicule autonome basé sur les SMA hétérogènes a été conçu et réalisé à l'aide des agents issus du modèle GAM et de l'environnement GEMAS.

REMERCIEMENTS

Je remercie mes deux codirecteurs, M. Mario Tétreault et M. Ruben Gonzalez-Rubio, qui m'ont permis de réaliser ces travaux de recherche. Leurs précieux conseils, leur compréhension et leur grande patience m'ont permis de mener à bien les travaux présentés dans cette thèse.

D'autres personnes m'ont donné de précieux conseils lors de la phase de réalisation et je tiens à les remercier : les professeurs Jean-Marie Dirand et François Michaud ainsi que deux collègues, Saed Ehsani et Éric Laurin.

Je voudrais aussi remercier le personnel du département pour leur soutien dans les différentes phases de mes travaux. Plus particulièrement, j'aimerais remercier Marcel Lapointe, Bernard Beaulieu, François Côté, Micheline Tessier et Yvon Turcotte. Sans leur aide, le travail aurait été plus difficile.

Je tiens aussi à remercier ma famille ainsi que Mélanie pour leur soutien sans relâche. Je dois énormément à Mélanie qui m'a offert un support moral inconditionnel et une aide inestimable avec d'innombrables suggestions et corrections au texte de cette thèse.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

CHAPITRE 1

INTRODUCTION

La majorité des humains considèrent que les systèmes informatiques sont de simples exécutants, car ils ne font rien de plus que ce qui est exigé d'eux. Heureusement, la plupart du temps, ces systèmes exécutent correctement les ordres reçus. Selon Müller [109], dans certaines situations comme des systèmes de commande industriels ou des robots, il est souhaitable que les systèmes soient dotés d'une plus grande intelligence et qu'ils soient plus autonomes afin qu'ils prennent des décisions raisonnables par eux-mêmes. Ces caractéristiques sont désirables, car elles sont synonymes d'outils plus performants.

La conception et la mise en oeuvre de systèmes intelligents est le sujet de beaucoup de travaux de recherche. Le domaine de l'intelligence artificielle y est spécialement dédié. Les discussions sur les systèmes intelligents abordent généralement le thème de la complexité de manière directe ou indirecte, car inculquer de l'intelligence à un système n'est pas une tâche simple. Plusieurs approches de conception, dont les systèmes multi-agents (SMA), ont été proposées afin de faire face à la complexité des systèmes intelligents.

1.1 Agents et systèmes multi-agents

Aujourd'hui, un système informatique intelligent ou un logiciel intelligent est souvent associé au terme **agent** ou au terme **agent logiciel**. Le concept d'agent est une approche de conception de logiciels permettant de faire face à la complexité et à l'hétérogénéité grandissante des systèmes informatiques en favorisant la modularité et la réutilisation. Selon Nwana et Ndumu [118], deux raisons rendent le concept d'agent très populaire : c'est un concept technique et une métaphore naturelle.

Pour plusieurs, dont Tokoro [145], les systèmes basés sur les agents représentent l'approche de conception logicielle du futur, car elle est bien adaptée aux systèmes ouverts (*open systems*). Pour d'autres, comme Russel et Norvig [128], les agents représentent le but ultime de l'intelligence artificielle, car elle peut être définie comme étant le problème de concevoir un agent intelligent. Certains, comme Wooldridge [154], ne voient dans les agents qu'une approche de génie logiciel.

Généralement, les agents ne sont pas utilisés seuls ; ils sont utilisés en groupe et forment un SMA. Les SMA utilisent l'agent comme abstraction de base. Selon Jennings et Wooldridge [76], les systèmes composés de plusieurs agents sont les plus prometteurs, mais plusieurs défis devront être relevés afin qu'ils le deviennent :

- Les SMA doivent permettre l'intégration d'éléments hétérogènes pour préserver les investissements passés, pour faciliter l'introduction graduelle de nouvelles technologies et pour optimiser l'utilisation de la plate-forme matérielle et logicielle tout en utilisant les méthodes les plus appropriées pour chacune de ces technologies.
- Des outils de développement réutilisables doivent être développés. Les chercheurs doivent développer une méthodologie pour la conception d'agents et de systèmes basés sur les agents. Ils doivent aussi développer des bancs d'essais pour l'évaluation des différentes options de conception.
- Les agents doivent, dans certains cas, pouvoir raisonner avec des informations incertaines, incomplètes et contradictoires. Ils doivent opérer en temps réel et démontrer de l'adaptabilité.
- Les SMA doivent être prévisibles et contrôlables par l'usager afin qu'ils se comportent correctement dans des situations critiques et pour donner la puissance du système à l'usager.
- Les composants d'un SMA doivent être localement autonomes afin de mieux gérer la sécurité, les changements incrémentaux et l'obéissance aux besoins légaux.

Les SMA représentent une méthode très intéressante de résolution de problèmes pour intégrer des collaborateurs ou des experts issus de différents domaines et dont l'expertise est représentée sous différentes formes. Malgré l'abondance des travaux réalisés, d'après Martin et al. [99], les SMA n'atteindront pas leur plein potentiel et ne deviendront pas omniprésents tant que des outils et des environnements de développement adéquats ne seront pas disponibles.

Les SMA sont utilisés dans plusieurs domaines d'applications dont l'aviation, la distribution d'électricité et les systèmes de télécommunication ; Chaib-draa [28] en fait une revue

intéressante. En particulier, les véhicules autonomes bénéficient des avantages de l'approche multi-agents. Selon Hexmoor et al. [66], les véhicules autonomes sont attrayants auprès des industriels à cause de leur potentiel utilitaire et commercial. Ils attirent aussi l'attention des chercheurs en intelligence artificielle, car ils évoluent dans un monde réel, dynamique et imprévisible et ils ont une composante matérielle qui a nécessairement des limitations physiques. De plus, le système informatique qui contrôle le véhicule doit démontrer de l'intelligence et de l'autonomie en s'adaptant aux situations imprévues et en gérant les incertitudes qui sont nécessairement présentes.

1.2 Véhicules autonomes

La réalisation d'un système de contrôle pour un véhicule autonome représente un défi à plusieurs niveaux. La vision, la prise de décision, la commande du véhicule, l'analyse du champ visuel, la coordination avec l'environnement et un grand nombre d'autres tâches représentent autant de défis. Pour ces raisons, les véhicules autonomes et la robotique représentent un domaine d'application intéressant pour les SMA.

La modélisation complète d'un pilote pour contrôler un véhicule représente un défi autant du point de vue matériel et logiciel que du point de vue conceptuel. Les chercheurs ont proposé une multitude d'architectures afin de résoudre les problèmes liés au développement d'un système complet. L'architecture du système de contrôle du véhicule est une des considérations les plus importantes.

Les architectures sont souvent conçues à partir de niveaux ou d'une hiérarchie de niveaux ; des exemples se trouvent dans [4, 17, 69, 71, 88, 100]. Ce type d'architecture semble être une approche qui fait l'unanimité. Ceci est compréhensible, car le modèle d'un système de pilotage de véhicule est souvent perçu par les humains comme un ensemble hiérarchisé de tâches distinctes.

Afin de réaliser un véhicule autonome, un système complet doit intégrer de manière cohérente toutes les solutions aux sous-problèmes. Selon Rochwerger et al. [126] ainsi que Smith et Starkey [138], les domaines impliqués dans les solutions étant très différents, il semble

évident qu'un modèle complet de pilote doit être réalisé à l'aide d'un environnement de développement permettant l'utilisation d'approches hétérogènes. Une approche comme les SMA est bien adaptée à ce type de problématique.

Plusieurs problèmes liés aux véhicules autonomes restent sans solution acceptable. Un des plus importants est l'architecture du système informatique contrôlant le véhicule. L'architecture du modèle complet de pilote doit être facilement extensible et portable pour permettre une intégration future des solutions aux problèmes restant à résoudre et afin qu'elle puisse être adaptée aux nouvelles architectures matérielles. Sans ces caractéristiques d'extensibilité et de portabilité, les concepteurs doivent recréer le système à chaque fois, ce qui peut être très coûteux et indésirable.

1.3 Problématique

Les SMA représentent le futur des systèmes informatiques pour plusieurs chercheurs [108, 109, 156]. Si cette prédiction s'avère exacte, les SMA pourront donc avoir un impact majeur sur l'informatique de demain. Si les SMA deviennent une réalité quotidienne dans les produits informatiques et autres produits commerciaux ayant une composante informatique, alors on doit solutionner certains problèmes importants demeurés sans réponse acceptable jusqu'à ce jour.

Selon Jennings et Wooldridge [76], un des défis les plus importants soulignés par la littérature est le manque d'outils génériques et réutilisables. De bons outils de développement de SMA doivent être disponibles pour que les SMA deviennent une composante importante du futur de l'informatique. Par exemple, si le langage C++ est utilisé pour développer autant de logiciels et qu'il est aussi présent, c'est parce qu'il y a des environnements de développement disponibles. Sans ces environnements, il ne pourrait pas y avoir autant de projets logiciels basés sur le langage C++.

La réalisation d'outils génériques et réutilisables de développement de SMA ferait avancer la recherche d'un grand pas. L'intérêt de la communauté informatique envers le développement de tels outils est grand et ils méritent une investigation sérieuse. Si de tels outils existaient,

trois défis importants identifiés par les chercheurs [76, 97, 99] dans le domaine des SMA seraient relevés :

1. définir et réaliser un environnement générique de développement de SMA hétérogènes ;
2. définir un modèle générique d'agent ;
3. faire l'intégration systématique d'agents hétérogènes dans un système.

1.4 Thèse

Face à cette problématique de réalisation d'outils de développement, deux buts principaux sont fixés afin de la résoudre :

1. **Définir et réaliser un environnement générique de développement de SMA hétérogènes basé sur un modèle générique d'agent.** Un tel type d'environnement résout les trois problèmes importants liés aux SMA qui viennent d'être mentionnés. De plus, la définition d'un modèle générique d'agent pourrait ouvrir la voie à l'unification des efforts de recherche sur les SMA ; il fournirait un modèle conceptuel qui pourrait rapprocher la théorie et la pratique.
2. **Définir et réaliser une architecture extensible et réutilisable de pilote basée sur les SMA.** Il s'agit de valider les concepts sur lesquels repose l'environnement de développement proposé et de valider son implémentation en réalisant une application réelle.

Afin d'atteindre les deux buts fixés, la recherche se décompose en quatre étapes :

1. **Proposer un modèle générique d'agent.** Ce modèle doit permettre d'engendrer les types d'agents les plus courants. Il s'agit de trouver un principe unificateur et d'établir une base architecturale d'agent afin d'unifier tous les types d'agents sous un même modèle conceptuel.
2. **Démontrer la validité du modèle générique d'agent.** Il s'agit de démontrer que le modèle générique peut conceptuellement et pratiquement engendrer des agents pour tous les paradigmes principaux.
3. **Proposer un environnement générique de développement de SMA hétérogènes avec des mécanismes intégrés de débogage.** Il s'agit de proposer un environnement qui assure le développement d'agents hétérogènes en le basant sur le modèle générique d'agent proposé à l'étape 1. Il s'agit aussi de définir des mécanismes intégrés de débogage pour l'environnement de développement et pour le modèle générique d'agent afin d'assurer leur utilité pratique.

4. **Proposer et réaliser un modèle de pilote de véhicule autonome.** Il s'agit de valider le modèle générique d'agent, l'environnement de développement et les outils de débogage en réalisant une application. Il s'agit aussi de résoudre la problématique présentée dans le deuxième but ci-haut.

Les travaux de recherche présentés dans cette thèse proposent et réalisent un environnement générique et portable de développement de SMA hétérogènes nommé GEMAS (*Generic Environment for Multi-Agent Systems*). Cet environnement est basé sur un modèle générique d'agent nommé GAM (*Generic Agent Model*). Les travaux de recherche réalisés sont originaux et contribuent aux domaines des agents et des SMA au niveau conceptuel et pratique de plusieurs manières :

- Ils définissent et réalisent un modèle générique d'agent, le modèle GAM, qui permet de générer des agents hétérogènes (sections 5.2 et 6.2).
- Ils réalisent des agents basés sur le modèle GAM avec les paradigmes les plus courants : logique, logique floue, réseaux de neurones artificiels, algorithmes génétiques et langages procéduraux (sections 5.4 et 6.3 à 6.8).
- Ils proposent et réalisent un environnement générique et portable de développement de SMA hétérogènes avec des mécanismes intégrés de débogage : l'environnement GEMAS (section 5.6 et le chapitre 6).
- Ils conçoivent et réalisent des outils de débogage de SMA pour l'environnement GEMAS (sections 5.7 et 6.9).
- Ils conçoivent et réalisent un modèle extensible et réutilisable de pilote pour véhicules autonomes basé sur les SMA hétérogènes. La réalisation est effectuée en utilisant l'environnement GEMAS (chapitres 7 et 8).

1.5 Organisation du document

Cette thèse est organisée en quatre parties. Voici une description de chacune d'elles :

Revue de la littérature : chapitres 2, 3 et 4.

Les chapitres 2 et 3 définissent les notions d'agent et de SMA à partir des recherches déjà effectuées par la communauté informatique. Ils énoncent les principes reliés à ces notions et les principaux thèmes des recherches effectuées. Entre autres, ils identifient les différentes architectures d'agents et de SMA de même que les principaux problèmes reliés à ces domaines.

Le chapitre 4 fait un survol de la littérature concernant la modélisation de systèmes de pilotage pour véhicules autonomes. Il identifie les différents types de véhicules autonomes et les différents types d'architectures utilisés pour modéliser un pilote. Entre autres, ce chapitre identifie les caractéristiques architecturales désirables pour la modélisation de pilotes et les principaux problèmes restant à résoudre.

Environnement GEMAS : chapitres 5 et 6.

Le chapitre 5 fait l'analyse conceptuelle d'un agent et propose le modèle GAM, l'environnement GEMAS et leurs outils intégrés de débogage.

Le chapitre 6 expose la réalisation des concepts annoncés au chapitre 5. Il montre l'implémentation du modèle GAM et des agents hétérogènes qui en sont dérivés ainsi que l'implémentation de l'environnement GEMAS et des outils de débogage. Il donne aussi les résultats des tests effectués à l'aide d'applications de tests.

Application aux véhicules autonomes : chapitres 7 et 8.

Le chapitre 7 analyse de manière conceptuelle un modèle complet, extensible et réutilisable de pilote basé sur les SMA hétérogènes.

Le chapitre 8 explique comment le modèle de pilote proposé au chapitre 7 a été implémenté à l'aide du modèle GAM et de l'environnement GEMAS. Il donne aussi les résultats des simulations effectuées.

Bilan : chapitre 9.

Le chapitre 9 présente une discussion et tire des conclusions sur les travaux de recherche. Il identifie les contributions apportées au domaine des SMA et au domaine de la modélisation de pilote. Il se termine par une ouverture sur des travaux futurs qui pourraient être entrepris à partir des travaux de recherche présentés.

PREMIÈRE PARTIE

REVUE DE LA LITTÉRATURE

CHAPITRE 2

AGENTS

La plupart des gens considèrent les systèmes informatiques comme de simples exécutants. Généralement, les systèmes exécutent correctement les ordres reçus, mais ils ne font rien de plus. Selon Müller [109], les usagers souhaitent souvent plus que de l'obéissance. Ils désirent des systèmes intelligents et autonomes qui prennent des décisions raisonnables par eux-mêmes, car ces caractéristiques sont synonymes d'outils plus performants. Un système informatique intelligent ou un logiciel intelligent est souvent associé au terme **agent** ou au terme **agent logiciel**.

2.1 Introduction

Selon Kay [80], l'idée d'un agent logiciel a été proposée vers le milieu des années cinquante par McCarthy et quelques années plus tard par Selfridge. La mise en oeuvre d'agents et de langages agent a vraiment débuté pendant les années soixante-dix. Par exemple, Negroponte [114] propose que des agents aident à la gestion de l'interface aux utilisateurs. Un des premiers développements qui a eu un impact important est issu des travaux de Hewitt [65] et du modèle d'acteur qu'il propose. Ces travaux ont été suivis par une grande quantité de recherches et de développements au cours des années qui suivirent.

Aujourd'hui, le terme agent est utilisé dans plusieurs domaines et dans des contextes très variés. Par exemple, il est attribué aux acteurs, aux systèmes experts, aux robots physiques ou virtuels, aux processus informatiques en général, aux entités intelligentes qui réagissent à leur environnement, aux avions, aux organisations comme des entreprises industrielles, et aux systèmes qui simulent des sociétés humaines [109, 136, 156]. Il est devenu une notion centrale en intelligence artificielle et dans d'autres disciplines informatiques.

Le terme agent est très populaire pour deux raisons, selon Nwana et Ndumu [118] : c'est un concept technique et une métaphore naturelle. L'agent est une approche de conception de logiciels permettant de faire face à la complexité et à l'hétérogénéité grandissante des systèmes informatiques en favorisant la modularité et la réutilisation. La réutilisation est cependant encore loin d'être atteinte, d'après Hayes-Roth et al. [62].

La popularité des agents a généré de nombreux travaux de recherche depuis les vingt dernières années. Wooldridge et Jennings [156] classifient les efforts des chercheurs selon trois axes majeurs :

Théories : Les théories sur les agents sont essentiellement des spécifications formelles. L'approche théorique s'intéresse surtout à des sujets comme la manière de conceptualiser formellement un agent, quelles propriétés il doit avoir, comment représenter formellement ces propriétés et comment raisonner sur celles-ci.

Architectures : Les architectures d'agents représentent la transition de la spécification formelle à l'implémentation. Lorsqu'ils créent une architecture, les chercheurs s'intéressent généralement à la conception de systèmes informatiques qui répondent aux spécifications formelles des théoriciens et à la conception d'architectures logicielles ou matérielles appropriées pour ces théories.

Langages : Les langages agent matérialisent les principes énoncés par l'approche des théoriciens dans des langages de programmation. Les concepteurs d'un langage agent doivent trouver une façon de programmer un agent avec un ensemble de primitives qu'ils doivent définir. Ensuite, ils doivent trouver une manière de compiler et d'exécuter les programmes basés sur ce langage agent.

Ce chapitre fait état de la recherche sur les agents logiciels. Il donne une définition de ce qu'est un agent à la section 2.2 et de leurs caractéristiques les plus usuelles à la section 2.3. Une classification des agents selon leur architecture est exposée à la section 2.4 et une discussion et quelques conclusions sont données sur l'état de la recherche à la section 2.5.

2.2 Définitions

S'engager à donner une définition du terme agent est une aventure périlleuse. D'ailleurs, lors d'une conférence[1], Hewitt faisait remarquer que donner une définition du terme agent est une tâche difficile pour les chercheurs, car c'est l'équivalent de demander une définition

[1] *Thirteenth International Workshop on Distributed Artificial Intelligence.*

de l'intelligence aux chercheurs en intelligence artificielle. Il y a des ententes, mais pas de consensus ; il y a des définitions, mais pas d'absolu.

Les difficultés inhérentes à la définition précise d'un agent et la pluralité des points de vue sont les deux sources majeures de divergence des définitions. Le point de vue du génie logiciel et le point de vue de l'intelligence artificielle se distinguent lorsque vient le temps de définir un agent. Ces deux points de vue mènent respectivement à une définition pratique et à une définition théorique d'agent. Malgré les différences, beaucoup de caractéristiques sont communes et il y a entente au moins sur une définition minimale d'agent.

Le modèle **boîte noire** d'un agent qui est illustré à la figure 2.1 est celui de Müller [109]. Ce modèle représente la définition minimale d'un agent. L'agent est décrit par une fonction f qui a comme entrées les perceptions de l'environnement et les communications externes reçues. L'agent génère comme sorties des actions sur l'environnement et des communications externes. L'agent est qualifié d'**autonome**, car la fonction f n'est pas contrôlée par une autorité externe.

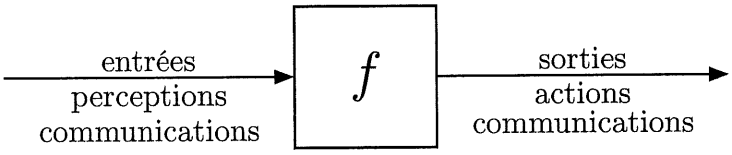


Figure 2.1 – Modèle boîte noire d'un agent [109].

Cette définition générale et simple d'agent convient à toutes les approches et modèles proposés par les chercheurs. Elle énonce les conditions nécessaires pour une définition d'agent, mais non les caractéristiques suffisantes. Par exemple, selon cette définition, un algorithme de tri est un agent. On peut argumenter que l'algorithme de tri reçoit en entrée les données à trier et donne en sortie les données triées, que la fonction f est l'algorithme de tri et qu'elle n'est pas contrôlée par une entité externe. Bien qu'il soit **légitime** de considérer un algorithme de tri comme un agent, la question est plutôt : est-ce qu'il est **utile** de le faire ? Cette question de légitimité et d'utilité génère des discussions depuis la parution d'un article de McCarthy [103] sur l'importance des états mentaux dans le cadre des systèmes intelligents.

Afin d'en arriver à une définition unifiée d'agent, des échanges ont eu lieu lors de la conférence internationale ATAL'96 [110]. Les définitions du terme agent de plusieurs chercheurs dans le domaine furent rassemblées et un article a présenté le résultat de cette réflexion, soit celui de Franklin et Graesser [53]. Cet article a été suivi de plusieurs autres de la part de certains des chercheurs participants qui apportent des précisions et des variantes sur le résultat. Il n'y a toujours pas de consensus, mais il est clair qu'il y a deux points de vue : l'approche pratique et l'approche théorique.

2.2.1 Définition pratique

Agent : Un système informatique matériel ou logiciel qui a les propriétés suivantes :

- Il est **autonome**. Il agit sans intervention directe, humaine ou autre, et possède un certain contrôle sur ses actions et ses états internes.
- Il est **communicatif**. Il communique avec d'autres agents, possiblement humains, en utilisant une communication basée sur un langage agent.
- Il est **réactif**. Il perçoit son environnement et réagit aux changements perçus. Si la situation l'exige, il réagit adéquatement dans le temps et sans planification.
- Il est **proactif**. Il ne fait pas simplement que réagir à son environnement, il est capable d'initiative en démontrant un comportement dirigé par ses propres buts.

Cette définition d'agent est celle de Wooldridge et Jennings [156]. Elle est très populaire et une des plus adoptées. Elle est perçue comme une extension naturelle de la programmation orientée objet. D'après Genesereth et Ketchpel [56], la définition pratique est celle qui est généralement adoptée par le génie logiciel basé sur les agents.

Les agents de ce type partagent beaucoup de similitudes avec la programmation orientée objet, comme les messages, l'encapsulation, les états, mais ils se distinguent au moins de deux manières. La première, d'après Wooldridge [154], est que les agents sont exécutés en parallèle alors que les objets sont exécutés dans le même *thread*. La deuxième, selon Tokoro [145], est que l'agent est composé d'une hiérarchie d'objets ; l'agent est comme une entité vivante et les objets en constituent les organes.

D'autres chercheurs proposent des définitions très semblables. Généralement, elles sont un peu plus sévères ou un peu plus indulgentes sur certains critères. Un exemple de ce type de définition est celle de Müller [108] :

Agent : Un système qui a les propriétés suivantes :
- Il vit avec d'autres agents dans un monde artificiel appelé W.
- Il a les capacités de percevoir W et de manipuler W.
- Il a une représentation (au moins partielle) de W.
- Il est dirigé par ses propres buts ; il a donc les capacités de planifier ses activités.
- Il peut communiquer avec d'autres agents.

Cette définition ressemble beaucoup à celle de Wooldridge et Jennings, même si elle est formulée différemment. En fait, elle diffère surtout sur la notion d'autonomie. Müller juge que la définition d'autonomie de Wooldridge et Jennings est trop sévère, car elle interdit l'intervention directe humaine ou autre, mais elle permet l'interaction avec d'autres agents qui eux peuvent être humains. Müller reformule la définition d'autonomie de manière plus permissive comme étant la capacité d'un agent d'agir même en l'absence d'intervention humaine directe.

2.2.2 Définition théorique

Agent : Un système intelligent pour lequel il faut adopter le **point de vue intentionnel** (*intentional stance*).

Cette définition est celle de Singh [136]. Généralement, d'après Wooldridge et Jennings [156], elle est la définition qui est utilisée pour définir un agent intelligent. Le point de vue intentionnel est le fait d'attribuer à un système des notions mentales, comme des attitudes, des croyances, des désirs et autres termes semblables. Le philosophe Dennett [37] a nommé ce type de notions mentales des **notions intentionnelles**. Elles sont utiles pour expliquer ou prédire les comportements humains. Dennet a aussi nommé **systèmes intentionnels** les entités dont le comportement peut être prédit en leur attribuant des notions intentionnelles.

Cette définition théorique d'agent est plus stricte et restrictive que la définition pratique. Elle inclut la définition pratique, mais exige que la conception ou l'implémentation de l'agent soit faite en utilisant des notions mentales, comme les notions intentionnelles. En d'autres termes, d'après Singh [136], un système est considéré intelligent si on a besoin, pour des raisons intuitives ou scientifiques, d'attribuer au système des concepts cognitifs intentionnels

pour le caractériser, le comprendre, l'analyser ou prédire ses comportements. Les notions intentionnelles permettent aussi de faire un lien direct avec les actes du langage naturel qui sont utilisés, par exemple, dans les langages KQML (*Knowledge Query Manipulation Language* [48]) et FIPA-ACL (*FIPA-Agent Communication Language* [49]).

L'approche intentionnelle est très utile, car elle fournit des abstractions pour traiter et discuter des systèmes intelligents. Toutefois, d'autres chercheurs pourront peut-être démontrer que ces systèmes ne possèdent pas vraiment de croyances, de désirs, d'émotions ou de toutes autres notions intentionnelles. L'important, faisait remarquer Dennett [37], n'est pas de savoir si ces systèmes possèdent vraiment ces caractéristiques, mais plutôt s'il est possible de les percevoir et de les concevoir comme systèmes intentionnels de manière cohérente.

Selon Singh [136], les notions intentionnelles sont des abstractions qui peuvent être appliquées profitablement pour analyser des systèmes qui ont des implémentations inconnues ou qui peuvent changer. Il est pratique de les utiliser pour plusieurs raisons :

- Elles sont naturelles pour les humains.
- Elles donnent une description courte qui aide à comprendre et à expliquer le comportement de systèmes complexes.
- Elles fournissent des patrons d'actions qui sont indépendants des représentations physiques.
- Elles peuvent être utilisées par les agents pour raisonner sur d'autres agents.

2.2.3 Remarques

Donner une définition du terme agent est une tâche difficile et en donner une qui fait l'unanimité l'est encore plus. Franklin et Graesser [53] remarquent que les seuls concepts qui mènent à des définitions précises, claires et sans ambiguïtés sont les concepts mathématiques. Les agents vivent dans le vrai monde ou un monde virtuel et les concepts réels mènent à des définitions ou à des catégories floues lorsque leurs limites sont questionnées. Il est normal que l'essentiel de la définition d'agent plaise à une majorité de chercheurs et qu'il y ait des divergences d'opinions lorsque les limites sont approchées. Il existe une terminologie pour les agents, mais elle est facilement interprétée selon les désirs de chacun. Shoham [135, p. 51] résume bien l'état de cette terminologie (traduction libre) :

> Certaines notions sont surtout intuitives et d'autres plutôt formelles, certaines très restrictives et d'autres très générales. Le plus souvent, lorsque le terme agent est utilisé, il est question d'une entité fonctionnant continuellement et de façon autonome dans un environnement dans lequel peuvent exister d'autres agents ou d'autres processus. C'est peut-être la seule propriété acceptée par tous ceux utilisant le terme agent. Même le terme autonome n'est pas précis dans ce contexte, mais, généralement, son sens est que les activités de l'agent ne requièrent pas d'intervention ou d'aide humaine continuelle. Finalement, un agent est généralement considéré de haut niveau ou intelligent lorsqu'il possède des représentations symboliques ou que se manifestent des fonctionnalités qui semblent cognitives. Mais, même l'existence de représentation de haut niveau chez les agents n'est pas uniforme en intelligence artificielle.

Selon Luck et al. [91], le manque de consensus sur les concepts, les termes et les définitions cause une fragmentation de la recherche et des efforts de réalisation. Elle empêche le progrès d'approches théoriques générales et elle disperse les efforts de construction et de réalisation d'agents.

Le manque de définition absolue d'agent n'est pas nécessairement un problème, selon Russel et al. [128], car le concept d'agent est quand même un outil utile de conception et de réalisation de systèmes logiciels. La notion d'agent est utilisée comme outil d'analyse de systèmes et non pas comme une caractérisation absolue qui divise le monde en deux catégories, les agents et les autres.

Malgré l'absence d'une définition absolue d'agent, l'ensemble des définitions permet de dégager des conditions qui indiquent la pertinence d'utiliser l'approche agent. Müller [108] souligne bien ces conditions :

- Le système montre une distributivité naturelle, c'est-à-dire des entités autonomes, une distribution géographique et des données distribuées.
- Il existe un besoin d'interactions flexibles ; il n'y a pas d'assignation des tâches *a priori* et pas de processus fixes.
- Les agents sont impliqués dans un environnement dynamique physique ou virtuel.

Un environnement de développement générique doit permettre la réalisation de tout type d'agent, que sa définition soit pratique ou théorique. L'environnement ne doit pas limiter les concepteurs dans leurs réalisations.

2.3 Caractéristiques

La définition pratique ou théorique d'un agent impose à un système un minimum de caractéristiques afin qu'il puisse être considéré comme un agent ; il doit être autonome, communicatif, réactif et proactif. D'autres caractéristiques fréquemment mentionnées dans la littérature peuvent se révéler intéressantes ou souhaitables. Elles ressemblent beaucoup à celles attribuées généralement aux humains. Le tableau 2.1 énumère les caractéristiques intrinsèques et extrinsèques principales d'un agent telles qu'identifiées par Huhns et Singh [72].

Caractéristique	**Valeurs possibles**
longévité	transitoire à longue durée
niveau de cognition	réactif à délibératif
construction	déclarative à procédurale
mobilité	stationnaire à itinérant
adaptabilité	fixe à apprenant à autodidacte
modélisation	de l'environnement, de lui-même ou des autres agents
localisation	local à distant
autonomie sociale	indépendant à contrôlé
comportement social	autiste, conscient, responsable, joueur d'équipe
associativité	coopératif à compétitif à antagoniste
interactions	- logistique : directe ou via *facilitator*, médiateurs ou non agent - style/qualité/nature : avec des agents et le monde - niveau sémantique : déclaratif ou procédural

Tableau 2.1 – Caractéristiques intrinsèques et extrinsèques des agents [72].

Il est également possible d'imaginer des agents qui ont des défauts, comme être paresseux, menteur, antagoniste et hypocrite avec les conséquences qui en découlent pour le système.

Chacune des caractéristiques listées dans le tableau 2.1 peuvent être adoptées à divers degrés par un agent. Ces caractéristiques ne sont pas mutuellement exclusives ; un agent peut en posséder plusieurs.

La mobilité est une caractéristique qui est le sujet de beaucoup de recherche. Certains chercheurs, comme Johansen et al. [78], considèrent qu'un agent est nécessairement mobile, alors que d'autres chercheurs, comme Huhns et Singh [72], considèrent la mobilité uniquement comme une des caractéristiques que peut posséder un agent.

Indépendamment du point de vue adopté au sujet de la mobilité ou des autres caractéristiques, un environnement de développement générique doit permettre d'adopter ou de ne pas adopter chacune de ces caractéristiques afin de permettre aux concepteurs de spécialiser un agent selon les besoins de la tâche. L'environnement de développement doit laisser les concepteurs libres de faire les choix qu'ils désirent.

2.4 Architecture

L'architecture d'un agent est une facette importante de sa conception et de son implémentation, car elle fournit un cadre de développement. Il y a de bonnes raisons pour se concentrer sur l'architecture avant toute chose, d'ailleurs Hayes-Roth [62] ainsi que Müller [108] les soulignent bien :

- L'architecture fournit des **directives** générales pour une méthodologie de conception et d'implémentation.
- L'architecture donne une **structure**. Les modules et les différents niveaux de l'architecture permettent de structurer précisément les connaissances opérationnelles de l'agent.
- L'architecture permet la **réutilisation**. Le modèle d'exécution, qui est implicite à l'architecture, évite de tout programmer à partir de zéro.
- L'architecture permet la **standardisation**, car des mécanismes prédéfinis et indépendants de l'application sont disponibles.
- L'architecture permet de faire de la **prédiction**, car le comportement du système peut être prédit, dans une certaine mesure, en étudiant les comportements de base des agents.
- L'architecture permet l'**héritage**, car des développements ou des raffinements peuvent être ajoutés à l'architecture existante et ainsi l'étendre.

Les agents peuvent être classifiés selon leurs architectures. Dans ce cas, trois grandes classes d'agents se distinguent : les agents délibératifs, les agents réactifs et les agents hybrides.

2.4.1 Agents délibératifs

Une architecture délibérative ou un agent délibératif (*deliberative agent*), selon Wooldridge et Jennings [156], possède une représentation symbolique explicite du monde. L'agent prend des décisions en utilisant un raisonnement logique basé sur des manipulations symboliques ou sur des mécanismes de filtrage (*pattern-matching*). En simplifiant, il suffit de fournir une

représentation logique du monde et de laisser la machine logique faire ses déductions pour ainsi obtenir, en principe, un agent fonctionnel.

Depuis plusieurs années, les agents délibératifs sont le sujet de plusieurs recherches. Une des approches explore la modélisation en basant l'architecture sur les concepts de **croyance**, de **désir** et d'**intention** (*belief*, *desire* and *intention* ou BDI). La modélisation BDI prend donc le point de vue intentionnel. Müller [109] définit de manière informelle ces trois notions intentionnelles comme suit :

Croyances : Elles représentent les attentes de l'agent à propos de l'état courant du monde et sur les probabilités d'obtenir les résultats espérés en posant certaines actions.

Désirs : Ils spécifient les préférences de l'agent sur les états futurs du monde ou sur une suite d'actions. Une caractéristique importante des désirs est qu'un agent peut avoir un ensemble de désirs qui n'est pas cohérent et il n'est pas obligé de croire que ses désirs sont atteignables ou réalisables.

Intentions : Les intentions courantes de l'agent sont décrites par un ensemble de buts. L'agent doit souvent sélectionner un sous-ensemble de ses buts, possiblement un seul, et s'engager uniquement avec ceux retenus.

Deux autres notions sont souvent ajoutées afin de clarifier certaines étapes du processus interne d'un agent délibératif :

Buts : Ils dénotent les différentes options qu'un agent possède dans le processus de sélection d'un ensemble cohérent de désirs qui peuvent être poursuivis, sans s'y être encore engagé. Il est souvent requis que l'agent croie que ses buts sont réalisables.

Plans : Les plans sont très importants lorsque vient le temps d'implémenter les intentions, même s'ils ne sont pas un ingrédient conceptuel de l'architecture BDI. Les intentions sont souvent perçues comme des plans partiels d'actions pour lesquels l'agent s'est engagé pour atteindre ses buts. Il est ainsi possible de structurer les intentions en plans et de définir les intentions de l'agent comme les plans que l'agent a adoptés.

Toujours d'après Müller [108], une façon simple de voir les choses est de considérer les croyances comme une source de connaissances (*knowledge base*), les désirs guidant le choix des buts et les intentions comme des plans pour atteindre les buts fixés.

Les architectures BDI ne sont pas les seules architectures délibératives. Par exemple, la figure 2.2 illustre l'architecture ARCHON de Wittig [153]. L'architecture ARCHON permet d'intégrer plusieurs systèmes intelligents, possiblement déjà existants, dans un système.

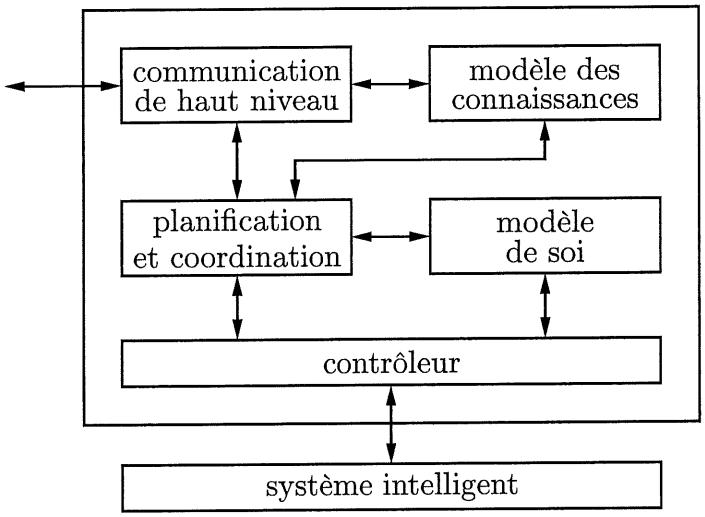


Figure 2.2 – Exemple d'architecture délibérative : ARCHON [153].

Les rôles de chacun des modules de la figure 2.2 sont les suivants. Le module contrôleur gère les interactions avec le système intelligent à intégrer. Le module communication de haut niveau gère les communications avec les autres agents du système. Le module modèle de soi contient les informations spécifiques à l'agent, comme ses comportements, ses tâches et ses plans. Le module modèle des connaissances contient les modèles des agents externes avec lesquels l'agent interagit. Finalement, le module planification et coordination contient la partie cognitive de l'architecture ; c'est ce module qui raisonne à propos du rôle de l'agent dans le système. Ce module évalue l'état courant des choses et décide des actions à poser afin d'exploiter les interactions avec les autres agents tout en s'assurant que sa contribution est bénéfique à la communauté d'agents.

Un agent basé sur le raisonnement logique est une idée simple et séduisante, mais dans la pratique il y a deux problèmes majeurs. Le premier est le problème de traduction : com-

ment traduire dans un temps raisonnable le monde réel dans une représentation logique adéquate et précise ? Ce problème a généré des travaux dans des domaines comme la vision, la reconnaissance de la parole et l'apprentissage. Le deuxième problème est celui de la représentation et du raisonnement : comment représenter symboliquement l'information sur des entités complexes et réelles et comment en arriver à faire raisonner l'agent sur ces informations à temps pour que les résultats soient utiles ? Ce problème a généré des travaux dans des domaines comme la représentation des connaissances, le raisonnement automatique (*automated reasoning*) et la planification automatique (*automated planning*).

En réalité, il est très difficile de faire des raisonnements logiques sur le monde réel. Selon Wooldridge et Jennings [156], même des problèmes qui semblent triviaux, comme des tâches impliquant le bon sens (*common sense*), sont extrêmement difficiles à résoudre. Le projet CYC [61] en est un bon exemple.

L'idée de bâtir des agents basés uniquement sur le raisonnement logique est très attrayante en théorie, mais, pour l'instant, elle n'est pas réalisable en pratique. Les agents délibératifs ont tendance à être lents et à utiliser intensivement les ressources. Les agents ou une société d'agents doivent se protéger des abus et du manque de ressources. Dans ce sens, Aubé et Senteni [8] proposent les émotions comme structure de régulation. Une autre critique des agents délibératifs, venant de Luck et al. [91], est qu'ils ne donnent pas de pistes pour la conception d'agents, car ils sont trop théoriques.

Malgré tout, les agents délibératifs représentent une classe d'agents très importante, nécessaire et utile. Ils représentent une bonne partie des efforts de recherche sur les agents, et récemment, des efforts soutenus ont été faits pour rapprocher les agents délibératifs du monde réel. Les réalisations d'agents délibératifs sont nombreuses. En voici quelques exemples : ARCHON [153], BDI [124, 125], dMars [57], GRATE [75], HOMER [149], JACK [26], JAM [70], PLACA [144] et PRS [58].

2.4.2 Agents réactifs

Au milieu des années quatre-vingts, les chercheurs en intelligence artificielle ont remis en question le paradigme de la représentation symbolique, comme celle adoptée pour les agents

délibératifs. Cette remise en question a mené aux architectures réactives (*reactive architecture*).

Les agents réactifs ne possèdent pas de représentation symbolique interne de leur environnement. Nwana et Ndumu [118] expliquent que les agents réactifs réagissent à l'état présent de leur environnement selon un mode stimulus-réponse sans aucune planification ou spécification *a priori*. Ce type d'agent prend ses décisions à l'exécution et généralement avec très peu d'information. Habituellement, les informations sont obtenues directement à partir des capteurs du système.

Une des prémisses de base de l'architecture réactive, selon Müller [109], est que la complexité du comportement de l'agent est le reflet de la complexité de son environnement et non pas de sa complexité interne. Dans ce sens, les architectures réactives sont en opposition avec les architectures délibératives. Cette classe de systèmes met l'emphase non pas sur un comportement optimal ou correct, mais plutôt sur un comportement robuste, flexible, avec plusieurs buts et une raison d'être.

Brooks [18, 19, 20] a énoncé trois principes clés lors de ses recherches sur les architectures réactives. Ces principes montrent bien comment les architectures réactives sont en opposition avec les architectures délibératives :

1. Un comportement intelligent peut être obtenu sans aucune représentation symbolique explicite tel que proposé par l'intelligence artificielle.
2. Un comportement intelligent peut être obtenu sans aucun raisonnement abstrait explicite tel que proposé par l'intelligence artificielle.
3. L'intelligence est un comportement émergeant de certains systèmes complexes.

Une des architectures réactives les plus connue est celle de Brooks, la *subsumption architecture* [17]. Tel qu'illustré à la figure 2.3, une architecture réactive est composée d'une hiérarchie de comportements spécialisés pour une tâche particulière. Chaque comportement est en compétition avec les autres pour le contrôle du système. Les étages les plus bas dans la hiérarchie représentent des comportements plus primitifs, comme éviter un obstacle dans le cas d'un robot. Les étages supérieurs ont priorité sur les étages inférieurs.

Chaque comportement représente un module très simple. La quantité totale de traitement réalisé par de tels systèmes est petite et ne nécessite aucune représentation symbolique. Cette architecture est traitée plus en détail à la section 4.5.

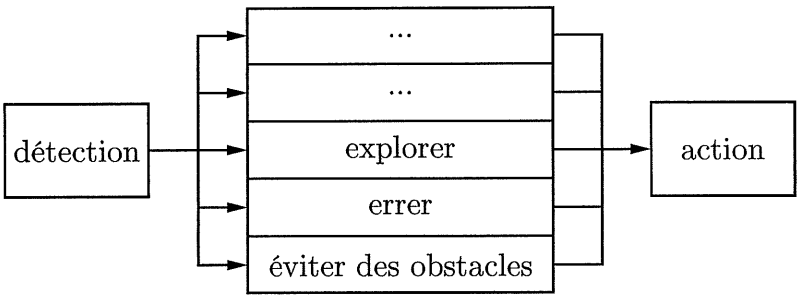


Figure 2.3 – Exemple d'architecture réactive : *subsumption architecture* [17].

Avec cette architecture, Brooks a conçu plusieurs robots qui accomplissent des tâches qui auraient été considérées impressionnantes si elles avaient été réalisées avec des systèmes basés sur une architecture délibérative. Il a identifié deux idées principales qui ont guidé ses recherches :

1. La localisation et l'incarnation : la vraie intelligence est située dans le monde, elle n'est pas incarnée dans des systèmes comme des systèmes experts.
2. L'intelligence et l'émergence : un comportement intelligent émerge de l'interaction de l'agent avec son environnement.

Les défenseurs des architectures réactives soulèvent un point important. Certaines classes de systèmes, comme les robots, ne peuvent pas s'arrêter, réfléchir et planifier lorsque la situation exige l'action. Il faut un comportement réactif face à l'environnement, car tout système en temps réel doit réagir promptement à des situations imprévisibles. L'approche est séduisante ; les systèmes n'ont plus besoin de réfléchir, car réagir suffit. Aucun besoin de raisonnement symbolique ou de tout autre raisonnement de haut niveau, car tous les problèmes sont résolus en réagissant localement aux situations qui surviennent.

Werner [151] a fait une réflexion intéressante sur les architectures réactives. Les agents réactifs sont réputés avoir une conception simple et démontrer un comportement complexe en réponse à une situation complexe. Plus la situation est complexe, plus le comportement est complexe.

Il est supposé qu'une action complexe est réduite à un environnement complexe. En fait, il est supposé qu'un comportement complexe émerge des interactions d'agents simples dans le contexte d'un environnement complexe.

Les agents réactifs supposent, toujours selon Werner, qu'aucun raisonnement symbolique ou raisonnement de haut niveau n'est nécessaire. Un ensemble d'agents réactifs agissant de concert verront une solution émerger magiquement de leurs interactions, aucune programmation n'étant nécessaire. Il argumente qu'on ne peut pas obtenir quelque chose pour rien. La complexité de la réaction de l'agent est limitée non seulement par la complexité de son environnement, mais aussi par la complexité de sa stratégie. L'argument des avocats de l'architecture réactive est que si l'on observe une fourmi dans un environnement complexe, son comportement semble complexe, même s'il est généré par des stratégies simples. Le problème avec ce raisonnement est que la fourmi a peu d'alternatives et que c'est surtout l'environnement qui l'influence, la fourmi n'influence que très peu l'environnement. La complexité interne de la fourmi reflète uniquement la complexité de son environnement. Elle ne génère pas de complexité significative d'elle-même. Pour un agent ou un ensemble d'agents réactifs simples, les possibilités sont limitées par leur manque de complexité. Ils ne construisent pas d'objets complexes et ne génèrent pas de complexité significative à moins que suffisamment d'information soit programmée en eux.

Albus [4] argumente dans la même direction. Les systèmes basés uniquement sur des comportements simples survivent rarement dans le vrai monde, sauf s'ils sont appliqués à un très grand nombre d'individus, comme les fourmis. Les comportements simples réussissent parce que la mort d'un individu n'a aucun impact sur la communauté dans lequel il se trouve. Un agent basé sur des stratégies simples gagne rarement un duel contre un agent basé sur une intelligence sophistiquée.

Malgré certaines objections, les architectures réactives ont bouleversé les conceptions classiques en intelligence artificielle. Elles sont une classe d'architectures qu'on ne peut pas ignorer et qui comporte beaucoup d'avantages. Les réalisations d'agents réactifs sont nombreuses. En voici quelques exemples : *Subsumption architecture* [17], *Agent Network Architecture* [94, 95, 96], Pengi [1] et ECO [46].

2.4.3 Agents hybrides

Plusieurs chercheurs pensent que les approches purement délibératives ou purement réactives ne sont pas bien adaptées pour la réalisation d'applications complexes réelles. Des architectures hybrides ont alors été proposées pour combiner les avantages des deux types d'architectures. Selon Guessoum et Dojat [60], les architectures hybrides décomposent un agent en modules ou en sous-agents de nature délibérative ou réactive. Elles intègrent les deux architectures dans un même système.

D'après Müller [109], l'agent hybride tente de résoudre le problème majeur de l'approche délibérative et celui de l'approche réactive en intégrant les deux. Le problème majeur des architectures délibératives vient du fait qu'elles sont généralement basées sur des mécanismes de raisonnement généraux et lents ; elles ont de la difficulté à réagir à temps. Le problème majeur des architectures réactives réside dans la difficulté d'avoir un comportement global qui est dirigé par des buts.

Les agents hybrides sont basés sur des architectures à niveaux ou à couches, comme la TouringMachines de Ferguson [47] qui est illustrée à la figure 2.4. Le fonctionnement des TouringMachines est le suivant.

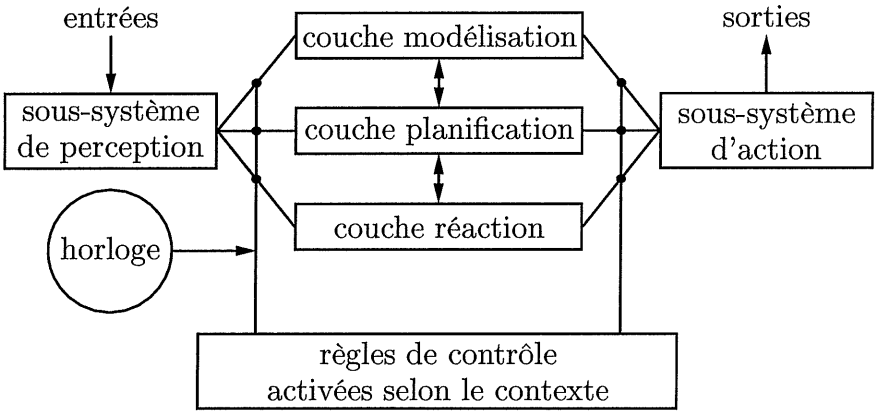


Figure 2.4 – Exemple d'architecture hybride : TouringMachines [47].

La couche réactive contient des fonctionnalités qui permettent à l'agent de réagir promptement face à des situations imprévues par les couches supérieures, comme pour éviter un obstacle. La couche planification contient les fonctionnalités nécessaires à l'agent pour créer,

modifier et exécuter des plans, comme pour décider d'un itinéraire pour se rendre à une destination. La couche modélisation permet à l'agent de modéliser les comportements d'entités externes ainsi que les siens afin d'expliquer les comportements observés et de faire des prédictions sur des comportements futurs. Un peu à la manière de la *subsumption architecture* (voir la section 2.4.2), chaque couche est capable d'inspecter et de modifier les données des autres couches afin d'altérer leurs flots de données.

Chaque couche est indépendante des autres et chacune d'elle tente de contrôler le système en générant des actions à partir de ses perceptions. Un système de médiation qui encapsule les couches est nécessaire, car les actions qui sont proposées par chacune des couches peuvent être conflictuelles. Les entrées et les sorties des couches sont générées de manière synchrone et les règles de contrôle sont appliquées sur les entrées et les sorties à chaque point de synchronisation. Les règles agissent comme des filtres entre les couches et les entrées et les sorties.

Une conception par couches est une approche puissante pour structurer les fonctionnalités et le contrôle. Elle est une approche de conception très utile pour intégrer dans un système des propriétés comme la réactivité, la délibération, la coopération et l'adaptabilité.

L'idée principale d'une architecture hybride est de structurer les fonctionnalités d'un agent en une hiérarchie de couches. Plus la couche est haute dans la hiérarchie, plus les informations traitées ont un niveau d'abstraction élevé. Ainsi, les couches les plus basses dans la hiérarchie s'occupent de tâches de nature plutôt réactive et à court terme et les couches les plus hautes s'occupent des tâches de nature plutôt délibérative et de planification à long terme. Le comportement global du système émerge de l'interaction entre les couches.

Selon Müller [109], une architecture à couches offre plusieurs avantages :

- Elle divise l'agent en modules. Les différentes fonctionnalités sont clairement séparées et liées par des interfaces bien définies.
- La conception de l'agent est plus compacte, robuste et facilite l'implémentation.
- Les différentes couches peuvent être exécutées en parallèle, ce qui augmente la capacité de traitement de l'agent.
- La réactivité de l'agent est augmentée, car, tout en planifiant, l'agent peut quand même réagir à des situations pressantes.

- La division en couches des connaissances et des types de connaissances diminue la quantité de connaissances nécessaires dans chaque couche. Il est ainsi possible d'augmenter la capacité réactive du système.

Un problème majeur de ces architectures, d'après Wooldridge et Jennings [156], se situe au niveau du contrôle des interactions entre des modules de natures très différentes, des modules délibératifs et des modules réactifs. Ces deux types de modules n'évoluent pas dans la même échelle de temps, ce qui rend difficile la tâche d'intégrer des séquences d'actions précises.

Un autre problème est de concevoir une interface propre qui permet la communication ou la collaboration entre deux niveaux d'abstraction très différents. Les couches délibératives possèdent une représentation symbolique du monde externe alors que les couches réactives n'en possèdent pas. Lors de la réalisation d'architectures hybrides, une des grandes difficultés est de définir une structure de contrôle pour la gestion des interactions entre les couches.

Bien que les architectures hybrides aient des difficultés inhérentes de contrôle interne, elles représentent, selon plusieurs, l'architecture la plus prometteuse. Les réalisations d'agents hybrides sont nombreuses. En voici quelques exemples : TouringMachines [47], InteRRaP [51, 52], 3T [13], MIX [73] et DSSA [62].

Du point de vue d'un environnement de développement générique, l'architecture des agents est un thème important. L'architecture est le principe organisateur de l'agent. Un environnement de développement générique doit permettre de développer des agents avec les architectures les plus courantes, comme les architectures délibératives, réactives et hybrides. L'environnement ne doit pas limiter les concepteurs dans leurs choix de conception.

2.5 Discussion

Le concept d'agent existe et évolue depuis un bon nombre d'années et ce n'est que récemment que les performances des ordinateurs et des réseaux permettent de réaliser des systèmes avec des agents de haut niveau. Les agents représentent encore une branche de la recherche en intelligence artificielle qui est en pleine croissance, mais, depuis quelques années, il y a une

tendance à la stabilisation des définitions et des concepts et une tendance à l'unification des approches.

Deux définitions de ce qu'est un agent se font compétition, la définition pratique et la définition théorique. Elles génèrent beaucoup de discussions et d'échanges entre les praticiens et les théoriciens. Elles ont des points en commun, mais il n'y a pas d'unification. Sur les caractéristiques de base il y a un consensus, mais pas dans les détails. En fait, la définition pratique attaque le problème de manière concrète par le bas et la définition théorique de manière abstraite et théorique par le haut. Éventuellement, les deux approches devront se rejoindre au milieu au grand bénéfice de l'avancement de la recherche sur les agents et de l'implémentation d'agents.

Dans les faits, d'après Petrie [122], la plupart des agents effectivement implémentés sont du type pratique. Les agents de type pratique ajoutent une plus-value au système et permettent d'intégrer des services hétérogènes. Pour l'instant, les agents de type pratique sont non seulement puissants, mais ils bénéficient d'une définition claire.

En fait, il ne s'agit pas de choisir entre la définition théorique ou pratique, car les deux sont utiles. La théorie représente la fondation sur laquelle il faut bâtir, mais il ne faut pas attendre une théorie complète sur les agents avant d'en réaliser. La théorie et la pratique contribuent l'une à l'autre et à l'avancement du domaine.

Les systèmes basés sur les agents représentent, pour plusieurs dont Tokoro [145], l'approche de conception logicielle du futur, car elle est bien adaptée aux systèmes ouverts (*open systems*). Par contre, comme le souligne Müller [108], peu de travail a été fait sur les méthodologies de conception agent.

Pour certains chercheurs, comme Russel et Norvig [128], les agents représentent le but ultime de l'intelligence artificielle, car elle peut être définie comme le problème de concevoir un agent intelligent. D'autres chercheurs, comme Wooldridge [154], voient dans les agents uniquement une approche de génie logiciel. Dans ce sens, il explique que si les chercheurs ne reconnaissent pas bientôt que les systèmes basés sur les agents ne relèvent pas de l'intelligence artificielle mais plutôt du génie logiciel ou tout simplement de l'informatique, alors, d'ici dix ans, ils

se demanderont pourquoi la technologie des agents a souffert du même sort que beaucoup d'autres idées en intelligence artificielle qui semblaient pourtant bonnes. Il insiste sur le fait que les agents sont simplement des composants logiciels qui sont conçus et implémentés de la même façon que beaucoup d'autres composants logiciels. Par contre, les techniques issues de l'intelligence artificielle sont souvent les meilleures techniques de conception d'agents.

Du point de vue de l'intelligence artificielle, selon Castelfranchi [27], les théoriciens rappellent que les vraies questions sont les suivantes : est-ce que les agents ont vraiment des buts et des croyances ? Est-ce qu'ils prennent vraiment des décisions ? Sont-ils vraiment autonomes ? Ce sont des questions à propos de mécanismes internes et elles sont très importantes pour une théorie délibérative et une architecture délibérative des agents.

En fait, selon Albus [4], il ne s'agit pas de choisir entre une architecture réactive ou délibérative, mais plutôt de trouver comment les intégrer dans une même et unique architecture. Une telle intégration permet d'obtenir des systèmes qui réagissent à temps, qui planifient à moyen et à long terme et qui ont des comportements continus. Le point clé de la conception d'un agent intelligent est d'intégrer les deux architectures pour former une seule architecture efficace de contrôle. Peu importe le choix de la définition d'un agent, son architecture est un point central.

Petrie [122] fait aussi une remarque intéressante. En général, dans les définitions d'agents, il n'y a pas d'objectifs de tests opérationnels. Il n'y a pas de méthode claire permettant de distinguer un agent d'un autre logiciel. De plus, les définitions généralement suggérées ont peu d'utilité pratique. Elles ne suggèrent pas de pistes techniques ou d'aide à la conception d'architectures. Aussi, les vrais agents sauvegardent les états d'une partie du problème et contribuent en raisonnant sur les changements de ces états. Ceci est un vrai critère réel, mais difficile à démontrer.

Comme le mentionnent Jennings et Wooldridge [76], les chercheurs doivent continuer le travail, car il reste plusieurs défis importants à relever comme l'hétérogénéité, le raisonnement avec des informations incertaines, incomplètes et contradictoires, l'opération en temps réel et l'adaptabilité. Éventuellement, d'après Nwana et Ndumu [118], de nouveaux problèmes

sociaux et éthiques surviendront à mesure que la technologie des agents sera présente dans la société :

- Le respect de la vie privée : il faut assurer que l'agent ne divulgue pas de détails de la vie privée de son propriétaire lorsqu'il pose des actes en son nom.
- La responsabilité des actes : il faut non seulement être conscient du transfert des responsabilités du propriétaire vers l'agent, mais aussi des conséquences lorsque les choses tournent mal.
- Les considérations légales : qui est responsable lorsqu'un agent, qui agit au nom de son propriétaire, cause des dommages ou pose des actions incorrectes malgré le bon vouloir du propriétaire lors de l'ajustement des paramètres de l'agent ?
- Les considérations d'éthique : quels comportements l'agent doit-il adopter dans différents contextes ? L'éthique est un sujet qui a déjà été abordé. Par exemple, une étiquette a été proposée pour les agents qui récoltent de l'information sur internet.

Pour que la technologie des agents ou toute autre technologie devienne populaire, il faut résoudre de nouveaux problèmes qui n'étaient pas résolus ou résoudre d'une manière plus naturelle, facile, efficace ou rapide des problèmes qui sont déjà solutionnés. La technologie des agents, si elle tient ses promesses, peut devenir la technologie du futur en informatique. À ce moment, les agents seront intégrés graduellement dans des objets de tous les jours, probablement en remplaçant des technologies existantes pour ensuite s'attaquer à des applications de plus en plus complexes.

Dans le contexte des travaux de recherche présentés, les problèmes liés à l'architecture et à l'hétérogénéité des agents ainsi que leur mise en oeuvre sont abordés. Le chapitre 5 permet d'analyser et de solutionner ces problématiques.

Les agents sont rarement utilisés seuls. Généralement, plusieurs agents sont impliqués pour accomplir un travail ; ils forment alors un système multi-agents. Ces systèmes sont très prometteurs et ils sont le sujet du prochain chapitre.

CHAPITRE 3

SYSTÈMES MULTI-AGENTS

Plusieurs agents utilisés dans un même système forment un système multi-agents (SMA). Des agents qui communiquent permettent d'interconnecter des systèmes et de les faire interagir. Si la situation le requiert, les agents peuvent coordonner leurs activités ou encore collaborer pour résoudre le problème auquel ils font face.

3.1 Introduction

Les SMA utilisent l'agent comme abstraction de base. Selon Wooldridge [154], les systèmes qui font intervenir plusieurs agents sont parmi les plus prometteurs. Les SMA, étant basés sur les agents, héritent des avantages et des inconvénients liés aux agents tels que décrits au chapitre 2. En cachant les détails de conception et d'implémentation derrière l'abstraction d'agent, il est possible de se pencher sur les problèmes uniquement liés aux SMA et à leur conception.

Les SMA sont le sujet de ce chapitre. La section 3.2 donne une définition des SMA et discute des motivations et des intérêts qui s'y rattachent. La section 3.3 donne une classification des SMA. Les problématiques principales liées directement à la nature des SMA sont ensuite exposées : l'intégration d'agents dans un SMA à la section 3.4, la communication entre les agents à la section 3.5 et la coordination des agents à la section 3.6. Ce chapitre montre aussi, à la section 3.7, que des outils de débogage sont nécessaires afin de faciliter la mise en oeuvre de SMA. Les types d'environnements existants sont aussi discutés à la section 3.8 et le chapitre se termine par une discussion et quelques observations sur les SMA à la section 3.9.

3.2 Définitions, motivations et intérêts

La définition d'un SMA est plus unanime, contrairement à celle de l'agent. Un SMA est généralement défini comme suit :

Système multi-agents : Un ensemble d'agents autonomes qui coexistent dans un même monde et qui peuvent collaborer.

Selon Tokoro [145], les agents sont habituellement réputés être continus dans le temps et capables de poser leur prochaine action en consommant de manière limitée les ressources, comme le temps ou la mémoire. Ce type de comportement est particulier aux SMA et s'appelle **rationalité limitée** (*bounded rationality*). Elle a comme conséquence que les SMA ont des comportements complexes et difficilement déductibles du comportement de chaque agent.

Bien que cette définition soit adoptée par plusieurs chercheurs, certains y ajoutent quelques nuances. Par exemple, Tokoro [145] requiert que le système comporte une notion de **soi**. D'autres, comme Singh [136], considèrent tout système distribué comme un SMA.

Peu importe la définition adoptée, la question fondamentale des SMA est la suivante : est-ce que chaque agent peut bénéficier du travail de groupe plutôt que de travailler seul ? En d'autres termes, est-ce qu'il y a des bénéfices à travailler en groupe ? Tous s'entendent à dire qu'il y a effectivement des bénéfices, mais à condition que les difficultés intrinsèques aux SMA ne viennent pas contrebalancer les bénéfices.

D'après Genesereth et Ketchpel [56], le travail de groupe, comme dans le cas des SMA, comporte des difficultés dont une des plus importantes est l'hétérogénéité des agents impliqués. Les agents peuvent provenir de différentes sources, être conçus avec des architectures et des langages utilisant des paradigmes différents, et s'exécuter sur des systèmes d'opération différents. Ces produits logiciels hétérogènes, qui fonctionnaient souvent en isolation, sont de plus en plus sollicités pour travailler en groupe. Ils doivent interagir avec d'autres systèmes. Un SMA doit permettre l'interopérabilité de ces systèmes hétérogènes.

Bien que les SMA puissent engendrer des bénéfices en collaborant, une motivation plus grande doit être présente pour justifier les efforts investis dans les SMA. Jennings et Wooldridge [76] ainsi que Moulin et Chaib-draa [107] énumèrent plusieurs avantages et intérêts importants en faveur des SMA :

- Les SMA sont bien adaptés aux **systèmes ouverts**. Les systèmes ouverts sont des systèmes dont les composants ne sont pas connus d'avance, qui peuvent changer avec le temps et qui sont hautement hétérogènes. Avec ce type de système, l'interface de communication entre les agents doit être flexible et robuste, car les interactions sont peu connues au moment de la conception. Les agents doivent avoir des capacités sophistiquées, comme la persuasion, la négociation et des mécanismes de coordination. Les agents doivent continuellement superviser leur environnement et réviser leurs objectifs pour adopter de nouveaux buts quand l'opportunité se présente.
- Les SMA gèrent bien la **complexité**. Un problème est complexe lorsque le domaine est large, sophistiqué ou imprévisible et que la seule façon raisonnable de le solutionner est de développer des composants modulaires spécialisés dans leurs représentations et leurs paradigmes.
- Les SMA s'adaptent bien à la **distribution**. Un problème est distribué lorsque plusieurs entités sont impliquées pour résoudre des parties du problème et qu'elles sont distinctes physiquement ou logiquement distribuées en ce qui concerne les données, le contrôle, l'expertise ou les ressources. Dans ce cas, il est naturel de modéliser un tel système par un SMA.
- Les SMA sont basés sur une **métaphore naturelle**. La notion d'agent autonome peut être une façon plus naturelle de conceptualiser ou de présenter une fonctionnalité logicielle.
- Les SMA permettent la **récupération**. Il est possible de récupérer des systèmes existants et de les intégrer dans un SMA.

Les SMA peuvent être composés d'agents faciles à entretenir, à développer et à spécialiser pour résoudre un problème de manière efficace. Chaque agent étant autonome, la meilleure méthode pour chacun des sous-problèmes peut être choisie sans compromis. Les SMA sont plus qu'une simple variante de la méthodologie **diviser pour conquérir**. Les agents doivent être capables d'interagir dans le contexte présent et avec flexibilité. De là émerge le besoin de caractéristiques comme l'habileté sociale, la réactivité et la proactivité, plutôt qu'une interface fixe et prédéterminée. Les agents doivent être opportunistes tout en étant réactifs et proactifs.

Singh [136] mentionne que les SMA peuvent être utilisés pour aborder des problèmes distribués pratiques et simples et pour aborder des problèmes complexes et ouverts. Il donne

un premier exemple simple : si le grille-pain et la cafetière pouvaient communiquer, alors les deux pourraient coordonner leurs activités afin que les rôties et le café soient prêts en même temps à la grande satisfaction du consommateur. Il donne aussi un deuxième exemple plus complexe et ouvert qui représente une situation plus typique des SMA, soit l'atterrissage d'un avion. Pour les fins de l'exemple, le problème est représenté par un agent pilote et deux agents contrôleurs aériens. Cet exemple est illustré à la figure 3.1. Ces trois agents peuvent être humains ou non.

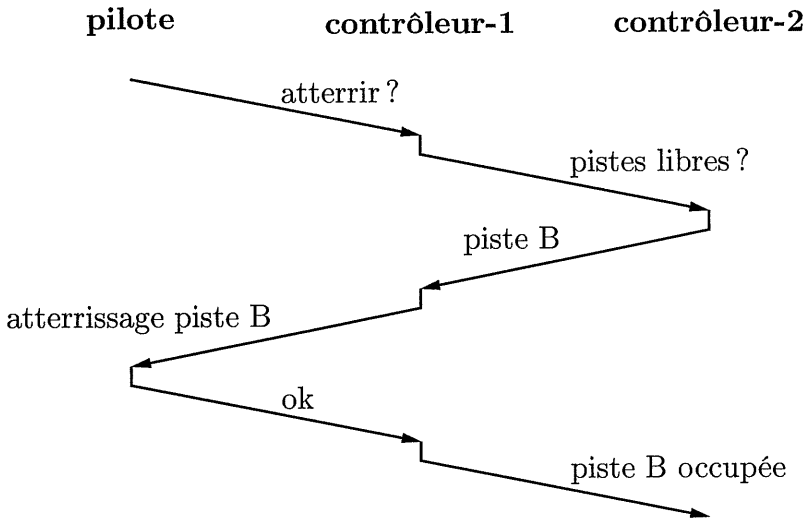


Figure 3.1 – Exemple de système multi-agents : contrôle de trafic aérien [136].

Lorsque l'avion approche de l'aéroport, le pilote demande une permission d'atterrissage à un contrôleur, le contrôleur-1. Le contrôleur-1 ne sait pas s'il y a des pistes de libres, alors il s'informe auprès du contrôleur-2. Le contrôleur-2 répond au contrôleur-1 que la piste B est libre. Le contrôleur-1 offre alors la piste B au pilote, mais à condition qu'il accepte de payer les frais applicables. Le pilote accepte de payer et le contrôleur-1 assigne donc la piste B à cet avion et prohibe le contrôleur-2 d'assigner cette piste à un autre avion.

Cet exemple de contrôle de trafic aérien illustre plusieurs caractéristiques des SMA :

- Diversité des types de messages, comme des demandes, des questions, des affirmations, des permissions, des promesses, des prohibitions, etc.
- Diversité des protocoles pour ces messages.
- Respect des interfaces et des comportements. Les agents peuvent être changés ou améliorés de manière transparente, à condition que les interfaces et les comportements soient respectés.

Les SMA sont une technologie prometteuse, mais plusieurs problèmes restent à régler. Les différentes facettes des SMA et leurs problématiques sont exposées dans les sections suivantes.

3.3 Classification

Les agents qui composent un SMA peuvent être homogènes ou hétérogènes. Dans un SMA avec agents homogènes, tous les agents ont les même caractéristiques. Dans un SMA avec agents hétérogènes, les différences sont caractérisées par la capacité de raisonnement des agents ou encore par les paradigmes utilisés, comme les base de connaissances et les réseaux de neurones artificiels. Les SMA à agents hétérogènes apparaissent plus adaptés pour résoudre des problèmes réels, car ils offrent la possibilité d'intégrer tout type de raisonnement ou de traitement pouvant contribuer à une solution.

Le tableau 3.1 énumère les caractéristiques des SMA telles qu'identifiées par Huhns et Singh [72].

Caractéristique	**Valeurs possibles**
unicité	homogène à hétérogène
granularité	fine à grossière
structure de contrôle	hiérarchique à démocratique
autonomie d'interface	communication : spécifie vocabulaire, langage, protocole intellect : spécifie buts, croyances, ontologies habiletés : spécifie procédures, comportements
autonomie d'exécution	indépendant à contrôlé

Tableau 3.1 – Caractéristiques des systèmes multi-agents [72].

Généralement, les agents constituant un SMA échangent des informations. Le type d'information dépend du niveau de sophistication et du type des agents qui sont impliqués. Deux types de modèle de communication ont émergé des travaux de recherche, le modèle du **tableau noir** et le modèle d'**acteurs**.

3.3.1 Modèle du tableau noir

Le modèle du tableau noir de Nii [115], illustré à la figure 3.2, utilise des sources de connaissances (SC) pour encoder les comportements. Selon la stratégie de contrôle utilisée, les SC

du système sont activées ou désactivées lorsque certaines informations sont présentes ou non dans la base de données globale, soit le tableau noir.

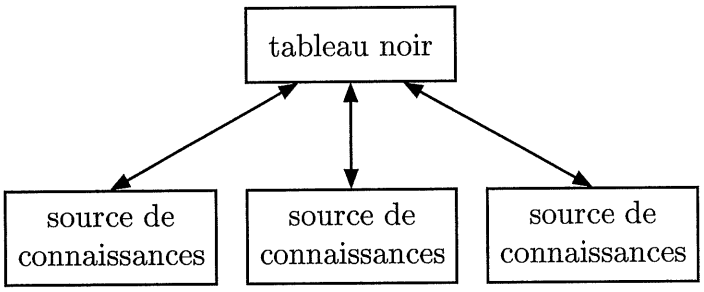


Figure 3.2 – Modèle du tableau noir [115].

Farhoodi et Graham [45] ajoutent qu'un tableau noir peut être divisé en plusieurs sections, appelées niveaux d'abstraction, afin que chaque SC ne puisse accéder qu'à certaines parties des connaissances. Un ordonnanceur décide de l'ordre d'exécution des SC qui ont été déclenchées.

Müller [108] explique aussi que la communication entre les SC est faite en lisant ou en écrivant dans le tableau noir. Les SC communiquent donc indirectement au travers du tableau noir en y écrivant et en y lisant des solutions ou des solutions partielles. La présence de plusieurs tableaux noirs, possiblement hiérarchisés, permet de faire du traitement en parallèle. Le modèle du tableau noir a généralement été réalisé sur une seule machine, mais des versions distribuées ont également été réalisées. Les SC sont perçues par certains comme des agents à granularité fine. Généralement, ce type d'agent n'est pas considéré intelligent malgré que le comportement global du système puisse l'être.

3.3.2 Modèle d'acteurs

Le modèle d'acteurs de Hewitt [65], illustré à la figure 3.3, est composé d'un ensemble d'agents appelés acteurs qui communiquent à l'aide de messages. L'ensemble des acteurs du système représente la solution au problème global et chaque acteur en solutionne une partie. L'implémentation de ce modèle peut être réalisée sur une seule machine ou être physiquement distribuée sur plusieurs.

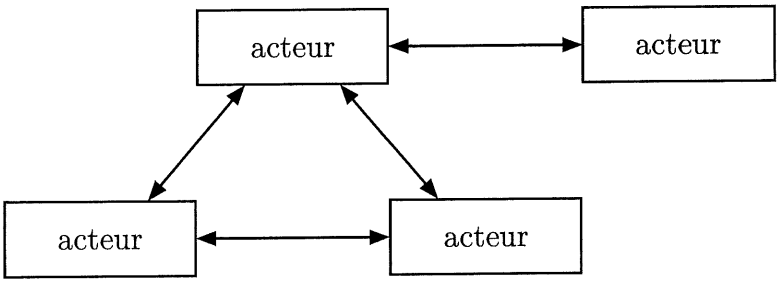


Figure 3.3 – Modèle d'acteurs.

Farhoodi et Graham [45] expliquent que chaque acteur peut agir intelligemment et a une vie indépendante des autres. Il n'y a pas de mécanisme central de coordination ; ce modèle requiert donc des mécanismes de planification et de coordination de tâches afin que les acteurs puissent planifier intelligemment leurs actions en fonction des autres. Le comportement global du système émerge des interactions des acteurs plutôt que d'être fixé *a priori*.

Les liens de communication entre les acteurs peuvent être établis de manière quelconque selon les besoins. Par contre, une architecture hiérarchisée comme les systèmes fédérés (*federated systems*) est une architecture qui a beaucoup d'adeptes [56, 118]. Cette approche, illustrée à la figure 3.4, a deux avantages comparativement à la communication directe lorsque le nombre d'acteurs est grand. Le premier avantage est qu'elle peut réduire le nombre de connexions établies et possiblement réduire le nombre de messages échangés. Le deuxième avantage est qu'elle réduit la complexité des capacités de communication de chacun des agents. Cette complexité est transférée dans un nombre limité de *facilitators* qui sont des agents spécialisés dans ce domaine.

Guessoum et Dojat [60] mentionnent que le modèle d'acteurs est perçu comme l'architecture de choix pour les SMA. Les acteurs sont considérés comme les blocs de base des SMA. Par contre, du point de vue d'un environnement de développement, les modèles les plus courants doivent être offerts aux concepteurs de SMA. Les concepteurs doivent pouvoir choisir le modèle le plus approprié.

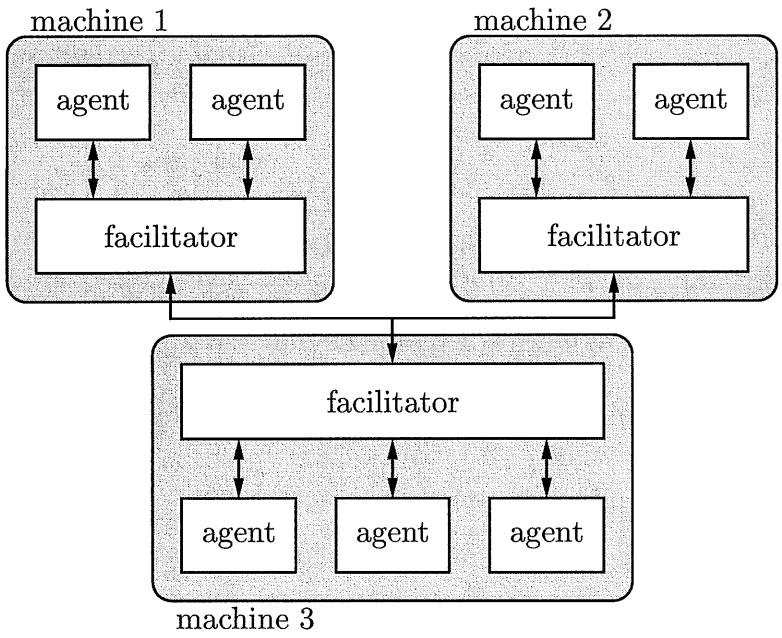


Figure 3.4 – Modèle d'acteurs structuré en un système fédéré [56].

3.4 Intégration d'agents

Les concepteurs d'un SMA doivent intégrer plusieurs agents dans un même système. En simplifiant, cette tâche consiste à permettre à l'agent de communiquer avec les autres agents du système. Lorsqu'un nouvel agent est conçu, ses capacités de communication font partie de sa conception et il est facilement intégré au SMA.

Les choses sont différentes si un des agents existait avant le SMA. Dans ce cas, l'agent ne possède pas nécessairement toutes les capacités de communication requises lorsque vient le temps d'intégrer l'agent dans le système. Par exemple, les protocoles de communication utilisés dans le SMA ne sont peut être pas reconnus par l'agent à intégrer. Genesereth et Ketchpel [56] donnent trois approches pour intégrer un agent ; elles sont illustrées à la figure 3.5.

Traduction

Il s'agit de concevoir un agent traducteur servant d'intermédiaire entre l'agent à intégrer et les autres agents du système. L'agent traducteur fait de la traduction et de la médiation entre l'agent à intégrer et les autres agents. Il accepte les messages de l'extérieur, les traduit

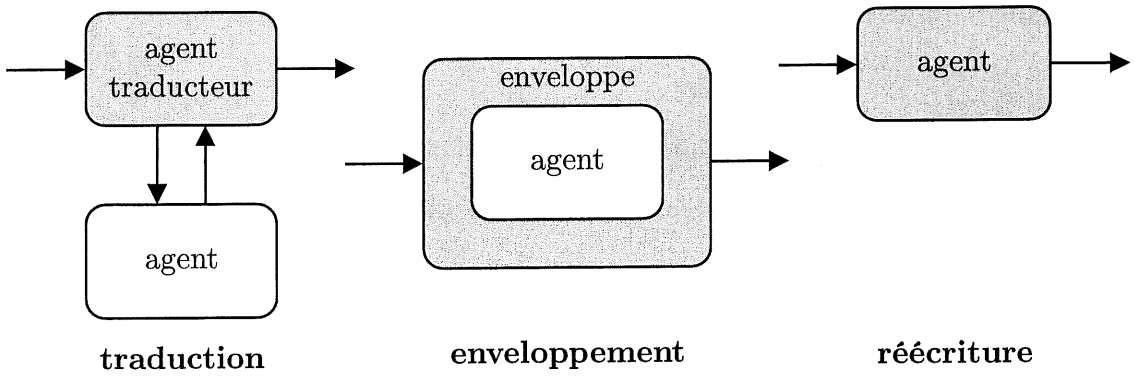


Figure 3.5 – Approches d'intégration d'agents : traduction, enveloppement et réécriture [56].

dans les protocoles reconnus par l'agent à intégrer et les lui transmet. L'agent traducteur récupère les résultats ou les messages de l'agent à intégrer, les traduit dans les nouveaux protocoles et les envoie aux agents externes concernés.

Cette méthode a l'avantage de demander de connaître uniquement les habitudes de communication de l'agent à intégrer. Elle est rapide et utile pour les agents dont le code n'est pas disponible ou trop délicat pour être modifié. Elle fonctionne aussi avec d'autres ressources, comme les humains. Par exemple, il suffit d'une interface graphique à l'agent traducteur, pour qu'il puisse acquérir les données, les traduire et les transmettre aux autres agents du SMA.

Enveloppement

Il s'agit d'ajouter du code à l'agent à intégrer afin qu'il acquière les capacités de communication nécessaires. Cette approche a de bonnes chances d'être plus efficace que la traduction, car l'agent traducteur est éliminé. Elle a aussi l'avantage de fonctionner pour les agents à intégrer qui n'avaient aucune capacité de communication. Par contre, elle requiert que le code source de l'agent à intégrer soit disponible.

Réécriture

Il s'agit de réécrire au complet l'agent à intégrer en lui ajoutant des fonctionnalités de communication. Cette approche est la plus longue à réaliser, mais elle peut permettre d'augmenter

les performances ou les capacités de l'agent à intégrer de manière beaucoup plus significative que les deux autres méthodes.

Ces différentes méthodes d'intégration d'agents doivent être offertes par un environnement de développement afin d'assurer l'intégration d'agents dans un système et la conservation de développements antérieurs.

3.5 Communication

Une raison majeure d'utiliser l'approche multi-agents, selon Nwana et Wooldridge [120], est qu'elle ajoute une plus-value comparativement à une conception monolithique basée sur un seul agent. Cette valeur ajoutée est typiquement atteinte par la coopération des agents du SMA et donc par la communication. Afin de coopérer efficacement, les agents doivent communiquer de manière tout aussi efficace. Les agents doivent comprendre et utiliser un **langage de communication agent** (LCA).

La caractéristique la plus importante d'un LCA est son expressivité. L'importance d'un bon LCA est bien soulignée par Genesereth et Ketchpel [56] ; il permet d'échanger des données, des informations logiques, des commandes et des scripts. En utilisant ce langage, les agents peuvent communiquer des informations complexes et des buts. Pour certains chercheurs, comme Genesereth et Ketchpel, l'importance du LCA est telle qu'ils basent leur définition d'agent sur cette notion : « *An entity is a software agent if and only if it communicates correctly in an agent communication language* ».

Un LCA utilise une sémantique indépendante de l'implémentation des agents et du type des données. Le concept d'agent soulève des questions importantes en ce qui concerne les LCA : quel langage est approprié pour un LCA ? Comment concevoir et implémenter des agents utilisant un LCA ? Quelle architecture de communication facilite la coopération ?

Le langage KQML (*Knowledge Query Manipulation Language* [48]) est un LCA qui est très utilisé. KQML est un langage de communication pour échanger des informations et des connaissances. Une douzaine d'**actes** sont au coeur de KQML. Ces actes peuvent être utilisés par un agent et ils fournissent une base afin de construire des stratégies de coordination ou

de négociation plus complexes. Aussi, le langage FIPA-ACL (FIPA-Agent Communication Language [49]) est un ACL semblable à KQML, mais qui a l'avantage d'avoir sa sémantique standardisée.

Selon Mataric [102], la communication est le type d'interaction la plus courante dans les SMA. Le type de communication dépend de la stratégie adoptée pour résoudre le problème et donc de la manière dont le problème est présenté à l'agent et de quels services ou de quelles ressources il a besoin. La communication se manifeste de manière directe ou indirecte :

Communication directe : Une action dont le seul but est de transmettre de l'information. Le message n'a pas besoin d'être symbolique. Une communication dirigée est une communication directe qui vise un individu en particulier. La communication dirigée est faite de **un à un** ou de **un à plusieurs**. Les récipiendaires sont identifiés dans tous les cas.

Communication indirecte : Elle est basée sur l'observation du comportement des autres agents. C'est une communication qui est basée sur les changements qui s'effectuent dans l'environnement plutôt que sur la réception directe de messages. Par exemple, Chaib-draa et Levesque [29] ont expérimenté avec des signes, des signaux et des symboles comme moyens de communication.

Un environnement de développement doit permettre l'utilisation ou non de différents LCA ; les concepteurs doivent pouvoir choisir le LCA le plus approprié pour l'application à implémenter. De la même manière, tout type de communication doit pouvoir être utilisé.

3.6 Coordination

Utiliser un SMA pour implémenter certains types d'applications peut être avantageux, mais ce choix de conception apporte aussi ses problèmes. La coordination est un thème central aux SMA. Sans elle, mentionnent Chaib-draa et Levesque [29], les bénéfices des interactions des agents disparaissent, car le système dégénère rapidement en un groupe d'agents individualistes avec un système ayant un comportement chaotique. La coordination est une nécessité et Nwana et Jennings [117] le démontrent bien en lui donnant plusieurs justifications :

- prévention de l'anarchie et du chaos ;
- respect de contraintes globales ;
- expertises, ressources ou informations distribuées ;
- dépendances entre les agents ;
- efficacité ;
- allocation des tâches et des ressources entre les agents ;
- reconnaissance et résolution d'incohérences ou de conflits dans les buts, les faits, les croyances, les points de vue et les comportements.

Sans mécanismes de coordination, un agent est amené à décider et à agir en se basant sur ses vues locales qui peuvent être incomplètes, inconsistantes, contradictoires ou périmées. Un agent doit utiliser ses ressources non seulement pour la tâche à accomplir et ses besoins de communications, mais aussi pour gérer ses actions et interactions. D'après Durfee et al. [41], le problème principal de la coordination dans les SMA est dû à l'absence de mécanismes centralisés de contrôle.

Le contrôle et la coordination d'agents distribués font intervenir des considérations comme la reconnaissance des interactions possibles entre les agents, la promotion du parallélisme, l'intégration de résultats locaux dans des solutions complètes, la résolution d'inconsistances dans les résultats locaux et la communication d'informations pertinentes aux moments appropriés.

Pour la coordination, selon Singh [136], la considération la plus importante est que les agents doivent avoir la capacité d'acquérir et d'utiliser une représentation des autres agents. Ceci est un prérequis afin que les agents puissent négocier, coopérer, coordonner et apprendre. Pour cela, quatre éléments sont nécessaires [41, 117] :

- Une **structure** : elle permet aux agents d'interagir de manière prévisible.
- De la **flexibilité** : c'est elle qui permet aux agents d'agir dans un environnement dynamique et de gérer des situations dans lesquelles ils ne possèdent qu'une vue partielle ou imprécise.
- Une **structure sociale** : elle permet de décrire comment les agents doivent se comporter les uns envers les autres lorsqu'ils sont engagés dans des mécanismes de coordination.
- Des **connaissances** et du **raisonnement** : les agents doivent posséder suffisamment de connaissances et avoir une capacité de raisonnement suffisante afin qu'ils puissent exploiter la structure et la flexibilité.

La coordination est le sujet de nombreuses recherches et elle a engendré quatre grandes classes de techniques : structure organisationnelle, structure contractuelle, planification multi-agents et négociation. Ces techniques sont résumées dans [40, 41, 108, 117, 150].

Les SMA offrent des bénéfices potentiels pour des applications où les informations, les ressources et l'expertise sont distribuées ou encore lorsque la distribution est avantageuse pour la performance, la modularité ou la fiabilité du système. Par contre, ces bénéfices sont inexistants si les agents ne sont pas coordonnés adéquatement.

Nwana et Jennings [117] font plusieurs remarques intéressantes sur les stratégies de coordination. Plusieurs techniques ont été proposées, mais il y a peu de support empirique ou théorique pour chacune d'entre elles. Dans ce sens, d'autres études doivent être faites afin de les valider. Il n'est dont pas clair comment, quand et pourquoi utiliser une stratégie ou une combinaison de stratégies pour une application donnée.

Les stratégies de coordination maître-esclave et le *Contract Net Protocol* (CNP) [34] semblent être les plus courantes à cause de leur simplicité. Plusieurs techniques, dont le CNP, assument que les agents sont bienveillants, fiables, non-conflictuels et aidants. Ceci est irréaliste dans beaucoup d'applications, comme le commerce électronique, car des agents malhonnêtes pourraient prendre avantage du système. De plus, la plupart de ces stratégies ne font pas de raisonnements complexes sur le domaine en question pendant la négociation, ce qui est souvent requis. Par exemple, peu d'approches tiennent compte du temps et du fait que les buts, les croyances et les intentions des agents changent avec le temps. Malgré les avancements et les réalisations, aucune des techniques de coordination n'est assez évoluée pour être une solution générale à tous les problèmes d'une coordination intelligente d'agents.

De la même manière qu'avec les communications, un environnement de développement doit permettre l'utilisation de tout type de stratégie de coordination et même de permettre l'utilisation d'une combinaison de stratégies. Les concepteurs d'un SMA doivent pouvoir choisir la méthode de coordination la plus appropriée pour l'application à réaliser.

3.7 Débogage de systèmes multi-agents

Historiquement, selon Ndumu et Nwana [113], le débogage de SMA n'est pas un sujet qui attire l'attention de beaucoup de chercheurs, mais, depuis quelques années, il connaît un certain regain de vie.

La mise au point d'un SMA peut être un travail ardu et compliqué, car les SMA sont des systèmes qui varient dans leur complexité, leurs stratégies de contrôle, leurs protocoles d'interactions et leurs architectures. Peu importe la nature du SMA, le développeur a besoin de savoir et de voir, car il veut vérifier les états du système pendant l'exécution.

Van Liedekerke et Avouris [148] résument bien la problématique du débogage de SMA. Un des problèmes majeurs avec les systèmes distribués est qu'il est impossible pour le développeur d'avoir une vue correcte de tous les états internes des agents, car les délais de communication rendent périmées les informations reçues. De plus, les informations reçues de plusieurs entités distinctes exécutées en parallèle ne peuvent être ordonnées correctement qu'avec des mécanismes d'estampage temporel. Les informations reçues ne peuvent être présentées au concepteur que de manière sérielle ; elles sont donc une approximation de la réalité.

Selon Van Liedekerke et Avouris [148], la seule manière d'avoir une vue correcte du système est de faire en sorte que les événements d'intérêt soient marqués d'une estampille temporelle et qu'ils soient sauvegardés localement. Suite à l'exécution, les traces laissées permettent de reconstituer une vue postmortem exacte. L'estampille temporelle est un paramètre important qui doit être standardisé, car c'est elle qui permet de reproduire une vue correcte.

L'inspection des traces est basée sur des techniques issues de la gestion de bases de données relationnelles. Les traces peuvent être perçues comme une relation dont les attributs sont le numéro de noeud, le type d'événement, le numéro de processus, l'estampille temporelle, le numéro de transaction et les données spécifiques à l'événement. Pour examiner les données, un langage relationnel ou des outils de visualisation graphique en-ligne (*on-line*) ou hors-ligne (*off-line*) sont utilisés pour rejouer les événements de manière contrôlable.

Plusieurs systèmes de visualisation de traces ont été proposés, mais ils ont tous un point commun ; ils séparent la collection des données de la visualisation des données. Les tâches de détection et de collection sont découplées des tâches d'analyse et d'affichage des informations, comme illustré à la figure 3.6 [148].

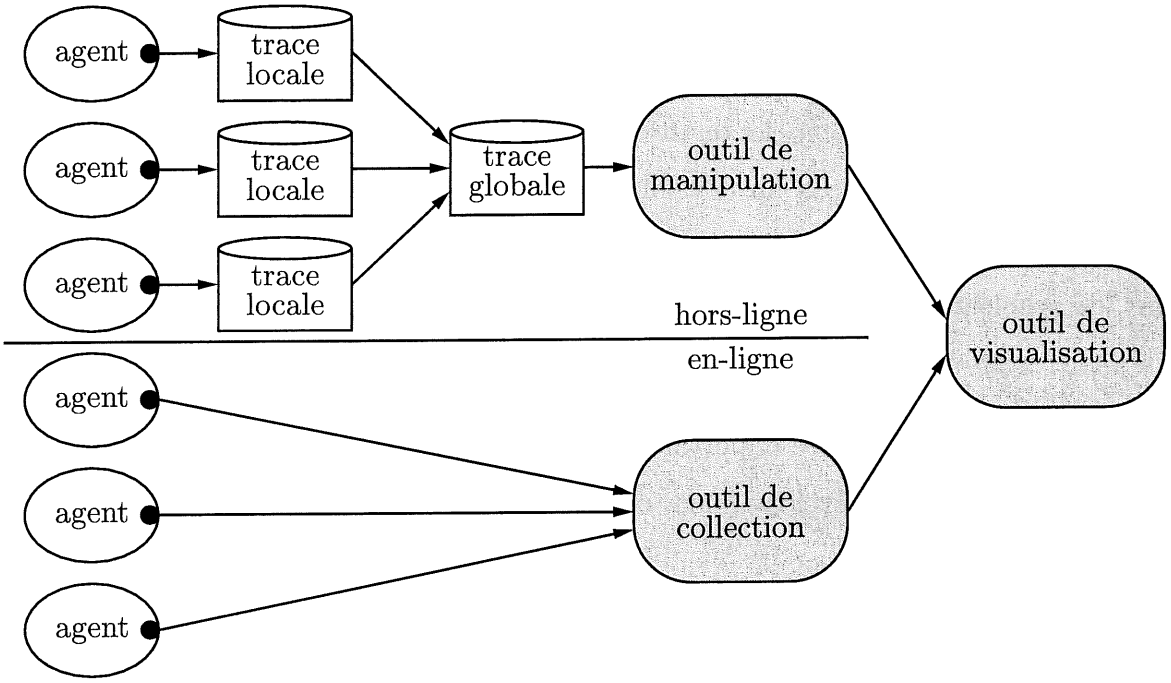


Figure 3.6 – Outils de débogage de systèmes multi-agents [148].

Lorsque vient le temps de visualiser les informations, les données peuvent être présentées de manière brute ou traitées. Van Liedekerke et Avouris [148] proposent dans leur système plusieurs vues alternatives des données. Ces vues sont appelées le modèle conceptuel du développeur (*Developer Conceptual Model* ou DCM). Le DCM décrit le choix de métaphore ou de représentation des concepteurs pour guider la façon d'interpréter les actions du SMA. Le DCM aide à préciser les aspects d'intérêt particulier du système. Plusieurs DCM ont été identifiés, la plupart sont reliés aux différents stages du développement d'un SMA :

1. modèle du domaine ;
2. modèle de l'agent ;
3. modèle de la distribution ;
4. modèle d'interaction ;
5. modèle de coopération ;

6. modèle des connaissances ;
7. modèle d'architecture.

Certains systèmes de débogage ont des agents supplémentaires pour observer les processus. Ces agents développeurs créent les fichiers de traces, gèrent les erreurs, surveillent l'exécution et gèrent les interactions avec l'usager.

Le contrôle des traces est l'une des questions majeures lors de la phase de débogage. Lorsque les traces ne conviennent pas au développeur, le système peut être exécuté pas à pas de manière contrôlée pour ainsi permettre l'observation des états intermédiaires de chacun des agents. Le problème est de synchroniser la pause et le redémarrage des agents pour l'exécution pas à pas et surtout de définir ce qu'est un pas pour un système distribué hétérogène.

Le débogage étant une facette importante du cycle de vie d'un SMA, il est donc important qu'un environnement de développement offre des outils adaptés au débogage de SMA.

3.8 Environnements existants

Les chercheurs conçoivent des environnements de développement multi-agents afin de valider les théories, les idées et les concepts ou encore de développer et de réaliser des SMA. Ces environnements ne donnent pas tous les mêmes libertés aux concepteurs et ils sont dédiés aux tests ou au développement.

Bouron et al. [15] classent les environnements de test comme étant de type 1 ou de type 2. Les environnements de type 1 permettent de tester une catégorie spécifique d'agent ou une technique spécifique de coordination. Ce type d'environnement fixe certains paramètres. Par exemple, l'application est fixée et l'environnement sert à valider une technique de coordination. Des exemples[1] de ce type d'environnement sont DVMT [89], ECO [46] et RATMAN [25].

Les environnements de type 2 sont des environnements généraux de tests. Ils sont généraux dans le sens où ils permettent de tester différentes techniques de coordination ou d'agents

[1]Seulement quelques environnements par catégorie sont cités, car tous les citer est inutile pour la discussion.

et de les comparer. Ce sont souvent des environnements de simulation. Des exemples de ce type d'environnement sont MACE [55], MICE [42], PANDORA-II [101], MAGES [15], SIM_AGENT [137] et GENSIM [5].

Les environnements de tests de type 1 et de type 2 servent surtout à démontrer la faisabilité d'une approche ou encore servent à résoudre des problèmes dans un domaine particulier. Ces environnements sont limités, mais ils sont importants, car ils permettent de valider les approches et les idées avancées.

Il existe aussi des environnements généraux de développement et de mise en oeuvre d'applications basées sur les SMA. Ces types d'environnements permettent normalement de développer tous les types d'applications. Des exemples de ces types d'environnements sont ARCHON [75, 153], ATOME [98], JADE [10], *Open Agent Architecture* [99] et ZEUS [32, 116].

Bien que ces environnements permettent de développer tout type d'application, peu d'entre eux donnent toutes les libertés aux concepteurs. Certains paramètres sont fixés, comme l'architecture des agents, le paradigme de programmation, les protocoles de communication ou la technique de coopération. Par exemple, un environnement récent nommé ZEUS, réalisé en JAVA, est portable et offre toutes les fonctionnalités typiques des environnements de développement modernes. Par contre, les concepteurs peuvent uniquement développer des agents selon l'architecture et le modèle qui sont proposés. Les concepteurs, bien que travaillant avec un outil moderne, ne peuvent pas choisir le paradigme le plus approprié pour la tâche et les sous-tâches à accomplir. Les concepteurs n'ont pas toutes les libertés.

Les environnements de développement se distinguent des plates-formes d'agents. Un environnement de développement offre des outils pour le développement d'agents alors qu'une plate-forme d'agents offre des services à un agent et l'agent doit suivre certaines procédures afin qu'il puisse y évoluer de manière adéquate. Une plate-forme d'agents, comme celle proposée par la FIPA [50], n'impose pas de structure interne d'agent, mais propose plutôt des conventions de fonctionnement afin d'accéder à des services. Certains environnements comme JADE [10] offrent des outils de construction d'agents tout en offrant les services de la plate-forme FIPA.

Un environnement de développement de SMA qui soit vraiment général avec agents hétérogènes, indépendant de l'application et de la structure de communication, portable, extensible, flexible, n'existe pas encore. La recherche sur les SMA et la communauté informatique bénéficieraient énormément d'un tel type d'environnement. Les concepteurs pourraient alors choisir les modèles de communication désirés, choisir le paradigme de programmation le plus approprié pour chaque agent et opter pour la technique de coordination la plus adaptée. Ils pourraient aussi étendre l'environnement avec leurs types d'agents.

3.9 Discussion

Les SMA sont un domaine de recherche très actif. Ils peuvent être classifiés selon les architectures et les types de communication, mais des problèmes subsistent.

Von Martial [150] mentionnait qu'un des problèmes qui amène la confusion est le manque de définitions formelles d'utilité pratique. Ce problème subsiste toujours. Les écarts entre l'approche logique et l'approche pratique en sont une conséquence directe. La théorie définit certains concepts, mais elle est très loin de la pratique et la pratique manque de degrés d'abstraction.

Plusieurs des problèmes des SMA sont reliés à la difficulté d'analyser leurs comportements, selon Mataric [102]. Cette difficulté est issue de deux propriétés intrinsèques des SMA :

1. Les actions des agents dépendent de l'état et des actions des autres.
2. Le comportement du SMA est déterminé par les interactions entre les agents plutôt que par leurs comportements individuels.

Cette difficulté à analyser les comportements des SMA a de fâcheuses conséquences, selon Jennings et Wooldridge [76] :

- Le comportement global du SMA est imprévisible et indéterminé. Il n'y a pas de garanties que les dépendances entre les agents seront gérées efficacement étant donné que les agents prennent leurs propres décisions.
- Le comportement et les propriétés d'un agent peuvent être fixés au moment de sa conception, mais pas ceux du SMA. Le comportement global du SMA émerge uniquement à l'exécution.

La nature distribuée des SMA ajoute aux difficultés de leur développement et de leur débogage. Elle cause aussi des problèmes liés à la coordination et à l'établissement de plans ou de compromis afin d'en arriver à une solution acceptable pour tous les agents. Ainsi, d'autres domaines liés aux SMA sont aussi le sujet de beaucoup de travaux de recherche, comme l'apprentissage, la planification, la négociation et la gestion de l'incertitude. Ces sujets sont cruciaux pour les SMA, car ils abordent des problématiques importantes liées aux comportements de groupe. Bien qu'elles soient importantes, ces problématiques ne sont pas abordées dans les travaux de recherche présentés ici, car ils n'affectent pas directement la conception d'un environnement générique de développement.

Singh [136] mentionne le besoin de techniques de spécification et de formalisation de haut niveau. Le manque d'une théorie de l'interaction entre les agents force les concepteurs de SMA à penser en termes des détails de la plate-forme sous-jacente. Pour les SMA, ces détails sont relégués dans les agents et sont donc cachés dans les agents. Le problème n'est pas pour autant résolu. Ce manque de théories d'utilité pratique a plusieurs conséquences pour les chercheurs et pour la recherche sur les SMA :

- Les interactions entre les agents sont conçues à partir de zéro à chaque fois.
- La sémantique des interactions est implémentée dans différentes procédures qui peuvent contenir des dépendances au système d'opération ou du réseau. Ces dépendances rendent la validation et la modification du SMA encore plus difficile.
- Des systèmes conçus séparément sont difficilement intégrables.
- Il est presque impossible de faire la mise à jour d'un système ou de le reconcevoir de manière élégante. Il est difficile de remplacer un agent par un autre.

Ces limitations vont carrément à contresens de l'idée de concevoir un SMA. Des réalisations importantes ont été faites, malgré les difficultés liées aux SMA et la coopération et la négociation représentent une bonne part des efforts. Beaucoup d'applications et de systèmes logiciels basés sur les SMA ont été réalisés. Par contre, les environnements de développements réalisés jusqu'à maintenant ont surtout servi à démontrer les avantages de certaines approches ou de certaines architectures. De plus, aucune de ces applications et aucun de ces environnements n'énonce les principes pour la conception systématique de SMA. Il n'existe pas d'approche complètement générale pour la construction de SMA hétérogènes de tous les types et pour tous les domaines d'applications.

Bien que les SMA et les systèmes distribués soient prometteurs, plusieurs problèmes et défis importants doivent être abordés et solutionnés [76, 119, 136] :

- Les systèmes distribués doivent permettre l'hétérogénéité pour préserver les investissements passés, pour faciliter l'introduction graduelle de nouvelles technologies et pour optimiser l'utilisation de la plate-forme tout en utilisant les méthodes les plus appropriées pour chacune d'elles.
- Des outils de développement réutilisables doivent être développés. Les chercheurs doivent développer une méthodologie pour la conception d'agents et de systèmes basés sur les agents. Ils doivent aussi développer des bancs d'essais pour l'évaluation des différentes options de conception.
- Les agents doivent pouvoir raisonner avec des informations incertaines, incomplètes et contradictoires. Ils doivent opérer en temps réel et démontrer de l'adaptabilité.
- Les systèmes distribués doivent être prévisibles et contrôlables par l'usager final afin qu'ils se comportent correctement dans des situations critiques et pour octroyer la puissance du système à l'usager.
- Les composants d'un système distribué doivent être localement autonomes afin de mieux gérer la sécurité, les changements incrémentaux et l'obéissance aux lois.

Les SMA représentent une méthode de résolution de problèmes très intéressante pour intégrer des collaborateurs ou des experts de différents domaines et dont l'expertise est représentée sous différentes formes. Martin et al. [99] rappellent que bien que beaucoup de choses aient été réalisées, les SMA ne peuvent pas réaliser leur plein potentiel et devenir omniprésents tant qu'il n'y aura pas d'outils et d'environnements de développement adéquats disponibles. Pour cela, il faut unifier la théorie et la pratique et combler le fossé qui divise les efforts de recherche pour ainsi permettre aux SMA de progresser à grands pas.

Dans le contexte des travaux de recherche présentés, les problématiques liées à la création de SMA hétérogènes à partir de composants réutilisables et portables, et les problématiques liées à la réalisation d'outils génériques portables avec des capacités de débogage sont abordées. Le chapitre 5 permet d'analyser et de solutionner ces problématiques.

Les SMA sont utilisés dans plusieurs domaines d'applications. Chaib-draa [28] fait la revue des plus importants. En particulier, un des domaines d'application intéressants est les véhicules autonomes. Ils sont le sujet du prochain chapitre.

CHAPITRE 4

VÉHICULES AUTONOMES

La réalisation d'un véhicule autonome représente un défi à plusieurs niveaux. La vision, l'analyse du champ visuel, la prise de décision, le contrôle du véhicule, la coordination avec l'environnement et beaucoup d'autres tâches représentent autant de défis. Les véhicules autonomes et la robotique en général sont des domaines d'application intéressants pour les systèmes multi-agents (SMA), car les véhicules autonomes interagissent avec un monde réel et dynamique et ils sont composés de tâches de bas niveau qui s'exécutent en temps réel et de tâches de haut niveau qui constituent l'intelligence du système. Les SMA sont bien adaptés à ce type de problématique.

4.1 Introduction

Selon Hexmoor et al. [66], les véhicules autonomes ou les robots mobiles sont intéressants pour les chercheurs en informatique, non pas parce qu'ils ont des détecteurs ou des moteurs, mais tout simplement parce qu'ils ont une composante matérielle. Les limitations physiques intrinsèques du système matériel imposent au système informatique qui le contrôle de tenir compte du bruit et des pannes, et cela peu importe la configuration utilisée. Le système de contrôle doit gérer les incertitudes nécessairement présentes dans le système.

Les véhicules autonomes et les systèmes informatiques qui les contrôlent interagissent intimement avec un environnement réel. Ils doivent réagir à des changements dans l'environnement qui ne sont pas modélisés, tout en tenant compte des contraintes de temps.

Les véhicules autonomes sont le sujet de ce chapitre. Les types de véhicules autonomes les plus courants sont présentés à la section 4.2. Les différentes tâches accomplies par un

pilote ou par le système informatique qui le remplace sont abordées à la section 4.3. Les considérations et les caractéristiques souhaitables pour un modèle de pilote complet sont discutées à la section 4.4. Les différents types de modélisation où une représentation explicite de l'intelligence du pilote est présente sont exposés aux sections 4.5 à 4.7. Une discussion termine le chapitre à la section 4.8.

4.2 Types de véhicules autonomes

Différents problèmes requièrent différentes solutions. Ainsi, les différents types de véhicules autonomes se distinguent selon leurs différentes utilités et les différents environnements dans lesquels ils évoluent. Selon Smith et Starkey [138], les trois types de véhicules autonomes les plus courants sont les AGV (*Automatic Guided Vehicles*) appelés aussi robots mobiles (*mobile robots*), les ALV (*Autonomous Land Vehicles*) et les AHV (*Automated Highway Vehicles*). Voici une description de chacun de ces types de véhicules autonomes.

AGV et robots mobiles : Ils circulent généralement dans un environnement structuré, comme une usine ou des bureaux. La motivation derrière leur conception est de libérer l'être humain de tâches répétitives, telle la livraison de courrier. Typiquement, les AGV se déplacent à basse vitesse en suivant un chemin indiqué par des marques distinctives, comme de la peinture réfléchissante. Il n'y a donc aucune flexibilité dans le parcours. Certaines recherches sur les robots mobiles, dont celles exposées dans [9, 138], utilisent la vision afin de donner plus de flexibilité au niveau du parcours.

ALV : Ce type de véhicule se déplace sur des routes en mauvaise condition ou sur des terrains vagues. La vitesse de déplacement se situe généralement entre 1 et 24 km/h. La vitesse réduite de ces véhicules est due à la nature accidentée des terrains sur lesquels ils circulent, à leur masse, à leur grandeur et au temps nécessaire pour la planification de trajectoire et de navigation à partir d'un système de vision et de capteurs. La plupart des développements pour ce type de véhicule sont issus des recherches effectuées par l'armée américaine.

AHV : Les AHV sont des véhicules automobiles qui circulent sur des routes en bonne condition à des vitesses plus élevées que les ALV. Ils sont un peu en opposition aux

ALV qui sont des véhicules militaires de grande taille et très massifs. Les AHV circulent sur des routes bien définies, comme les autoroutes. Ils ne sont pas concernés par des conditions en terrain accidenté, comme pour les ALV. Par contre, ils circulent parmi d'autres véhicules et autres objets faisant partie de l'environnement. Des considérations importantes de collisions, de coordination et de sécurité entrent alors en jeux, car des passagers peuvent être à bord du véhicule ou de ceux à proximité.

Parmi ces différents types de véhicules autonomes, les AHV servent pour les travaux de cette thèse qui ont un lien avec les véhicules autonomes.

4.3 Tâches de pilotage

Selon Rochwerger et al. [126] ainsi que Smith et Starkey [138], la diversité des domaines de recherche impliqués dans les solutions apportées pour l'implémentation d'un modèle complet de pilote nécessite l'intégration de solutions provenant de paradigmes différents. À la lumière des recherches déjà effectuées, il semble peu probable qu'un modèle complet de pilote pourra être réalisé à l'aide d'un seul paradigme, car la nature très différente des tâches impliquées pour réaliser le système suggère que les paradigmes soient hétérogènes.

Un cadre de travail qui permet d'intégrer différents sous-systèmes pour accomplir une tâche globale serait un outil idéal pour réaliser un modèle de pilote complet. Dans ce contexte, les SMA sont une approche appropriée pour résoudre ce genre de problème, car ils peuvent être basés sur des paradigmes ou des architectures très différentes.

Les sections suivantes décrivent les tâches de pilotage. Pour les fins de la discussion, une division des tâches est faite selon une architecture à trois couches : organisation, coordination et exécution.

4.3.1 Couche organisation

La couche organisation, la couche la plus abstraite, reproduit le comportement humain et l'intelligence humaine. Elle s'occupe de tâches telles que la planification, la prise de décisions, l'apprentissage et le stockage de données à long terme. L'accomplissement de ce type

de tâches requiert un traitement de données de haut niveau, comme celui réalisé par les bases de connaissances. Les échelles de temps impliquées sont de l'ordre de quelques secondes à quelques heures. Par exemple, un pilote ne réévalue pas le parcours de la maison jusqu'au travail à chaque seconde, mais plutôt lorsque qu'il a du temps de libre et seulement si la situation le justifie. Par contre, une tâche d'organisation peut être commandée en haute priorité en des temps imprévus. Par exemple, le parcours doit être immédiatement réévalué face à une route bloquée par un embouteillage ou une avalanche.

Plus spécifiquement, un pilote tient compte de plusieurs facteurs pour établir une stratégie de pilotage. Par exemple, il doit tenir compte des conditions météorologiques, de son expérience de pilotage, de sa connaissance particulière du véhicule et de l'état du véhicule. Ces facteurs demandent au pilote de s'adapter à la situation et de réévaluer sa stratégie de pilotage en accord avec les conditions présentes. Aussi, un bon pilote évalue son rendement par rapport aux buts fixés et en retire une expérience afin d'optimiser ses actions pour une prochaine fois. Par exemple, s'il y a eu dérapage et perte de contrôle du véhicule, un bon pilote réévalue et ajuste son comportement au meilleur de ses connaissances afin que cette situation dangereuse ne se reproduise pas.

La couche organisation assure aussi la sécurité par une supervision générale du système. Par exemple, la vitesse peut être augmentée si des contraintes de temps doivent être respectées, mais elle ne doit pas l'être au prix de déstabiliser le véhicule ; la vitesse doit être réduite si les conditions routières sont douteuses et que les pneus sont usés.

Le pilote possède aussi des connaissances de base sur un type de véhicule, comme le nombre de pneus et le type de transmission (automatique ou manuelle). Les connaissances du pilote sur le comportement du véhicule s'ajustent avec le vieillissement de ce dernier afin de mieux le commander. La couche organisation intègre des connaissances abstraites générales de haut niveau sur le pilotage de véhicules et sur les véhicules eux-mêmes.

Les types de connaissances et les types de traitements de données impliqués dans la couche organisation sont de haut niveau. Les techniques d'intelligence artificielle sont souvent utilisées pour la réalisation des tâches de cette couche, car elles sont souvent bien adaptées à ce type de traitement.

Voici quelques exemples de tâches et de connaissances situées dans la couche organisation :

- déterminer un parcours
- planifier
- apprendre
- optimiser les actions
- prendre des décisions
- superviser la conduite
- s'adapter
- connaissances du véhicule
- expériences passées
- connaissances de pilotage

4.3.2 Couche coordination

La couche coordination est une couche intermédiaire servant d'interface entre la couche organisation et la couche exécution. Elle prend des décisions à court terme et fait un certain apprentissage, mais uniquement en utilisant une mémoire à court terme. Elle utilise généralement des connaissances de moins haut niveau que celles de la couche organisation.

Les tâches réalisées répondent à des besoins plus spécifiques que ceux de la couche organisation. La couche coordination évalue la situation en tenant compte des contraintes et des objectifs qu'elle a elle-même fixés et de ceux qui sont imposés par la couche organisation. Typiquement, elle sélectionne la prochaine action à exécuter, décide d'un tracé à suivre sur la route et de la vitesse désirée. Par exemple, le tracé du véhicule est différent dans une courbe, le pilote tente de minimiser l'accélération latérale en « coupant » la courbe. Elle envoie à la couche exécution les consignes jugées adéquates à la réalisation des objectifs. La couche coordination fournit les informations et les paramètres nécessaires à la couche exécution. Les échelles de temps impliquées sont de l'ordre d'une fraction de seconde à quelques secondes.

La couche coordination est essentielle pour déléguer les commandes à la couche exécution, car des conflits peuvent émerger d'informations contradictoires ou conflictuelles venant des différentes tâches qu'elle exécute et de la couche organisation. Ces tâches peuvent avoir des intérêts très différents. Par exemple, une contrainte de temps peut demander une vitesse accrue, mais la proximité des véhicules environnants, les conditions routières ou les limites mécaniques intrinsèques du véhicule peuvent le proscrire. La couche coordination doit résoudre les conflits entre les tâches en trouvant des compromis qui tiennent compte de la situation actuelle et de ses connaissances sur le pilotage. Elle fait donc une gestion locale du pilotage tout en coordonnant les différents intérêts à court terme du système complet.

En analysant son champ de vision, le pilote humain anticipe les courbes, et si nécessaire, ajuste la vitesse du véhicule. Il anticipe et décide du tracé à suivre. La couche coordination fait de même. Elle intègre des éléments de traitement et d'analyse d'images afin de pouvoir analyser le champ de vision du véhicule. Selon Smith et Starkey [138], les véhicules autonomes déjà réalisés ont utilisé plusieurs stratégies pour obtenir les informations nécessaires dont les caméras, les lasers, les radars, les sonars et les rayons infrarouges.

Le pilote réalise une autre tâche importante, soit la déduction de la position et de l'orientation du véhicule par rapport à la route. Selon Smith et Starkey [138], les systèmes les plus prometteurs utilisent le déplacement angulaire des roues (*dead reckoning*) jumelé à un système de GPS (*Global Positioning System*) ou de *map matching* pour déterminer la position et l'orientation du véhicule. Le *map matching* consiste à déterminer la position du véhicule en comparant la position estimée avec une base de données cartographiques à partir d'un point de repère qui a été identifié.

Les tâches de la couche coordination réalisées utilisent des méthodologies venant de plusieurs domaines. Quelques exemples de réalisations sont le stationnement en utilisant la logique floue par Sugeno et Murakami [141], l'évitement d'une collision en utilisant le comportement adaptatif ou les réseaux de neurones artificiels par Zapata et al. [160] et la sélection d'actions en utilisant le comportement adaptatif par Tyrell [146]. D'autres tâches, comme la résolution de conflits et la délégation de commandes, peuvent être résolues par des techniques d'intelligence artificielle, comme les bases de connaissances. Dans ce cas, la réactivité du système doit être sévèrement contrôlée à cause des échelles de temps impliquées dans cette couche.

Voici quelques exemples de tâches situées dans la couche coordination :

- évaluation de situation
- freinage
- accélération
- résolution de conflits
- stationnement
- orientation du véhicule
- position du véhicule
- démarrage et arrêt
- évitement de collision
- vitesse désirée
- anticipation
- tracé désiré
- analyse de la vision
- dépassement

4.3.3 Couche exécution

La couche exécution manipule physiquement les commandes du véhicule, c'est-à-dire, le volant, les freins et l'accélérateur. Les actions posées sont en accord, avec une marge d'erreur acceptable, avec les instructions fournies par la couche coordination. Typiquement, tout comme le pilote humain, elle représente un système à rétroaction où les actions sont nécessairement exécutées en temps réel. La couche exécution effectue la commande latérale (position) et le commande longitudinale (vitesse) du véhicule. Typiquement, elle s'occupe des tâches suivantes :

- amener le véhicule sur le tracé choisi, à la vitesse désirée et le maintenir dans cet état ;
- corriger de manière stable les erreurs par rapport à la position désirée ;
- maintenir le tracé et la vitesse en présence de turbulences.

La couche exécution s'occupe aussi d'exécuter des manoeuvres d'urgence, comme éviter une collision ou un obstacle. Ces manoeuvres sont hautement réactives et elles ne sont pas modélisées de la même manière que les manoeuvres normales, car lors de ces manoeuvres, le véhicule est souvent instable ou en dérapage.

Le développement de la couche exécution a fait l'objet d'intenses recherches, mais peu d'entre elles font une représentation explicite de l'intelligence du pilote. Les modèles classiques de la littérature utilisent les théories sur la commande linéaire et non-linéaire et s'attaquent surtout à la poursuite d'une trajectoire [7, 33, 36, 38, 74]. Ces modèles rendent plus difficile l'ajout de couches supérieures et n'expriment pas directement les comportements propres à la conduite d'un véhicule. Selon Ehsani et Tétreault [43], les modèles reproduisant des comportements d'un pilote sont typiquement classés dans trois catégories.

Modèles rétroaction-anticipation

Ce type de modélisation contrôle uniquement la composante latérale du déplacement du véhicule, la vitesse étant souvent présumée constante. Ces modèles stabilisent le véhicule sur la trajectoire[1] prédéfinie en compensant pour les erreurs résiduelles. Ces modèles, par

[1]Le terme trajectoire fait référence à un tracé avec un temps associé. Lorsque la vitesse est constante, on peut également fournir seulement le tracé.

exemple [39, 64, 105, 121], combinent une commande anticipative pour produire la commande majeure et une ou plusieurs boucles de rétroaction pour corriger l'erreur résiduelle. La figure 4.1 illustre un exemple d'architecture de ce type ; elle illustre le modèle de Donges [39].

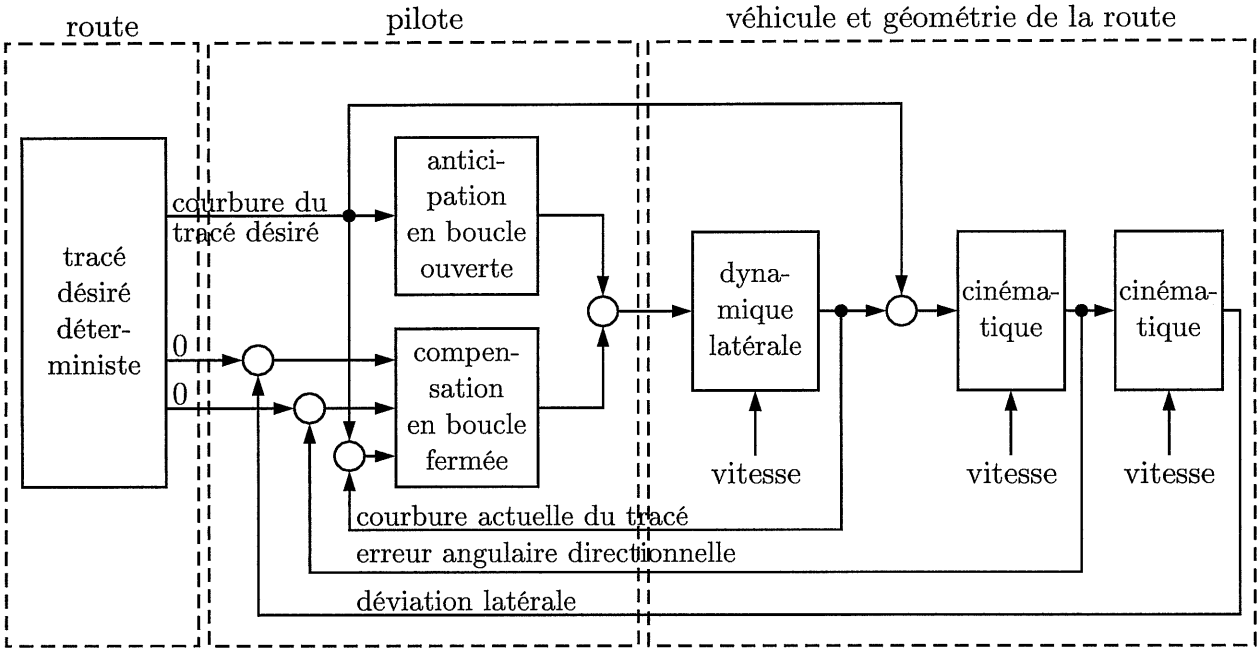


Figure 4.1 – Architecture du modèle de Donges à deux niveaux avec anticipation [39].

Le fonctionnement général du modèle est le suivant. D'un côté, le module compensation en boucle fermée corrige les erreurs (erreur angulaire directionnelle et déviation latérale) par rapport au tracé désiré (tracé désiré déterministe) et de la courbure actuelle de la route (courbure actuelle du tracé). D'un autre côté, le module anticipation en boucle ouverte ajoute une composante d'anticipation à la commande du véhicule, comme dans le cas d'un vrai pilote à l'arrivée d'une courbe. Le module dynamique latérale simule les comportements du véhicule et les deux modules cinématique simulent la géométrie de la route.

Modèles preview-predictor

Ces types de modèles produisent une commande basée sur la valeur présente et la valeur future de la référence, comme le trajet routier. Ils intègrent donc un élément de prédiction. De la même manière que les modèles rétroaction-anticipation, ils ne commandent que la composante latérale du déplacement du véhicule. Des exemples de modèles sont [54, 87, 93].

Modèles latéral-longitudinal

Contrairement aux modèles précédents, cette troisième catégorie sépare la commande latérale de la commande longitudinale du véhicule. Tel que suggéré par Sarkar et al. [132] et Ehsani et al. [44], la trajectoire est divisée en deux : le tracé géométrique et la vitesse désirée le long de ce tracé. La vitesse désirée est choisie pour s'assurer que l'accélération latérale ne dépasse pas certaines limites. En dissociant la vitesse du tracé, cette approche facilite l'ajout de couches supérieures, comme les couches coordination et organisation, en reproduisant une stratégie de commande qui s'apparente mieux à celle des pilotes humains. Un modèle de ce type est celui de Ehsani et al. [44], il est illustré à la figure 4.2.

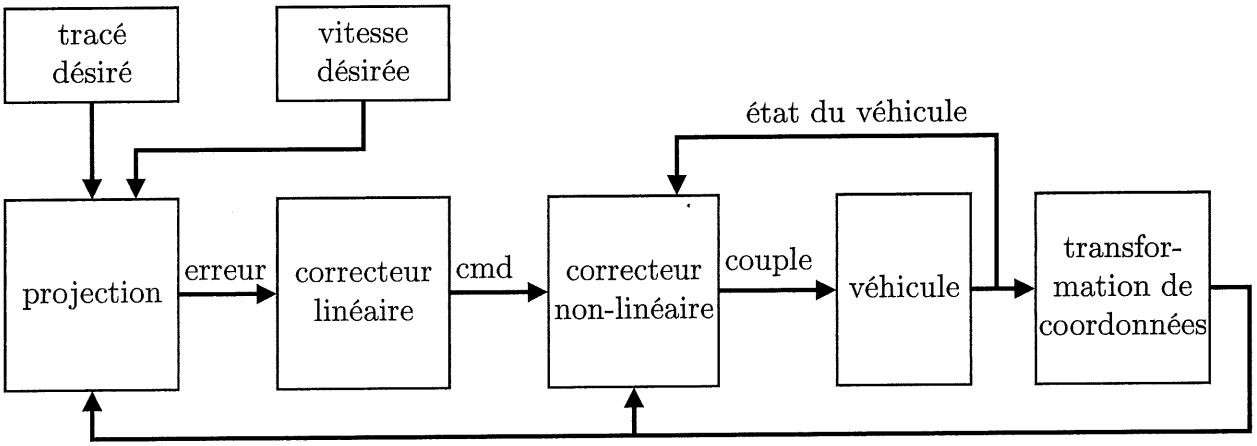


Figure 4.2 – Architecture du modèle de Ehsani et al. [44].

Le fonctionnement général du modèle de Ehsani et al. est le suivant. Le module projection calcule les erreurs par rapport au tracé désiré, à la vitesse désirée et à l'état du véhicule (position, vitesse, orientation, etc.). Le module correcteur linéaire calcule la commande qui doit être appliquée. Le module correcteur non-linéaire calcule alors les couples qui doivent être appliqués sur les roues et le volant. Le module véhicule simule la dynamique du véhicule.

D'autres modèles utilisant des paradigmes issus de recherches en intelligence artificielle ont également été proposés. Par exemple, certains modèles utilisent la logique floue [79, 86] ou encore les réseaux de neurones artificiels [81, 83, 90].

Toutes les tâches de pilotage, qu'elles soient au niveau organisation, coordination ou exécution, peuvent être présentes dans un modèle complet de pilote. Ce dernier doit donc pouvoir

traiter avec les différentes échelles de temps et les différentes représentations nécessaires à la réalisation des tâches à tous les niveaux. Dans ce contexte, une approche multi-agents est bien adaptée.

4.4 Considérations de modélisation d'un pilote

La modélisation d'un système informatique qui comporte des éléments matériels, comme les véhicules autonomes, demande aux concepteurs de considérer certaines limitations qu'un système purement informatique n'a pas besoin de traiter.

Hexmoor et al. [66] exposent bien cette problématique. Les concepteurs d'un système de contrôle d'un véhicule autonome ou d'un robot doivent faire des choix face aux limitations intrinsèques du matériel. Ces choix transcendent généralement le matériel spécifique en question et les solutions peuvent être portables. Typiquement, ces problèmes sont liés à l'architecture du système. Par exemple, lorsque vient le temps de porter une solution d'un système à un autre, les concepteurs ressentent le besoin de refaire une conception complète du système pour s'adapter au nouvel environnement matériel et logiciel. De la même manière, pour intégrer un nouvel élément dans le système, les concepteurs ont tendance à refaire une partie significative du système. Cette attitude de reconception de système est très coûteuse. Ajouter simplement un module à un système extensible serait beaucoup plus efficace et l'architecture y joue un rôle important.

L'architecture d'un système reflète les principes organisateurs de ses concepteurs. Ces principes ont été appris aux travers d'expériences passées de construction de systèmes. Albus [3] mentionne qu'une architecture est une description de la construction d'un système à partir de ses composants de base et de la manière dont ils forment un tout. Il ne suffit pas de définir un modèle de l'intelligence du système, il faut définir le système au complet. Dans ce sens, plusieurs questions importantes doivent être considérées pour la conception d'un système ayant une composante physique, tels les véhicules autonomes.

Selon Hexmoor et al. [66], les points suivants sont à considérer lors du développement ou du choix d'une architecture de commande d'un véhicule.

Représentation : L'information dans les architectures peut être représentée de deux manières différentes. La première est unificatrice, comme dans le cas des agents délibératifs. La deuxième est un mélange de représentations hétérogènes, comme dans le cas des agents hybrides, ou sans aucune représentation, comme dans le cas des agents réactifs. Des représentations hétérogènes sont utiles pour la modélisation. Par contre, la manière dont elles sont reliées d'un niveau à l'autre et comment coordonner efficacement le savoir-faire qui est distribué dans les niveaux de l'architecture sont deux problèmes qui restent sans réponse acceptable.

Contrôle et coordination : Le contrôle peut être central ou distribué. Le contrôle distribué requiert des mécanismes d'arbitrage afin de coordonner les processus qui sont en compétition pour les ressources matérielles. Plusieurs architectures ont été proposées pour solutionner ce problème dont les architectures réactives.

Apprentissage : L'apprentissage consiste à changer la structure interne afin d'améliorer les performances à long terme. Idéalement, une architecture est organisée afin de faciliter l'apprentissage. Par exemple, pour une architecture à trois couches, la couche la plus basse améliore les performances d'actions purement réflexives, la couche la plus haute améliore les performances cognitives et la couche du milieu améliore les performances de coordination entre les deux autres couches.

Toujours selon Hexmoor et al., ce type d'apprentissage horizontal à l'intérieur d'une même couche contraste avec l'apprentissage vertical entre les couches. En faisant migrer des comportements planifiés de la couche la plus haute vers les couches inférieures, ces comportements deviennent de plus en plus réactif. Une tâche qui était hautement cognitive devient purement réactive avec les améliorations de performance qui en découlent. Aussi, à l'aide de l'apprentissage, des patrons d'interactions répétitifs à un bas niveau, comme une habileté motrice, peuvent être redéfinis en utilisant des concepts cognitifs en des moments opportuns pour ainsi bénéficier des généralisations qui en découlent. Les mécanismes de migration d'habiletés entre les couches dans l'apprentissage vertical sont encore inconnus. Par contre, il semble évident que les couches intermédiaires de l'architecture joueront un rôle important.

Réactivité : À mesure que les systèmes informatiques sont appliqués à des domaines temps réel de plus en plus complexes, ils sont soumis à des contraintes de temps de plus en plus strictes. La plupart des architectures ne garantissent pas une réponse dans des conditions intensives de temps réel où le manque de respect des contraintes de temps signifie un échec. Les architectures se contentent généralement d'exécuter les tâches le plus rapidement possible en espérant que les choses se passent bien. Il y a cependant des exceptions. Certaines architectures comme CIRCA [111] peuvent garantir une réponse en un temps donné. Ces architectures peuvent raisonner sur les comportements en temps réel et ainsi concevoir des plans temps réel qui garantissent les résultats dans un temps donné.

Selon Musliner et al. [112], la problématique de garantir un comportement en temps réel à l'intérieur d'un système intelligent est loin d'être résolue. En fait, cette question a poussé les chercheurs en intelligence artificielle à créer un nouveau champ de recherche, l'intelligence artificielle en temps réel, qui s'attaque directement à cette question.

Inspiration biologique et psychologique : La biologie et la psychologie peuvent être des sources d'inspiration importantes. Les systèmes biologiques sont les seuls exemples connus de systèmes intelligents fonctionnels. Lorsqu'un système est conçu, les décisions prises sur certains aspects de base peuvent être issus d'expériences passées avec les humains ou les animaux. Le choix des capteurs et leur conception ainsi que les techniques de contrôle peuvent être d'inspirations issues de la biologie ou de la psychologie. Ces sources d'inspiration sont un couteau à double tranchant. Par exemple, lorsque vient le temps de concevoir un bras de robot, la plupart des gens ont des pensées anthropomorphiques. Le bras du robot ressemble à un bras humain ou à une patte d'araignée. Souvent, aucune autre alternative de conception n'est vraiment considérée alors que dans certains cas une conception différente et hétérodoxe pourrait être beaucoup plus avantageuse.

Évaluation : Évaluer et comparer les performances d'architectures différentes est une tâche difficile. Certains diront impossible, car malheureusement les expériences réelles ainsi que leurs environnements sont difficilement reproductibles.

Pourtant, il semble que l'évaluation et la comparaison d'architectures doivent être faites afin de déterminer les meilleurs choix d'architecture ou de conception. Des simulations et des évaluations permettent quand même de guider les stages préliminaires de conception du système.

Toutes ces considérations doivent être prises en compte pour la modélisation complète d'un pilote ou la création d'un nouveau modèle. Brady et Hu [16] identifient les aspects importants pour toute théorie de l'intelligence et donc des systèmes robotiques intelligents :

- réactivité et planification ;
- processus distribués ;
- traitement de l'incertitude ;
- comportement ayant un but ou une raison ;
- émergence de propriétés des processus interagissant entre eux.

Ces aspects ressemblent beaucoup à ceux de Hexmoor et al. [66]. Par contre, le traitement de l'incertitude et l'émergence des comportements sont ajoutés à la liste. Le traitement de l'incertitude est un point important relevé par un bon nombre de chercheurs. L'incertitude survient dans des situations imprévues ou inconnues ou encore lorsqu'il y a des connaissances incomplètes ou contradictoires. L'émergence de propriétés de processus qui interagissent représente aussi une caractéristique souhaitée des systèmes intelligents et des systèmes multi-agents. Il est souhaitable que le tout soit plus grand que la somme des parties, que des comportements naissent des interactions entre les différents processus impliqués.

Peu importe la modélisation choisie, le pilote doit réaliser toutes les tâches nécessaires pour accomplir le travail qui lui est assigné. Les capacités du pilote servent à contrôler le véhicule pour satisfaire les buts qui ont été fixés. Plusieurs architectures ont été proposées afin de modéliser un pilote. Les plus importantes sont abordées dans les sections suivantes.

4.5 Architecture réactive

Une architecture de contrôle pour robots mobiles a été proposée par Brooks [17] et elle est illustrée à la figure 4.3. Comme mentionné à la section 2.4.2, elle est aussi appelée *subsumption*

architecture. L'approche consiste en la décomposition d'un problème en plusieurs **niveaux de compétences** (*levels of competence*). Chaque niveau est en compétition avec les autres pour le contrôle des sorties du système.

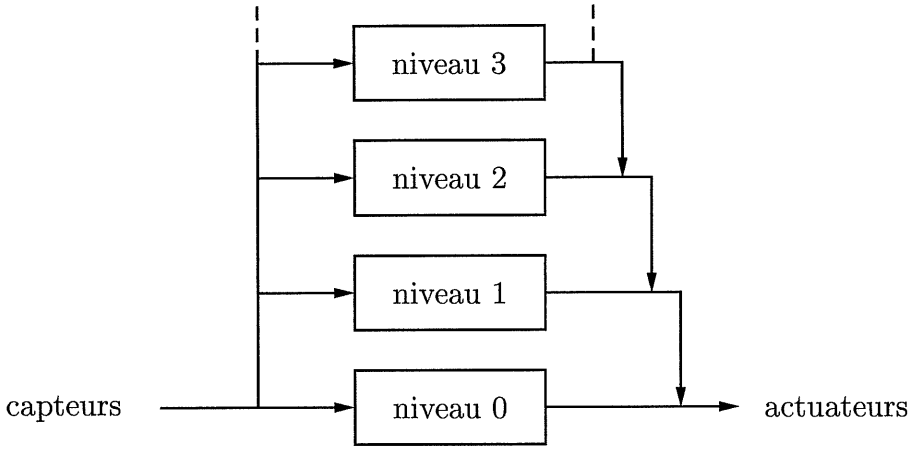


Figure 4.3 – Architecture à niveaux de compétences [17].

Un niveau de compétence correspond à une classe de comportements désirés pour le robot pour tous les environnements qu'il peut rencontrer. Chaque niveau de compétence inclut les niveaux de compétences inférieurs comme sous-ensemble. Un niveau de compétence peut être perçu comme imposant des contraintes additionnelles sur le niveau qui lui est inférieur. Un niveau de compétence plus élevé définit une classe de comportements plus spécifiques.

L'idée principale derrière les niveaux de compétences est de bâtir un système de contrôle par étage simplement en ajoutant un niveau de compétence supérieur à ceux existants. Normalement, cette architecture assure une bonne modularité du système.

Plus spécifiquement, à chaque niveau de compétence correspond un système de contrôle ou une **couche de contrôle** (*layer of control*). Une couche de contrôle de niveau N peut examiner les données de la couche de contrôle de niveau $N-1$ et lui injecter des données. Elle peut ainsi altérer le flux normal de données de la couche de niveau $N-1$ qui ne sait pas que la couche qui lui est supérieure interfère avec ses flux de données. La couche N atteint un niveau N de compétence, car elle inclut elle-même et les couches de contrôle qui lui sont inférieures.

Cette architecture a rapidement produit des résultats extrêmement intéressants, mais elle a aussi des désavantages. Werner [151] résume bien les problèmes majeurs de cette architecture qui se situent au niveau du contrôle. Une couche de contrôle ne fait pas seulement que supprimer les sorties de la couche qui lui est inférieure, elle les remplace. Les conflits entre les deux couches sont inexistants, car la stratégie de la couche supérieure l'emporte à chaque fois. S'il existe uniquement cette méthode de résolution de conflits, les stratégies inférieures sont toujours dominées. Les couches inférieures sont alors totalement inutiles et peuvent être supprimées du système.

Cette situation va complètement à l'encontre de l'idée de base de cette architecture. Les stratégies de bas niveau réagissent à des changements locaux, comme des obstacles. Dans de tels cas, les stratégies locales des niveaux inférieurs doivent dominer temporairement celles des couches supérieures. De la même manière, les stratégies locales ne peuvent pas toujours dominer les stratégies de plus haut niveau, car les couches supérieures deviennent inutiles. Le contrôle ne peut pas uniquement passer du plus haut niveau vers le plus bas ou l'inverse. Des mécanismes de médiation doivent exister afin d'équilibrer le choix des stratégies.

Face à ce problème, plusieurs travaux de recherche modifient ou ajoutent à l'architecture réactive. Hügli et al. [71] ajoutent un niveau cognitif basé sur un tableau noir. Martinengo et al. [100] définissent une architecture à trois niveaux : temps réel, moyen terme et long terme. Hu et al. [69] proposent une architecture hiérarchique hybride intégrant des éléments réactifs et délibératifs. Rochwerger et al. [126] ajoutent à l'architecture réactive un tableau noir et des *scripts*.

4.6 Théorie des machines intelligentes

La théorie des machines intelligentes (*Theory of Intelligent Machines* ou TIM) est une architecture proposée par Saridis [129] ; elle est illustrée à la figure 4.4. Cette architecture a été utilisée par Saridis [130] pour intégrer et gérer la prise de décision et l'intelligence dans des systèmes de contrôle. Plus particulièrement, elle a été appliquée au contrôle de véhicules par Lefebvre et Saridis [88].

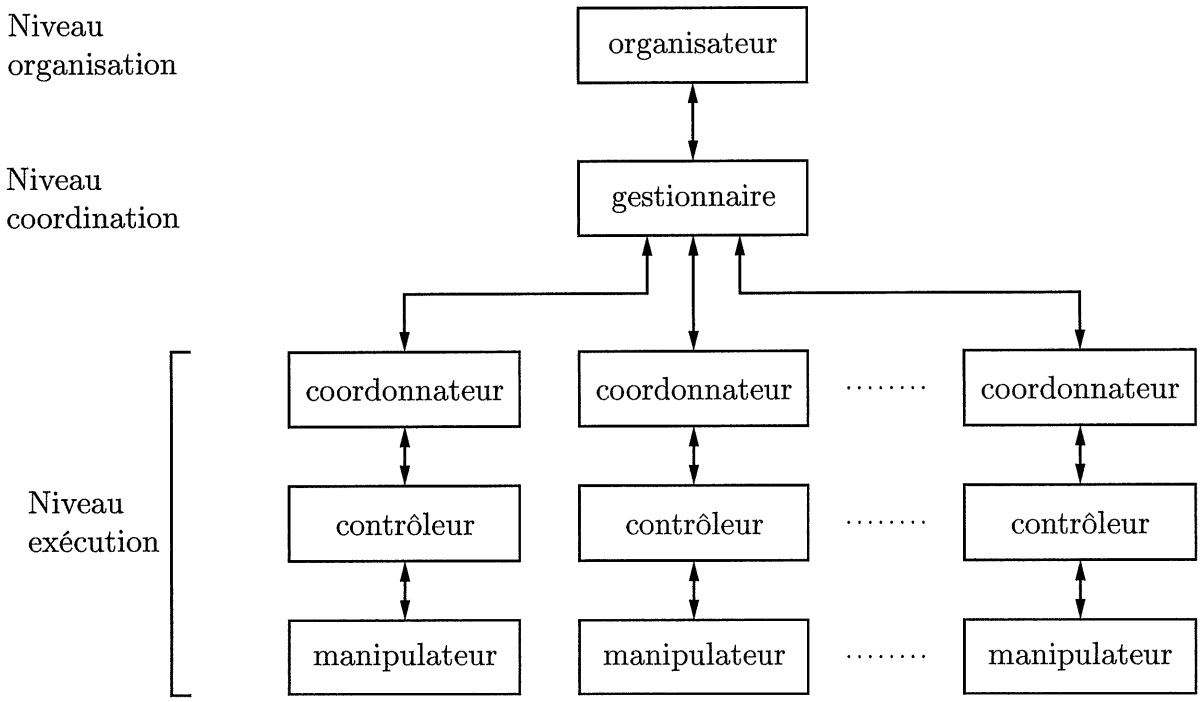


Figure 4.4 – Modèle logique d'une machine intelligente [88].

L'architecture TIM modélise un système par une architecture hiérarchique à trois niveaux : organisation, coordination et exécution, comme illustré à la figure 4.4. Une description de ces trois niveaux est donnée à la section 4.3. Saridis [131] explique que cette architecture hiérarchise les données et les fonctions en accord avec le principe de **précision grandissante avec intelligence décroissante**. Ce principe implique qu'au bas de la hiérarchie, la précision et la certitude sont grandes, mais que l'intelligence est petite, et qu'au sommet, la précision est petite ou inexistante, mais que l'intelligence est grande. Par exemple, au sommet, des raisonnements sur des concepts abstraits et généraux sont faits et au bas de la hiérarchie des séquences précises de commandes et d'actions sont exécutées sans aucune planification.

Lefebvre et Saridis [88] utilisent le modèle hiérarchique à trois niveaux de la TIM pour le contrôle de véhicule. Le niveau organisation représente les capacités de raisonnement et la planification de haut niveau. Ce niveau utilise des techniques souvent issues de l'intelligence artificielle. Le niveau coordination intègre les capacités des différentes composantes du système et le niveau exécution contrôle les circuits de commandes.

Cette architecture à trois niveaux est retrouvée dans plusieurs modèles de véhicules. McRuer et al. [105] décrivent, sans le réaliser, un modèle à trois niveaux : navigation, guidage et commande. Le niveau navigation est la sélection générale d'un parcours ; il engendre une série d'opérations qui affectent les niveaux guidage et commande. Le niveau guidage est concerné par les aspects de sélection, de décision et de définition d'un tracé pour accomplir une tâche particulière. Par exemple, si la tâche est d'effectuer un dépassement, le guidage inclut la décision du moment pour dépasser, la sélection du tracé en se basant sur la position respective des autres véhicules et des contraintes imposées par la route. Le niveau commande est l'exécution physique du guidage en manipulant le volant, l'accélérateur et les freins du véhicule en respectant le plan établi par le niveau guidage.

Une correspondance existe entre les niveaux de la TIM et de ce modèle : navigation et organisation, guidage et coordination, contrôle et exécution. Cette correspondance existe aussi avec d'autres modèles, comme ceux exposés dans [43, 71, 100].

Ce type d'architecture à trois niveaux est très courant dans la littérature, et cela dans plusieurs domaines. Elle est retrouvée dans des systèmes de contrôle jusque dans la structure interne d'un agent logiciel intelligent. Le concept de niveau ou de couche est naturel dans plusieurs domaines où la modélisation de systèmes complexes est nécessaire.

L'architecture TIM et ses semblables intègrent des éléments délibératifs et des éléments temps réel qui sont réactifs. Elle semble pousser dans la même direction que les architectures hybrides d'agents qui tentent d'ajouter aux architectures réactives pour obtenir des comportements délibératifs de leurs systèmes.

4.7 Système de contrôle en temps réel

L'architecture RCS (*Real-time Control System*) proposée par Albus [2, 3, 4] intègre des éléments réactifs et délibératifs. Un noeud de cette architecture est illustré à la figure 4.5. L'architecture RCS est présentée comme un modèle de référence pour la conception de systèmes intelligents. Cette architecture est composée d'une hiérarchie de couches. À chaque couche, l'échelle de temps varie d'un ordre de grandeur. Ainsi, le sommet de la hiérarchie

traite des éléments à long terme et le bas de la hiérarchie traite des éléments à court terme ou de nature temps réel.

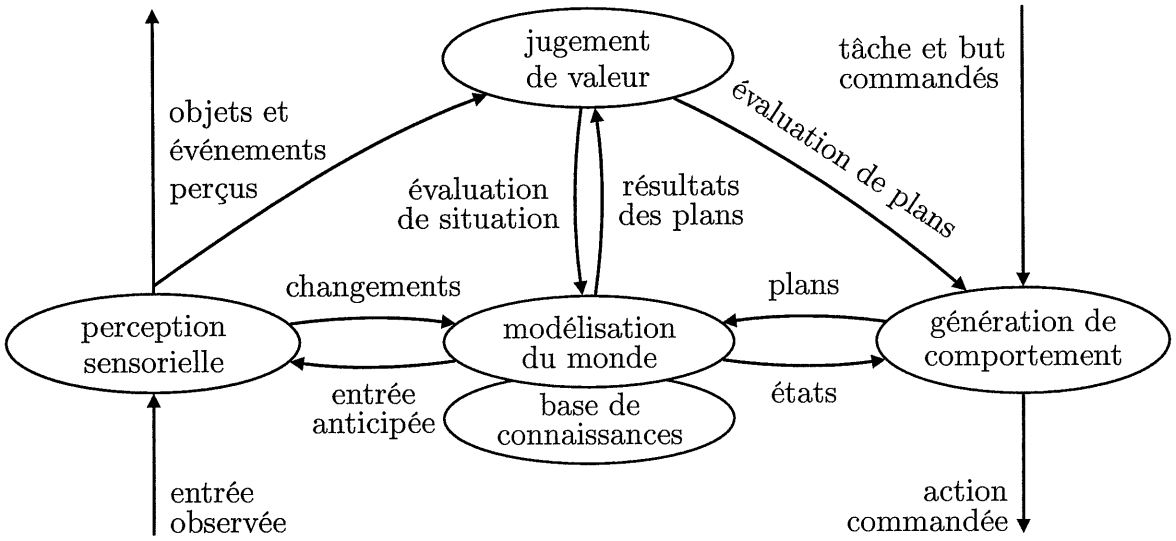


Figure 4.5 – Noeud typique de l'architecture RCS [4].

Chaque noeud de l'architecture RCS est composé de quatre modules de traitement. Les modules de traitement sont les suivants : génération de comportement (*Behavior Generation* ou BG), modélisation du monde (*World Modelling* ou WM), perception sensorielle (*Sensory Processing* ou SP), jugement de valeur (*Value Judgment* ou VJ) et base de connaissances (*Knowledge Database* ou KD). Voici une description de ces modules.

Module BG

Le module BG (*Behavior Generation*) prend les commandes qui arrivent du module BG du niveau supérieur et les décompose en sous-commandes. Un plan pour mener à bien le travail est construit afin d'assurer une bonne coordination des sous-commandes. Les sorties générées par chacune des sous-commandes deviennent des commandes pour un module BG au niveau inférieur dans la hiérarchie.

Les modules BG peuvent vérifier les résultats de leurs actions par la boucle de rétroaction composée des modules BG, actions, entrées, SP et WM. Ils peuvent ainsi ajuster les commandes générées et soumises au niveau inférieur pour en arriver à leurs fins.

Module WM

Le module WM (*World Modelling*) a quatre fonctions principales :

1. Faire la gestion de la base de données de connaissances par le module KD ; celui-ci assure son intégrité et sa mise à jour.
2. Générer des prédictions sur les perceptions futures afin de permettre au module SP de faire des corrélations et du filtrage prédictif.
3. Répondre aux requêtes du module BG concernant l'état courant de l'environnement.
4. Faire les simulations nécessaires pour satisfaire les exigences de planification du module BG. Ceci requiert des modèles dynamiques permettant de prédire les résultats d'actions futures. Les résultats prédits sont soumis au module VJ pour évaluation et sont ensuite soumis au module BG pour ses besoins en planification.

Module SP

Le module SP (*Sensory Processing*) traite les données sensorielles, visuelles, auditives, tactiles ou de tout autre type. Ce module contient des algorithmes pour traiter ces types de données, comme des filtres, des masques, de la reconnaissance de formes ou tout autre algorithme pertinent pour les traitements à faire.

Par exemple, pour la vision, le module SP peut utiliser différentes techniques pour analyser une scène ou calculer les attributs de certains objets. Il peut ainsi fournir les informations nécessaires pour différentes activités comme le déplacement, la manipulation ou la communication.

Module VJ

Le module VJ (*Value Judgment*) contient des algorithmes pour calculer les coûts, les risques et les bénéfices. Il peut évaluer les états et les situations pour analyser la fiabilité des estimations des états et pour assigner des valeurs coûts-bénéfices à des objets ou des événements.

Par exemple, il peut calculer si un objet ou un événement vaut la peine d'être considéré ou encore s'il vaut la peine d'être stocké dans la mémoire à long terme. Il peut aussi évaluer le niveau de confiance entre une prédiction et des faits.

Module KD

Le module KD (*Knowledge Database*) s'occupe du stockage de données de haut et de bas niveaux afin de supporter les autres modules dans leurs tâches respectives. Il consiste essentiellement en des structures de données qui contiennent des variables d'état. Les informations contenues dans ce module incluent les entités et les événements connus et de quelle manière l'environnement réagit statiquement et dynamiquement. Le module KD contient de la mémoire à court terme et à long terme.

Une couche de l'architecture RCS est donc composée d'éléments réactifs et délibératifs et permet à un système de bénéficier du meilleur des deux mondes. Le résultat est un système qui combine et distribue des éléments délibératifs et réactifs au travers de toute la hiérarchie. Ainsi, des comportements planifiés et réactifs sont intégrés à tous les niveaux et à toutes les échelles de temps.

Chaque couche est composée de noeuds interconnectés par un réseau de communication. Les noeuds connectés forment typiquement un graphe. Les modules de chaque noeud sont connectés à leurs semblables de même niveau, à celui du niveau supérieur et à ceux du niveau inférieur. Les modules ainsi reliés forment un arbre de commandes.

L'architecture RCS avec ces différents modules a l'avantage d'intégrer des composantes réactives et délibératives à tous les niveaux. Bien qu'elle soit une des plus complète et bien qu'elle ait été utilisée pour plusieurs types d'applications industrielles, elle est très complexe et peut devenir très lourde pour de petites applications qui ne nécessitent pas toute la complexité de cette architecture. Dans sa version simplifiée, où tous les modules ne sont pas présents à chaque noeud, il est possible de retrouver une architecture très semblable à l'architecture réactive de Brooks ou la TMI de Saridis.

4.8 Discussion

La modélisation complète d'un pilote pour le contrôle d'un véhicule représente un défi autant du point de vue matériel que logiciel. Afin de résoudre les problèmes liés à cette problématique, les chercheurs ont proposé plusieurs architectures dont les trois plus courantes sont

la *subsumption architecture* de Brooks, la TMI de Saridis et le RCS d'Albus. Des principes de certaines de ces architectures sont récupérés afin de proposer un nouveau modèle de pilote.

Afin de réaliser un véhicule autonome, un système doit intégrer de manière cohérente toutes les solutions aux sous-problèmes. Il semble évident qu'un modèle complet de pilote doit être réalisé à l'aide d'un environnement de développement permettant l'utilisation de paradigmes hétérogènes, car les domaines impliqués pour la réalisation des sous-problèmes viennent de techniques et de paradigmes très différents.

L'architecture du système de contrôle est un des problèmes importants. La plupart des architectures sont conçues à partir de niveaux ou de hiérarchies de niveaux. Ce type d'architecture semble être très répandu et une approche qui fait l'unanimité auprès des chercheurs. Ceci est compréhensible, car pour la majorité des gens, la modélisation mentale du pilotage d'un véhicule est constituée d'un ensemble de tâches distinctes.

Selon Hu et al. [69], les concepteurs de robots mobiles et autres véhicules autonomes utilisent des modèles hiérarchiques ou encore des modèles avec plusieurs niveaux pour des raisons d'implémentation. Un véhicule autonome réalise des tâches complexes, comme le déplacement dans son environnement, le traitement de la vision, la planification d'actions et l'adaptation du comportement aux changements de son environnement. Plusieurs de ces tâches complexes doivent s'exécuter en temps réel et sont très exigeantes en puissance de calcul, d'où la nécessité du traitement en parallèle et de systèmes multiprocesseurs. Les architectures doivent inclure toutes ces tâches simultanément et en faire la gestion. Dans ce contexte, les modélisations hiérarchiques ou à niveaux semblent naturelles et offrent plusieurs avantages :

- modélisation distribuée, donc plus facilement réalisée en parallèle ;
- modularité qui facilite la maintenance et l'extension du système ;
- facilité de traitement des différentes échelles de temps impliquées dans les tâches : le temps réel, le moyen terme et le long terme ;
- facilité de gestion des redondances ; elles sont souvent utilisées pour augmenter la fiabilité du système ;
- activation simultanée de plusieurs tâches ou de comportements différents.

Les architectures hiérarchiques semblent être bien adaptées à la problématique des véhicules autonomes. Par contre, l'intégration de composantes réactives et délibératives pose encore des problèmes de coordination, tout comme avec les agents hybrides. Par exemple, les couches réactives oeuvrent en temps réel, contrairement aux couches délibératives.

Aussi, beaucoup d'activités qui pourraient être réalisées par les véhicules autonomes restent encore à résoudre ou n'ont pas encore de solution acceptable. Un exemple est la collaboration de plusieurs véhicules pour accomplir une tâche globale ou la conduite dans une circulation dense.

Les architectures pour le contrôle de véhicules doivent être facilement extensibles et portables pour permettre une intégration des solutions aux problèmes restant à résoudre et à l'adaptation aux nouvelles architectures matérielles. Sans ces caractéristiques d'extensibilité et de portabilité, les concepteurs doivent recréer le système à chaque fois, ce qui est très coûteux et indésirable.

Dans le contexte des travaux de recherche présentés, les problématiques liées à l'architecture d'un modèle complet de pilote qui soit extensible, portable et supportant l'hétérogénéité des tâches sont abordées. Le chapitre 7 permet d'analyser et de solutionner ces problématiques.

DEUXIÈME PARTIE

ENVIRONNEMENT GEMAS

CHAPITRE 5

GEMAS : UN ENVIRONNEMENT GÉNÉRIQUE DE DÉVELOPPEMENT DE SYSTÈMES MULTI-AGENTS

Les discussions des chapitres 2 et 3 montrent que des questions centrales aux SMA sont encore sans réponse acceptable et que plusieurs problèmes importants restent encore à résoudre. Ce chapitre aborde certains des ces problèmes et propose des solutions.

5.1 Introduction

La littérature sur les SMA souligne à plusieurs reprises les défis les plus importants à relever et trois d'entre eux se distinguent :

Développer des outils de développement de SMA : Il faut développer et rendre disponibles de bons outils réutilisables et génériques de développement de SMA. Dans ce contexte, le terme réutilisable est utilisé pour indiquer l'indépendance des outils face à la plate-forme et du contexte d'utilisation. Le terme générique est utilisé pour indiquer l'indépendance des outils face à l'application et du type de SMA. En étant indépendants de l'application, les outils sont plus facilement réutilisables et leur durée de vie est allongée. Un environnement possédant ces caractéristiques permet de mieux évaluer les méthodologies de conception proposées et il est alors possible de les modifier afin de mieux refléter la réalité des SMA. Aussi, des bancs d'essais pourraient être plus facilement proposés et standardisés.

Supporter l'hétérogénéité : Les outils de développement doivent permettre l'intégration d'agents hétérogènes afin de préserver les investissements passés et optimiser les choix d'implémentation des agents en sélectionnant les meilleures approches.

Rapprocher la théorie de la pratique : La divergence entre ces deux approches a été mise en évidence au chapitre 2. D'un côté, les théories ont peu d'utilité pratique, car elles ne donnent pas d'indications précises sur la conception d'agents et de SMA. D'un autre côté, les approches pratiques manquent de degrés d'abstractions. Comme mentionné à la section 3.9, la divergence entre ces deux approches a de fâcheuses conséquences.

Ces problèmes qui n'ont pas encore été résolus permettent de dégager le développement d'outils réutilisables et génériques comme point commun central. Avec de tels outils, la réalisation de SMA serait plus facile et le nombre de systèmes développés augmenterait. Cela aurait comme conséquence que des études, des vérifications et des améliorations sur l'intégration d'agents hétérogènes pourraient être réalisées. De la même manière, la divergence entre la théorie et la pratique pourrait diminuer avec l'aide des études et des SMA réalisés, car ces derniers permettraient d'analyser la validité des théories, d'évaluer la valeur des approches pratiques et du rapprochement possible entre les deux. L'hétérogénéité des agents et le rapprochement de la théorie et de la pratique passent par la réalisation d'outils de développement bien adaptés à la réalisation de SMA. L'existence d'outils réutilisables et génériques serait une contribution importante au domaine des SMA.

L'absence d'outils génériques et réutilisables de développement de SMA, donc l'absence d'une base de développement de SMA, a plusieurs conséquences négatives importantes :

Division des efforts de recherche : Les groupes de recherche développent chacun de leur côté leurs propres solutions pour de nouvelles théories ou de nouvelles approches. Malheureusement, les solutions et les outils de développement ne sont généralement pas transférables d'un environnement de développement à l'autre ou ils le sont difficilement.

Difficulté de cumuler les réalisations : Les chercheurs ont de la difficulté à cumuler dans un ensemble réutilisable le fruit des développements réalisés par les différentes

équipes de recherche. Ceci ajoute aux difficultés de pousser encore plus loin les théories et les réalisations. De plus, cela empêche un effet d'entraînement qui permettrait aux SMA d'évoluer beaucoup plus rapidement.

Retardement de la mise en marché d'applications : Il est difficile pour les concepteurs de développer des applications basées sur les SMA selon leur conception et leur modélisation. Ils doivent généralement concevoir à partir de zéro tout le système, car il n'existe pas d'environnements génériques, réutilisables et portables de développement de SMA qui sont disponibles.

La réalisation d'outils génériques et réutilisables de développement de SMA ferait avancer la recherche des SMA d'un grand pas. Cependant, deux problèmes importants doivent être résolus en chemin vers la réalisation de tels outils :

1. Identifier les caractéristiques des environnements génériques pour le développement de SMA. Il faut identifier et définir un ensemble de caractéristiques et de mécanismes de base pour ce type d'environnement de développement. Il faut aussi assurer que cet ensemble est indépendant de la particularité des agents et de l'application.
2. Assurer que l'environnement de développement soit un bon outil de développement en faisant les choix de conception nécessaires pour favoriser la simplicité d'utilisation, la portabilité, l'extensibilité et la maintenance.

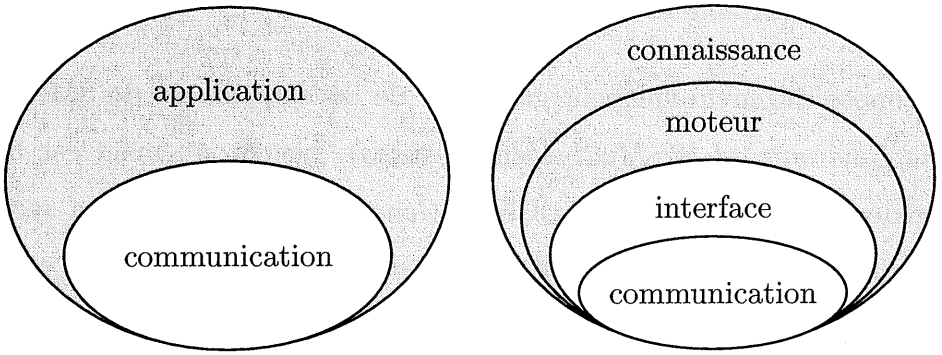
Ce chapitre propose un environnement générique de développement de SMA nommé GEMAS (*Generic Environment for Multi-Agent Systems*). L'environnement est basé sur une architecture générique d'agent nommée GAM (*Generic Agent Model*). La section 5.2 expose l'architecture du modèle GAM. C'est elle qui permet le développement et l'intégration d'agents hétérogènes et donc de réaliser des SMA hétérogènes. La section 5.3 identifie les fonctionnalités de base nécessaires pour qu'un agent puisse communiquer. Les sections 5.4 et 5.5 montrent comment développer des agents délibératifs, réactifs et hybrides à partir du modèle GAM. Les considérations de développement de SMA hétérogènes basés sur le modèle GAM sont discutées à la section 5.6. Le débogage de SMA et comment il est intégré dans l'environnement GEMAS est expliqué à la section 5.7. La section 5.8 permet de conclure sur les concepts présentés dans ce chapitre. L'implémentation des concepts de ce chapitre, le modèle GAM, les agents qui en sont dérivés et l'environnement de développement sont exposés au chapitre 6.

5.2 Modèle générique d'agent

Un modèle générique d'agent doit être indépendant du paradigme, de l'application et du système d'opération. Sans le respect de cette caractéristique, le modèle ne peut satisfaire à toutes les exigences de la diversité des applications. De nouveaux paradigmes et de nouveaux systèmes d'opération seront inventés et ils doivent pouvoir être intégrés dans le modèle générique d'agent. Cette section présente un modèle d'agent générique nommé GAM (*Generic Agent Model*).

5.2.1 Architecture

Les caractéristiques générales d'un agent fournissent des pistes quant aux différentes parties composant un agent générique. La partie application et la partie communication peuvent initialement être mises en évidence. La partie application exécute ses tâches selon les désirs des concepteurs et elle manipule les fonctionnalités de la partie communication afin d'interagir adéquatement avec le monde externe. La figure 5.1(a) illustre ces deux parties.



(a) Parties application et communication d'un agent.

(b) Modèle générique d'agent à quatre niveaux : le modèle GAM.

Figure 5.1 – D'un agent vers le modèle GAM.

La partie communication peut être décomposée logiquement en deux parties : l'interface des fonctionnalités de communication et les fonctionnalités de communication proprement dites. Dans la terminologie GAM, ces deux parties sont appelées le niveau communication et le niveau interface ; ce sont les niveaux inférieurs.

La partie application est réalisée en utilisant un paradigme particulier. Il est utilisé et exploité d'une manière spécifique selon les besoins de l'application et les choix faits par les concepteurs. Il est possible de distinguer logiquement le paradigme choisi de son exploitation spécifique. Dans la terminologie GAM, ces deux parties sont appelées respectivement le niveau moteur et le niveau connaissance ; ce sont les niveaux supérieurs.

Ainsi, l'architecture du modèle GAM est composée des niveaux communication, interface, moteur et connaissance ; ils sont illustrés à la figure 5.1(b). Chaque niveau peut être perçu comme encapsulant le précédant et joue un rôle très précis. Les sections suivantes donnent une description de chacun des niveaux.

5.2.2 Niveau communication

Ce niveau offre les mécanismes de base pour l'établissement de liens de communication et l'échange de messages avec le monde externe. Il n'a aucune connaissance du contenu des messages et de leur interprétation ; il en assure seulement le transfert. Seul le niveau interface interagit directement avec ce niveau. Deux types de communication peuvent être distingués :

Communication évoluée : Cette catégorie de fonctionnalités permet d'établir, de terminer, d'accepter et de valider une communication avec d'autres agents. Elle permet aussi d'échanger des données et donc de recevoir et d'envoyer des messages à un agent en particulier ou à un ensemble d'agents. Plusieurs protocoles de communication, comme TCP/IP, peuvent être disponibles. Dans ce cas, l'agent peut décider du type de protocole utilisé pour chaque connexion.

Communication primitive : Cette catégorie de fonctionnalités permet de communiquer avec les dispositifs du système, comme des détecteurs et des actuateurs. Typiquement, ce type de communication est peu évolué et il offre les fonctionnalités nécessaires pour recevoir des données des détecteurs, pour envoyer des commandes aux actuateurs et, lorsque possible, pour valider la connexion ou l'état du dispositif.

Une préoccupation majeure concernant le niveau communication est la facilité avec laquelle de nouvelles fonctionnalités de communication peuvent être ajoutées. Il doit être facile d'ajou-

ter de nouveaux protocoles de communication ou de nouveaux types de communication, de détecteurs ou d'actuateurs afin de bien supporter l'évolution de la facette matérielle du système. Il est impératif que ce niveau soit facilement extensible et portable afin d'assurer la longévité d'un agent ou d'un système.

Une autre préoccupation est d'offrir une interface semblable pour la communication évoluée et la communication primitive. Une interface unifiée est préférable, car les spécificités de chacun des types de communication sont encapsulées et la facilité d'utilisation et la portabilité du système sont augmentées.

Le niveau communication peut interagir directement avec le matériel ou indirectement avec l'aide de fonctionnalités offertes par le système d'opération sous-jacent. Ce niveau est indépendant de l'application et son interface est indépendante du système d'exploitation. Par contre, à cause de la nature des fonctionnalités, son implémentation peut dépendre du système d'opération et du matériel utilisé.

5.2.3 Niveau interface

Le niveau interface a deux utilités principales : fournir une interface de communication et ajouter des fonctionnalités de communication. Voici une description de ces deux utilités :

Fournir une interface de communication : Ce niveau sert d'interface entre le niveau communication et les niveaux supérieurs du modèle GAM en offrant une interface évoluée pour les fonctionnalités du niveau communication et des outils pour la communication. Ce niveau cache les spécificités respectives de la communication évoluée et de la communication primitive. Cette interface est indépendante de l'application, du système d'opération et du niveau communication.

Fournir des fonctionnalités ajoutées : Ce niveau ajoute des fonctionnalités de communication évoluées et des types d'échanges plus sophistiqués que ceux offerts par le niveau communication. Des fonctionnalités de communication de haut niveau sont ajoutées aux fonctionnalités de base du niveau communication, comme la négociation. Les interfaces spécifiques nécessaires à ce type d'interaction sont fournies aux niveaux supérieurs

afin qu'ils puissent les exploiter de manière uniforme. Ainsi, les fonctionnalités de haut niveau de communication typiquement reliées au SMA sont définies, standardisées et implémentées à ce niveau.

Tout comme dans le niveau communication, un effort particulier doit être fait afin de faciliter l'ajout de nouvelles fonctionnalités de communication évoluées et d'interfaces de communication. L'ajout de nouvelles fonctionnalités assure une longue durée de vie à l'environnement de développement et aux systèmes développés, car les dernières techniques de communication peuvent être intégrées dans les outils de développement et être ensuite utilisées par les concepteurs. L'extensibilité est une caractéristique importante du modèle et elle ne doit pas être compromise.

Afin de préserver l'indépendance de ce niveau par rapport au système d'opération et au matériel, l'implémentation des fonctionnalités ajoutées doit faire en sorte qu'elles interagissent uniquement entre elles et avec celles du niveau communication. Dans le cas contraire, l'indépendance, l'intégrité et la portabilité de ce niveau pourraient être compromises.

5.2.4 Niveau moteur

Ce niveau spécifie et implémente un paradigme et ses fonctionnalités particulières associées. Le paradigme instancié par le niveau moteur n'est pas nécessairement le même que celui utilisé pour réaliser les niveaux communication et interface. Aussi, si le paradigme ne prévoit pas toutes les fonctionnalités de communication offertes par le niveau interface, alors leurs interfaces et leurs implémentations doivent être rendues disponibles. Deux parties sont distinguées au niveau moteur :

Paradigme choisi : Le niveau moteur est instancié différemment selon le paradigme choisi et les fonctionnalités fournies avec chacun peuvent être différentes. Par exemple, un agent qui utilise le langage PROLOG aurait un niveau moteur composé du moteur d'inférence et de ses fonctionnalités associées. Ces fonctionnalités sont très différentes de celles qui composent un agent utilisant comme paradigme les réseaux de neurones artificiels.

Interfaces ajoutées : Le niveau moteur utilise les fonctionnalités du niveau interface pour les communications agent. Si le paradigme ne prévoit pas toutes les interfaces de communication correspondantes à celles du niveau interface, elles doivent être ajoutées. Cela permet de les rendre accessibles au niveau connaissance et d'en permettre l'exploitation.

Les agents qui sont basés sur des paradigmes connus peuvent être standardisés et fournis par l'environnement de développement. Ainsi, les concepteurs peuvent les utiliser directement pour la conception de leur système. Par exemple, un agent qui est basé sur la logique floue pourrait être fourni par l'environnement de développement.

À mesure que de nouveaux paradigmes seront inventés, de nouvelles instances du niveau moteur seront conçues, développées et rendues disponibles. L'ajout de nouveaux paradigmes est encouragé ; il contribue à rendre l'environnement de développement plus intéressant en augmentant sa diversité et ses capacités.

5.2.5 Niveau connaissance

Ce niveau spécifie la configuration et l'usage particulier du niveau moteur ; c'est à ce niveau qu'est exploité le paradigme. Les connaissances, les habiletés et les capacités de l'agent sont spécifiées et implémentées à ce niveau par les concepteurs d'agents. Les tâches accomplies par l'agent et la manière dont elles peuvent être complétées sont fixées à ce niveau. Pour un paradigme donné, c'est le seul niveau qui est dépendant de l'application. Ce niveau comporte deux facettes :

Exploitation du paradigme : Cette facette correspond à la partie de l'agent qui doit être programmée par ceux qui réalisent une application. Par exemple, si l'agent est basé sur les réseaux de neurones artificiels, l'exploitation du paradigme est représentée par la topologie du réseau de neurones, les poids associés aux interconnexions et le choix de la méthode d'activation.

Exploitation des communications : La configuration des communications est faite à ce niveau-ci. Il en est de même pour le traitement du contenu des messages qui sont

reçus du monde extérieur. Bien qu'un certain traitement de base puisse être fait aux niveaux inférieurs, c'est à ce niveau que leur interprétation complète est faite et que leur signification précise est connue. L'agent peut alors décider des actions ou des réponses appropriées en accord avec le contenu de ces messages.

5.2.6 Remarques

En admettant que les agents les plus courants sont fournis par l'environnement de développement, les concepteurs n'auront qu'à configurer et à programmer dans le niveau connaissance du modèle, car les fonctionnalités de communication et le paradigme sont fournis par les trois autres niveaux. Si nécessaire, de nouvelles fonctionnalités peuvent être ajoutées en réutilisant celles des niveaux inférieurs. Le contenu des niveaux du modèle GAM est résumé à la figure 5.2.

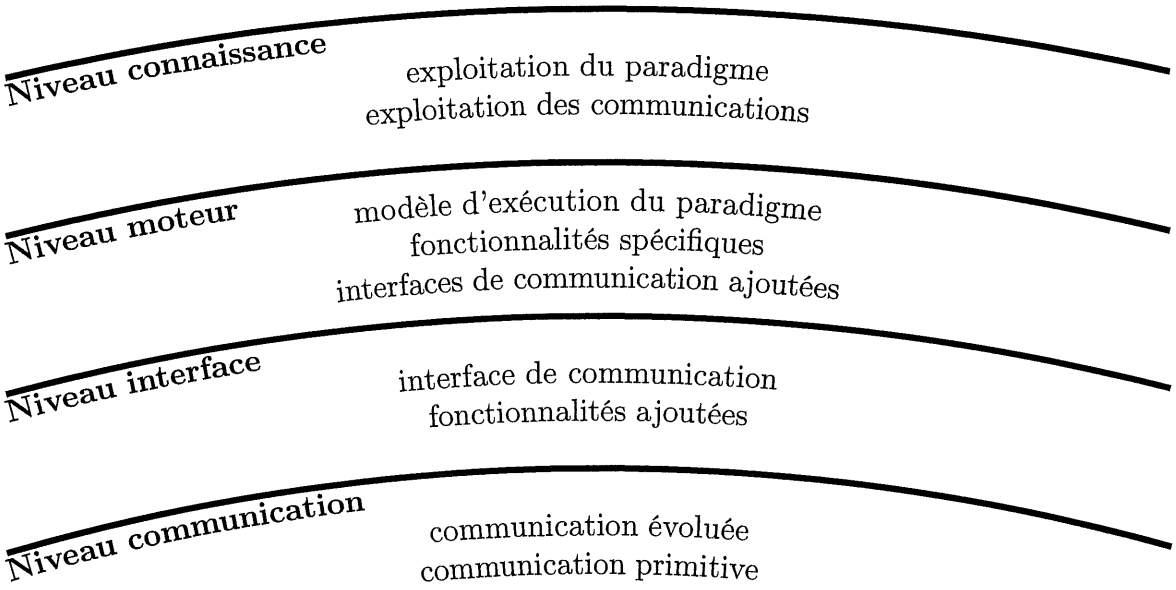


Figure 5.2 – Définition des niveaux de l'architecture GAM.

Un agent issu du modèle GAM peut avoir différents degrés d'évolution selon le nombre de niveaux qui sont instanciés. Trois degrés d'évolution sont distingués :

Agent abstrait : Un agent dont seulement les deux premiers niveaux du modèle GAM sont instanciés. Un agent abstrait ne peut pas être instancié ni exécuter une tâche, car il

ne possède pas les niveaux moteur et connaissance. Essentiellement, ce type d'agent possède les outils pour gérer des communications, mais rien pour les exploiter.

Agent spécialisé : Un agent dont seulement les trois premiers niveaux du modèle GAM sont instanciés. Un agent spécialisé peut être instancié, mais il n'exécute aucune tâche et il ne peut pas être intégré à un SMA, car le niveau connaissance n'est pas spécifié. Il ne peut pas gérer de communications, car elles sont configurées au niveau connaissance. Ce type d'agent possède un paradigme, mais il ne possède pas de connaissances ou d'habiletés pour les utiliser. Il est une sorte d'agent zombi qui sert de base pour bâtir d'autres agents.

Agent fonctionnel : Un agent dont tous les niveaux du modèle GAM sont instanciés. Généralement, l'agent fonctionnel est simplement appelé agent, car la plupart du temps un agent désigne un agent fonctionnel. Ce type d'agent peut être instancié et il peut exécuter les tâches qui lui sont dédiées selon les besoins des concepteurs.

Le modèle GAM permet d'obtenir des agents hétérogènes en spécialisant et en instanciant différemment le niveau moteur. Pour un agent donné, le niveau moteur représente un paradigme spécifique, et d'un agent à l'autre le paradigme peut être différent.

La figure 5.3 montre un exemple de SMA basé sur l'architecture GAM. Elle permet de souligner que l'architecture de communication et les types d'agents qui peuvent être instanciés sont deux domaines d'un intérêt particulier pour intégrer des agents dans un même système.

Architecture de communication

Du point de vue de l'environnement de développement, la topologie du réseau de communication du SMA ne peut pas être fixée *a priori*. Une application peut utiliser avantageusement toute topologie ou hiérarchie, qu'elle soit basée sur les acteurs, les tableaux noirs ou tout autre principe organisateur. Les concepteurs de l'application doivent pouvoir créer la topologie qu'ils désirent en accord avec leurs besoins. Un agent doit pouvoir établir des liens avec tout autre agent, pour négocier ou échanger des données. Cette liberté de création de liens donne la possibilité d'avoir des applications qui utilisent une architecture de communication connue ou quelconque, comme celle illustrée à la figure 5.4.

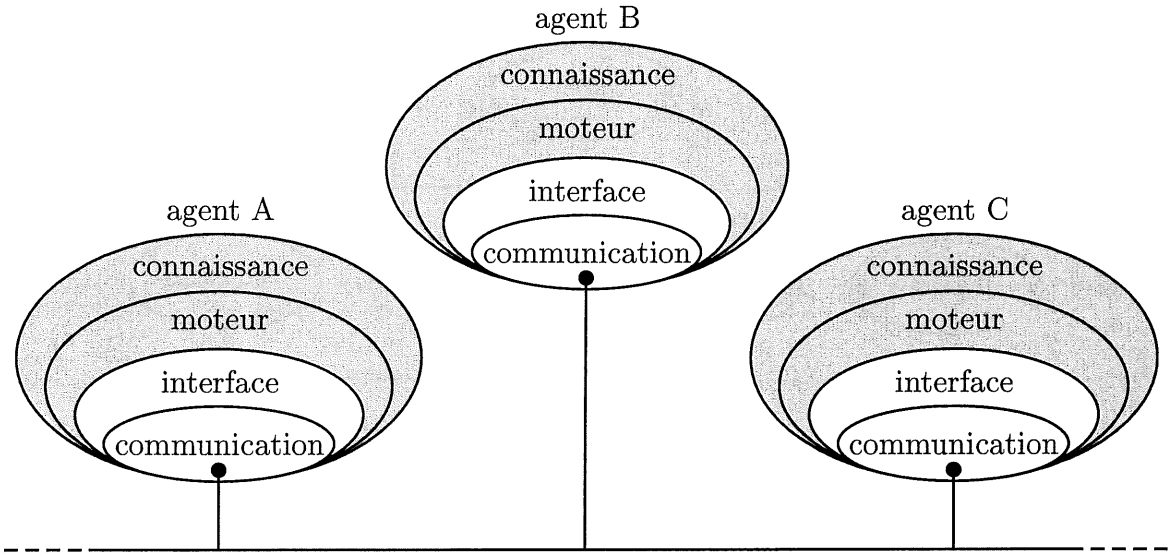


Figure 5.3 – Exemple d'un système multi-agents basé sur l'architecture GAM.

La possibilité de créer une architecture de communication quelconque est intéressante. Des liens secondaires de communication entre agents peuvent améliorer les performances du système. Ils sont des liens privilégiés pour l'échange d'informations spécifiques et exclusives. Les liens secondaires distribuent l'information sur le réseau de connexions. Ils peuvent ainsi réduire la taille des bases de données ainsi que le flot d'information sur les liens primaires du système.

Même si les concepteurs sont libres de construire une architecture quelconque, les outils de l'environnement de développement doivent fournir les fonctionnalités nécessaires pour les modèles classiques, comme le tableau noir et les acteurs. Ces outils doivent aussi permettre d'établir des liens de communication indépendamment de la distribution des agents. L'architecture de communication et la distribution du système doivent être transparents du point de vue des agents.

Types d'agents

Les types d'agents dans un SMA peuvent être hétérogènes, c'est-à-dire que leur spécialisation au niveau moteur est différente. L'intégration d'un agent dans un SMA doit être possible pour tous les paradigmes. Un ensemble de fonctionnalités de base commun à tous les paradigmes doit être défini afin d'assurer l'intégration de tous les types d'agents.

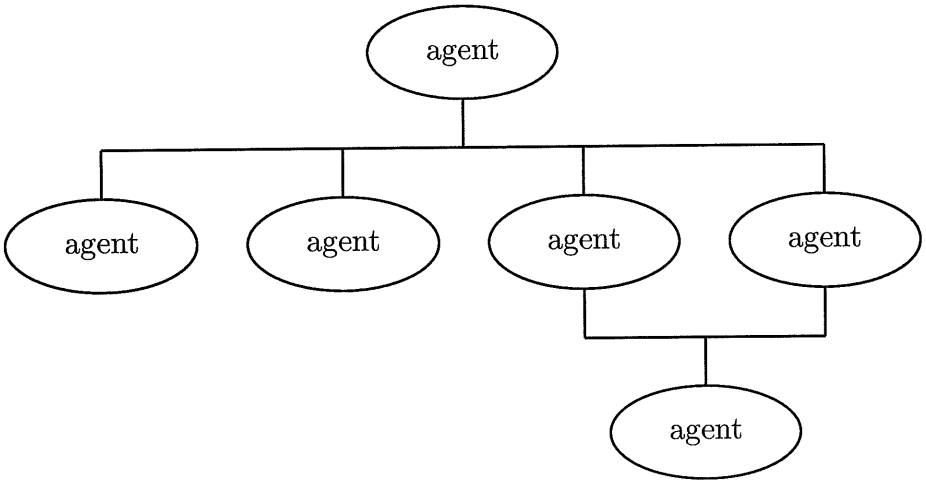


Figure 5.4 – Architecture de communication quelconque.

Les agents impliqués dans une communication peuvent déterminer le type d'informations qu'ils échangent sur les liens. Ce sont aux agents de déterminer les types pertinents pour leurs besoins. Les mécanismes de base de communication de l'environnement de développement ne doivent donc pas imposer de restrictions à ce niveau.

La communication de données entre entités distribuées est un domaine exploré de manière extensive, surtout depuis quelques années avec la venue d'internet. L'établissement de liens de communication et l'échange de messages entre deux entités distribuées ne constituent généralement pas un problème. Du point de vue des SMA, les problèmes de communication sont à un plus haut niveau d'abstraction, comme avec les protocoles de négociation ou de planification multi-agents.

L'agent abstrait et les agents spécialisés doivent être conçus afin d'obtenir des agents fonctionnels. Les sections qui suivent permettent de concevoir ces agents.

5.3 Agent abstrait

Un agent abstrait possède uniquement les niveaux communication et interface du modèle GAM, comme expliqué dans la section 5.2. Il s'agit maintenant de définir plus précisément quelles sont les fonctionnalités nécessaires à chacun de ces niveaux.

Niveau communication

Ce niveau définit les fonctionnalités de base de la communication évoluée de l'agent. L'établissement de liens de communication et l'échange de messages entre entités distribuées est une problématique connue et résolue. Ces solutions sont utilisées de manière extensive, surtout depuis la venue d'internet. Plusieurs protocoles, comme TCP/IP, sont disponibles pour établir une connexion fiable.

Essentiellement, en ce qui concerne la communication évoluée, un agent a besoin d'établir des liens, de les valider et d'échanger des données ou des messages. Aussi, si un autre agent tente d'établir une connexion, l'agent doit être en mesure d'accepter ou de rejeter la demande. L'agent a besoin de gérer les messages reçus et à envoyer, alors des mécanismes de listes et de gestion de listes sont fournis. Des fonctionnalités utilitaires peuvent être offertes afin de rendre plus simples les opérations les plus courantes et permettre d'avoir plusieurs listes de messages.

Des fonctionnalités doivent aussi être fournies pour la communication primitive. De manière conceptuelle, l'utilité de base pour un capteur est d'en récupérer des données et pour un actuateur de lui fournir des commandes. Dans les deux cas, il peut être nécessaire d'initialiser le dispositif et aussi de vérifier si son état est satisfaisant ou défectueux.

Niveau interface

Le niveau interface a un double usage. Le premier est d'offrir aux niveaux supérieurs une interface uniforme pour les des deux types de communication offerts par le niveau communication. Il sert d'intermédiaire de découplage et il rend transparent l'implémentation des mécanismes de communications évoluées et de communications primitives. Le deuxième usage est d'ajouter des fonctionnalités et leurs interfaces pour les communications évoluées et des protocoles de haut niveau comme la négociation. Initialement, le choix des types de communication et des protocoles disponibles dans l'environnement est petit. Avec le temps, la diversité des protocoles disponibles grandira, car ils seront ajoutés à l'environnement au fur et à mesure qu'ils seront développés.

Le tableau 5.1 résume les fonctionnalités de communication identifiées. L'agent abstrait sert de fondation pour les agents spécialisés de base. Les sections suivantes montrent de quelle manière.

FONCTIONNALITÉS ÉVOLUÉES
– initier une connexion/déconnexion
– accepter une connexion/déconnexion
– valider l'état d'une connexion
– envoyer et recevoir des messages
– gérer des listes de messages
– fonctionnalités évoluées ajoutées

FONCTIONNALITÉS PRIMITIVES
– récupérer des données d'un capteur
– fournir des données à un actuateur
– initialiser un capteur/actuateur
– vérifier l'état d'un capteur/actuateur

Tableau 5.1 – Fonctionnalités principales de communication de l'agent abstrait.

5.4 Agents spécialisés de base

Le modèle GAM permet d'obtenir des agents hétérogènes en spécialisant différemment le niveau moteur. Le modèle GAM ne doit pas simplement permettre de générer un ou deux types d'agents, il doit permettre d'engendrer des types d'agents avec les paradigmes existants, comme la logique, la logique floue, les réseaux de neurones artificiels et les algorithmes génétiques.

Cette section discute de la spécification du niveau moteur et de la manière dont il doit être instancié pour qu'il puisse engendrer des agents utilisant les paradigmes les plus courants. Elle montre aussi comment le niveau connaissance interagit avec le niveau moteur dans chacun de ces cas. Le chapitre 6 présente une implémentation et un exemple pour chacun des paradigmes présentés afin de valider les modélisations proposées.

5.4.1 Agent logique

Un agent logique a comme paradigme la programmation logique. Il utilise un langage de programmation de type logique comme PROLOG [31, 139]. Afin de pouvoir exploiter le pa-

radigme au niveau connaissance, le niveau moteur doit contenir toute la mécanique liée à ce type de programmation et fournir une interface pour les fonctionnalités de communication.

La mécanique liée au paradigme logique est constituée d'un moteur d'inférence et des primitives de base permettant de l'exploiter. Le moteur d'inférence, aussi appelé interpréteur, constitue le modèle d'exécution du langage et spécifie les mécanismes d'implémentation du paradigme. Les primitives de base permettent de rendre l'usage de ce paradigme plus intéressant en offrant des fonctionnalités utilitaires, comme la lecture dans un fichier.

L'utilisateur d'un agent logique exploite le paradigme logique en composant des faits et des règles au niveau connaissance. À l'exécution, les règles et les faits sont interprétés et des déductions logiques sont faites.

Généralement, les langages logiques ne prévoient pas de fonctionnalités ou de mécanismes de communication avec une composante matérielle ou entre entités distribuées. De nouvelles interfaces doivent être ajoutées afin que les mécanismes de communication du niveau interface soient standardisés et rendus disponibles au niveau connaissance, comme le prescrit le modèle GAM. Ainsi, les concepteurs d'agents peuvent exploiter les communications au niveau connaissance. L'interpréteur, ses fonctionnalités et les interfaces ajoutées de communication sont implémentés et intégrés au niveau moteur.

La configuration et les caractéristiques des communications évoluées ou primitives sont spécifiées par les concepteurs d'un agent à l'aide des interfaces offertes par le niveau moteur. L'agent logique peut communiquer les résultats de ses traitements aux agents intéressés ou répondre aux sollicitations des autres agents avec des requêtes spécifiques. Selon les besoins des concepteurs, le niveau de sophistication du contenu des communications peut être très élevé, car l'expressivité de la programmation logique le permet.

Un agent logique est obtenu avec les spécialisations données des niveaux moteur et connaissance. La figure 5.5 montre la spécialisation des quatre niveaux du modèle GAM pour un agent logique.

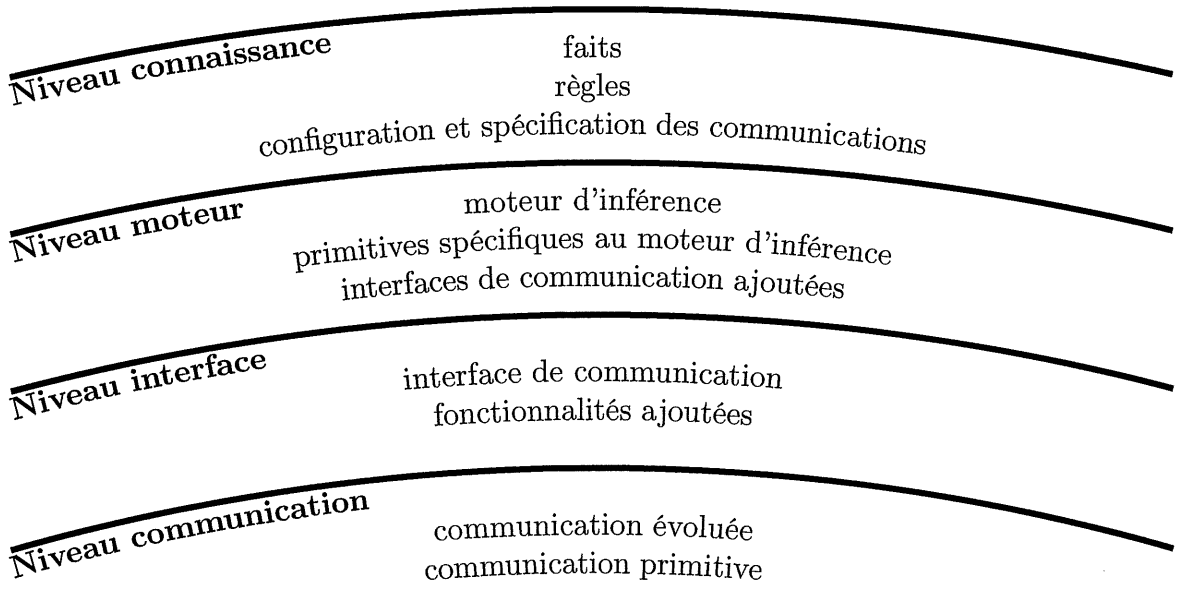


Figure 5.5 – Spécialisation du modèle GAM pour un agent logique.

5.4.2 Agent flou

Un agent flou est basé sur le paradigme de la logique floue [158, 159]. La logique floue et la logique étant très liées [11, 92], la spécialisation des niveaux moteur et connaissance de l'agent flou ressemble donc beaucoup à celle de l'agent logique.

Afin d'implémenter le paradigme de la logique floue, le niveau moteur doit contenir un interpréteur qui traite les règles floues (*fuzzy rules*) et les fonctions d'appartenance (*membership functions*). Cet interpréteur, aussi appelé le contrôleur flou (*fuzzy controller*), constitue le modèle d'exécution du paradigme. L'interpréteur doit aussi assurer l'application de la méthode de défuzzification (*defuzzification*) choisie par les concepteurs.

L'utilisateur d'un agent flou exploite le paradigme au niveau connaissance. Il compose des règles floues, définit des fonctions d'appartenance et il choisit une méthode de défuzzification afin de générer des résultats précis.

La logique floue ne prévoit pas de mécanismes de communication agent. Donc, tout comme avec l'agent logique, des interfaces sont ajoutées afin de standardiser et de rendre disponible aux niveaux moteur et connaissance les mécanismes de communication issus du niveau in-

terface. L'interpréteur, ses fonctionnalités et les interfaces ajoutées de communication sont implémentés et intégrés au niveau moteur, comme le prescrit le modèle GAM.

La logique floue manipule des styles d'entrées et de sorties semblables à celles des capteurs et des actuateurs. Un agent flou est donc bien adapté pour les communications primitives. Une nouvelle donnée d'une entrée cause un ajustement de la valeur courante d'une fonction d'appartenance. L'ajustement d'une sortie du système correspond à l'ajustement d'une commande d'un actuateur.

L'agent flou est limité au niveau de la sophistication du contenu pour les communications évoluées, car les entrées et les sorties s'apparentent à la communication primitive. Il est limité à envoyer aux agents intéressés le statut de ses entrées et de ses sorties et de recevoir des messages permettant la modification d'une entrée. Les agents intéressés à communiquer de nouvelles valeurs en entrée ou à être informés des résultats en sortie n'ont qu'à établir une connexion avec l'agent flou. La logique floue ne prévoit pas la communication évoluée, elle est donc réalisée à l'aide d'interfaces offertes par le niveau moteur qui prennent l'allure de directives spéciales au niveau connaissance afin d'obtenir des actes de communication.

Avec les spécialisations prescrites des niveaux moteur et connaissance, un agent flou est obtenu. La figure 5.6 montre la spécialisation des quatre niveaux du modèle GAM pour un agent flou.

5.4.3 Agent neural

Un agent neural est basé sur le paradigme des réseaux de neurones artificiels (RNA) [104, 106, 133]. L'implémentation d'un RNA se fait en deux étapes majeures. La première étape est la conception du RNA qui consiste à décider de la topologie du réseau et de l'entraînement du réseau. La seconde étape est l'utilisation du RNA conçu à la première étape dans un système réel. L'agent neural permet de réaliser cette deuxième étape.

Le niveau moteur de l'agent neural est composé d'un calculateur permettant de générer les sorties du RNA en accord avec ses entrées. Le calculateur est le modèle d'exécution des RNA. Selon les types de réseaux, différentes méthodes de calcul peuvent être utilisées par le

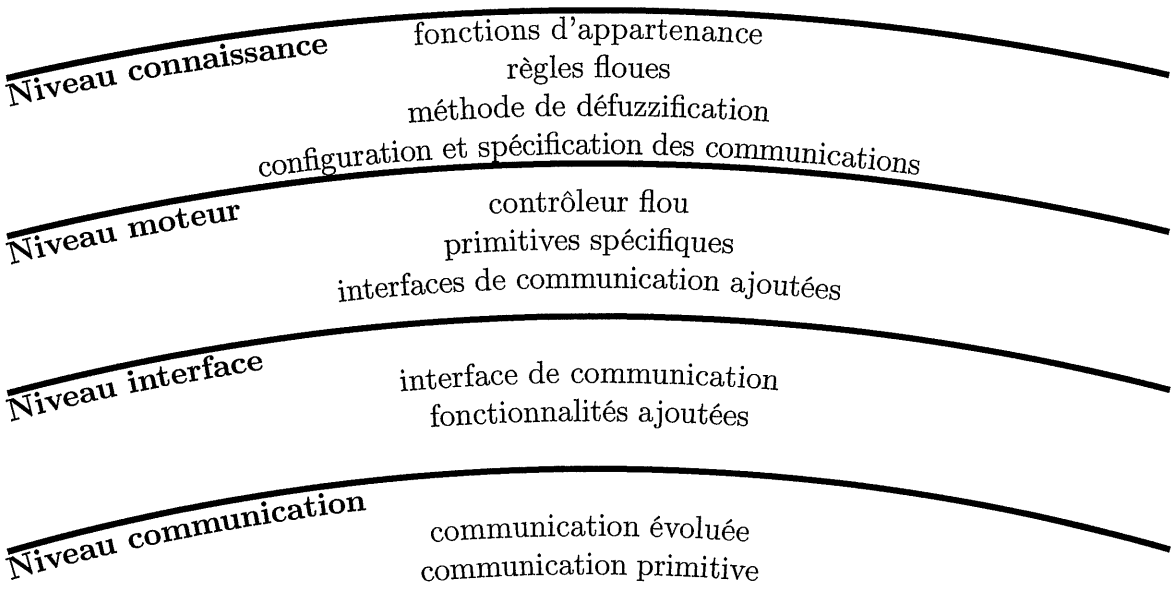


Figure 5.6 – Spécialisation du modèle GAM pour un agent flou.

calculateur. Il faut alors des mécanismes permettant d'identifier au niveau connaissance la méthode préférentielle de calcul.

Au niveau connaissance, les concepteurs d'un agent neural spécifient la topologie du réseau, les poids associés aux connexions et la méthode d'activation des neurones.

Le paradigme des RNA ne prévoit pas de mécanismes de communication agent ou avec une composante matérielle. Des interfaces sont donc ajoutées afin de standardiser et de rendre disponibles au niveau connaissance les mécanismes de communication issus du niveau interface. Le calculateur et les interfaces ajoutées de communication sont implémentés et intégrés au niveau moteur, comme le prescrit le modèle GAM.

Les RNA traitent des styles d'entrées et de sorties qui peuvent s'apparenter à des capteurs et des actuateurs. Pour les communications primitives, il suffit de faire correspondre une donnée venant d'un capteur avec une entrée du RNA et une sortie avec la valeur fournie à un actuateur.

Les communications évoluées sont limitées, tout comme avec l'agent flou. La réception d'un message correspond à une nouvelle valeur d'entrée du RNA et une nouvelle valeur de sortie

peut générer l'envoi d'un message. Les agents intéressés à communiquer de nouvelles valeurs en entrée ou à être informés des résultats en sortie n'ont qu'à établir une connexion avec l'agent neural. Les RNA ne prévoient pas de communication évoluée, elles sont donc réalisées à l'aide d'interfaces offertes par le niveau moteur qui prennent l'allure de directives spéciales au niveau connaissance afin d'obtenir des actes de communication.

Avec les spécialisations prescrites des niveaux moteur et connaissance, un agent neural est obtenu. La figure 5.7 montre la spécialisation des quatre niveaux du modèle GAM pour un agent neural.

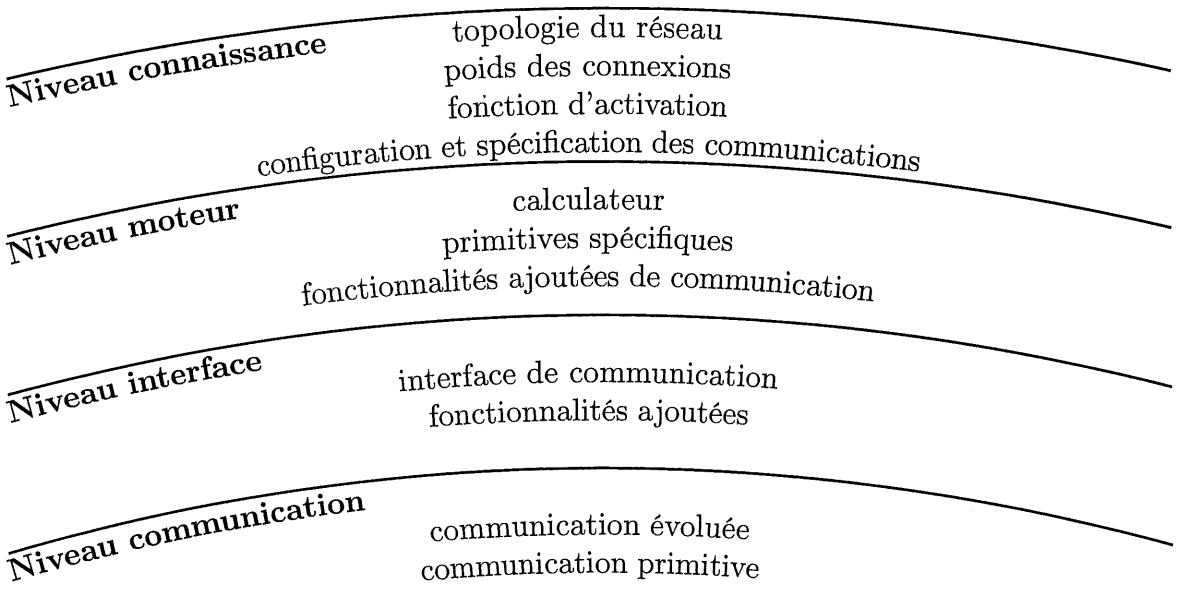


Figure 5.7 – Spécialisation du modèle GAM pour un agent neural.

5.4.4 Agent génétique

Un agent génétique est basé sur le paradigme des algorithmes génétiques [67, 68]. La réalisation d'un système génétique est constituée de deux phases. La première demande de concevoir un chromosome (aussi appelé individu) à partir d'un code génétique, de trouver une fonction d'évaluation du chromosome et de fixer certains paramètres de contrôle pour la simulation. La deuxième phase consiste à exécuter une simulation de la sélection natu-

relle sur une population de chromosomes. La simulation de la sélection naturelle constitue le modèle d'exécution du paradigme.

Le niveau moteur doit être composé des fonctionnalités permettant de gérer les différentes étapes de l'algorithme génétique qui sont les suivantes :

1. construire une population initiale d'individus ;
2. appliquer une fonction d'évaluation (*fitness function*) sur les individus ;
3. faire la reproduction sexuée des individus retenus à l'étape précédente ;
4. faire des mutations sur un certain nombre d'individus ;
5. terminer la simulation ou répéter et créer une nouvelle génération d'individus et aller à l'étape 2.

Au niveau connaissance, les concepteurs d'un agent génétique doivent fournir les caractéristiques définissant un individu et la configuration de la simulation. Plus précisément, ils doivent fournir les items suivants [152] :

- un **chromosome** qui spécifie le code génétique et sa fonction d'évaluation ;
- des **constructeurs** qui créent un chromosome à partir d'une liste de gènes (genèse) ou à partir de deux chromosomes (reproduction) ;
- des **écrivains** qui modifient de manière aléatoire certains gènes d'un chromosome (mutation) ;
- une **configuration** de simulation qui spécifie le nombre maximum d'individus, le taux de reproduction, le taux de mutation, le critère de terminaison de la simulation et la population initiale.

Le paradigme des algorithmes génétiques ne prévoit pas de mécanismes de communication agent ou avec une composante matérielle. Donc, des interfaces sont ajoutées afin de standardiser et de rendre disponible au niveau connaissance les mécanismes de communication issus du niveau interface. Le gestionnaire de l'algorithme génétique et les interfaces ajoutées de communication sont implémentés et intégrés au niveau moteur, comme le prescrit le modèle GAM.

La situation des communications pour l'agent génétique est différente de celle des autres agents. Les algorithmes génétiques trouvent une solution, un chromosome particulier, à partir d'une population initiale. Les entrées et les sorties ne correspondent pas à des dispositifs du

style capteur ou actuateur, bien que suite à l'analyse du résultat d'une simulation des actions puissent être posées.

Pour les communications évoluées, la réception d'un message correspond à une demande de simulation et la solution correspond à une sortie qui peut générer l'envoi de messages. Les agents intéressés à demander une simulation ou à être informés des solutions trouvées n'ont qu'à établir une connexion avec l'agent génétique. Les algorithmes génétiques ne prévoient pas de communication évoluée, elles sont donc réalisées à l'aide d'interfaces offertes par le niveau moteur et qui prennent l'allure de directives spéciales au niveau connaissance afin d'obtenir des actes de communication.

Avec les spécialisations prescrites des niveaux moteur et connaissance, un agent génétique est obtenu. La figure 5.8 montre la spécialisation des quatre niveaux du modèle GAM pour un agent génétique.

5.4.5 Agent procédural

Un agent procédural a comme paradigme un langage procédural comme le C [82] ou un langage orienté objet comme le C++ [140]. Un agent procédural est un cas particulier parmi les agents spécialisés de base, car le langage procédural du paradigme (L^+) et celui utilisé pour l'implantation de l'agent (L^-) peuvent être les mêmes ou être différents. Typiquement, les environnements de développement sont réalisés avec des langages procéduraux ou orienté objet. Dans ce cas, les langages L^+ et L^- peuvent être semblables. C'est ce qui rend l'agent procédural particulier, car dans les cas des agents précédents, les langages L^+ et L^- sont nécessairement différents.

Le cas où le langage utilisé est le même à tous les niveaux ne pose pas de problèmes particuliers. Dans le cas contraire, par exemple le langage C pour L^- et le langage PASCAL [77] pour L^+, il doit y avoir des mécanismes permettant l'intégration des deux langages. Sinon, le langage L^+ doit être interprété par le langage L^-. Dans ce cas, la vitesse d'exécution est perdue, un des avantages majeurs d'utiliser la programmation procédurale. Sauf pour

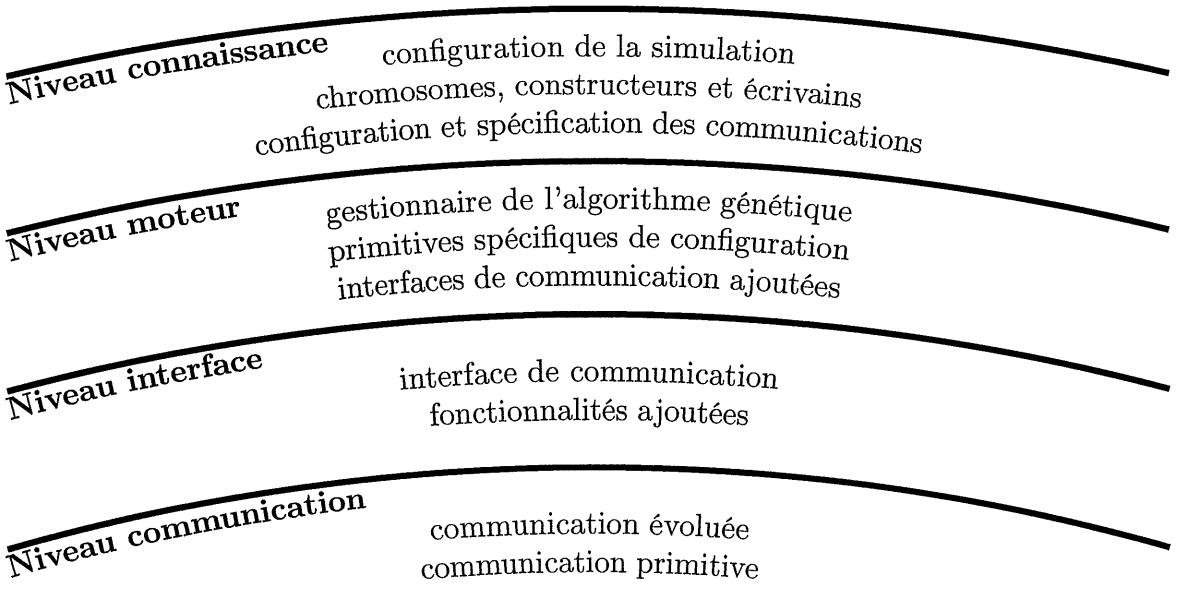


Figure 5.8 – Spécialisation du modèle GAM pour un agent génétique.

quelques langages de programmation où l'intégration avec d'autres langages est déjà prévue, il est préférable que les quatre niveaux utilisent le même langage pour L^+ et L^- afin d'éviter des problèmes inutiles d'intégration des deux langages ou d'interprétation du langage L^+ par le langage L^-.

Afin d'exploiter le paradigme procédural au niveau connaissance, le niveau moteur doit contenir toute la mécanique et les fonctionnalités du langage L^+. Le langage L^+ définit le modèle d'exécution de l'agent. Les mécanismes de base de fonctionnement du langage L^+ sont déjà fournis par le langage L^-, car les langages sont les mêmes ou l'intégration des deux est déjà prévue et les fonctionnalités sont déjà fournies par les niveaux inférieurs.

Au niveau connaissance, un usager d'un agent procédural exploite le paradigme en composant des instructions et en manipulant des données en accord avec les règles syntaxiques et sémantiques du langage L^+. De cette manière, des tâches sont exécutées et des actes de communication sont posés.

Le paradigme procédural ne prévoit pas de mécanismes de communication agent ou avec une composante matérielle, mais ils sont généralement fournis sous forme de librairies. Des interfaces sont ajoutées afin de standardiser et de rendre disponible au niveau connaissance

les mécanismes de communication issus du niveau interface. Les interfaces ajoutées de communication sont implémentées et intégrées au niveau moteur, comme le prescrit le modèle GAM.

La configuration des communications évoluées et primitives au niveau connaissance est faite à l'aide des interfaces offertes par le niveau moteur. Des fonctions ou des classes permettent de spécifier la configuration et les caractéristiques des communications. Les actes de communication sont accomplis à l'aide d'appels de fonctions ou de méthodes. Selon les besoins des concepteurs, le niveau de sophistication du contenu des communications peut être très élevé, car l'expressivité de la programmation procédurale ou de la programmation orientée objet le permet.

Avec les spécialisations prescrites des niveaux moteur et connaissance, un agent procédural est obtenu. La figure 5.9 montre la spécialisation des quatre niveaux du modèle GAM pour ce type agent.

5.4.6 Remarques

Les agents spécialisés de base ne font que réagir à un message venant de l'extérieur ou à un changement d'état d'un capteur, selon les spécifications du niveau connaissance par les concepteurs. La réponse de l'agent, s'il y en a une, est une commande à un actuateur ou des messages dirigés vers d'autres agents. Ceci est dû aux limitations de certains paradigmes qui ne peuvent pas traiter le contenu de messages complexes qui demandent une interprétation sophistiquée. L'agent procédural et l'agent logique sont les plus aptes à traiter et à analyser des messages complexes et sophistiqués.

Si des communications avec des contenus sophistiqués doivent absolument être utilisées avec un agent ayant peu de capacités pour les communications évoluées, alors il est toujours possible de jumeler deux agents dans une configuration de traduction (voir la section 3.4). Dans ce cas, comme illustré à la figure 5.10, un agent procédural peut servir de traducteur et agir comme intermédiaire entre l'agent flou et les autres agents du système. L'agent procédural traite les communications sophistiquées et communique à un niveau plus simple avec l'agent flou.

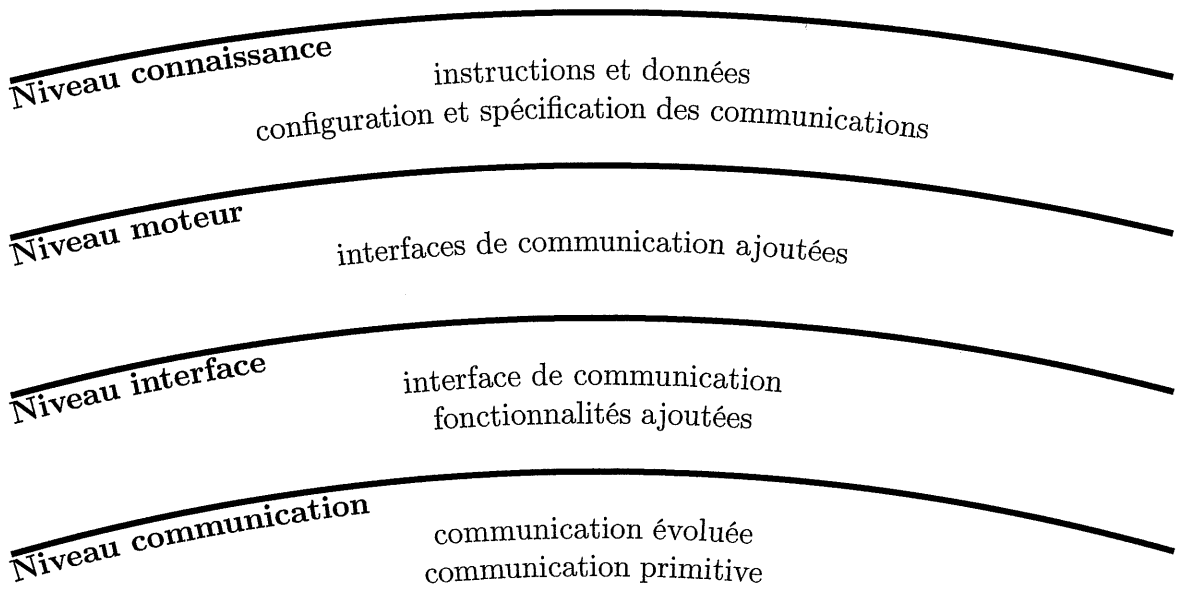


Figure 5.9 – Spécialisation du modèle GAM pour un agent procédural.

Afin de rendre les agents spécialisés de base plus performants, le parallélisme devrait être exploité afin d'éliminer les contraintes et limitations du traitement séquentiel. L'agent logique et l'agent procédural sont deux bons candidats pour le parallélisme. L'agent logique peut bénéficier de certaines extensions à la programmation logique, comme *Concurrent Prolog* [134] et PARLOG [30], lui permettant ainsi de faire de la programmation concurrentielle. L'agent procédural peut bénéficier du parallélisme avec l'aide des fonctionnalités standardisées offertes par la plupart des systèmes d'opération, comme les *threads*. La section suivante discute d'agents plus évolués qui ont des architectures qui peuvent prendre avantage du parallélisme.

5.5 Agents spécialisés évolués

Pour être intéressant, le modèle GAM doit permettre d'obtenir des agents plus complexes ou plus évolués que les agents spécialisés de base de la section précédente. Il doit permettre de recréer des agents évolués, comme les agents délibératifs, les agents réactifs et les agents hybrides. Cette section montre comment il est possible de recréer ces agents à partir des agents de base.

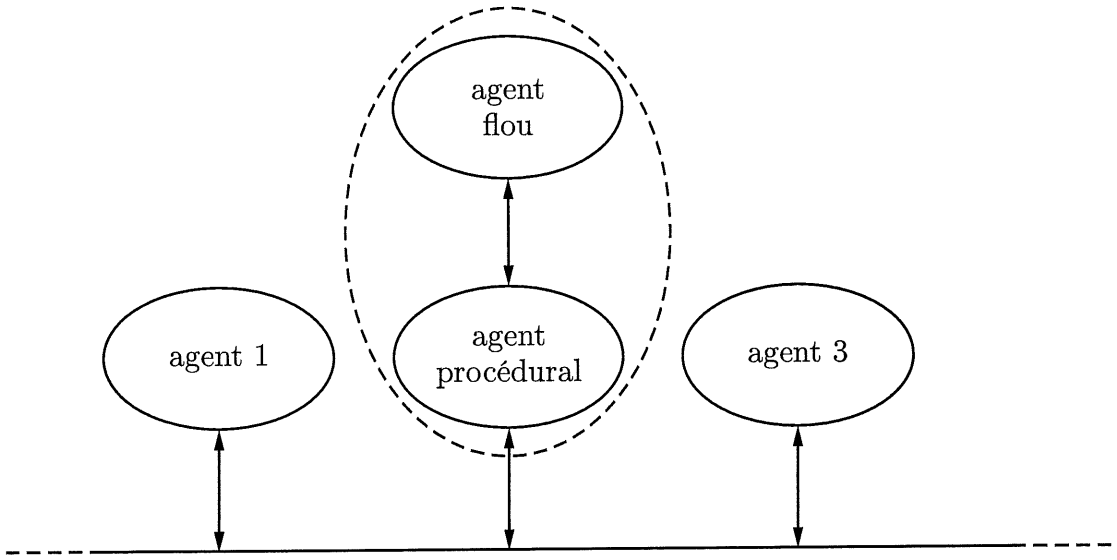


Figure 5.10 – Agent flou et agent procédural en configuration de traduction.

5.5.1 Agent délibératif

Wooldridge [155] résume les caractéristiques d'un agent délibératif. Il donne aussi une architecture générique pour un agent délibératif de type BDI. La figure 5.11 illustre une adaptation de cette architecture. Le fonctionnement de cette architecture est le suivant.

Une fonction de révision des croyances accepte de nouvelles entrées et ajuste les croyances de l'agent en accord avec les croyances actuelles. Les croyances représentent les informations que l'agent possède sur son environnement. À partir des croyances et des intentions, la fonction génération d'options détermine les différentes options qui se présentent, soit les désirs de l'agent. Ensuite, la fonction filtration, le processus délibératif de l'agent, détermine les intentions à partir des croyances, des désirs et des intentions. Finalement, la fonction sélection d'action sélectionne la prochaine action à poser par l'agent. Les intentions représentent les intérêts de l'agent ; elles représentent ce que l'agent s'est engagé à accomplir.

Cette description de l'agent délibératif correspond en partie à l'agent logique. Les croyances, les désirs et les intentions peuvent être représentés par des bases de connaissances. Les bases de connaissances sont généralement issues du paradigme logique et un tableau noir peut servir à leur implémentation. Les fonctions révision des croyances, génération d'options, filtration et sélection d'action peuvent être des faits et des règles qui exploitent les bases de connaissances.

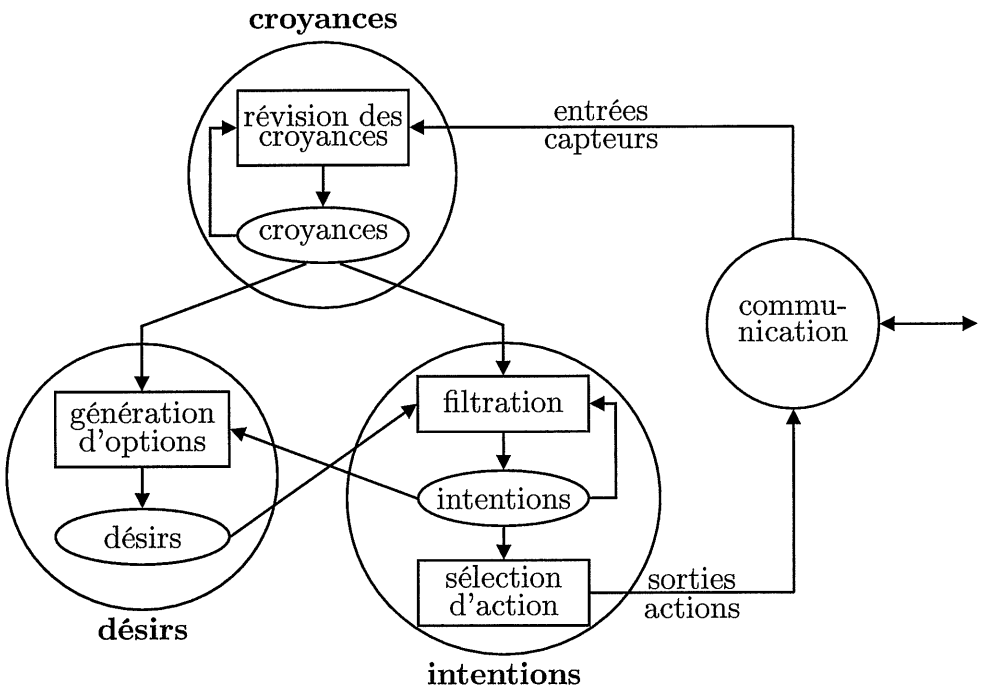


Figure 5.11 – Modélisation d'un agent délibératif (basée sur Wooldridge [155])

Un agent délibératif est recréé à partir d'un agent logique en y ajoutant les fonctionnalités des tableaux noirs, comme illustré à la figure 5.12.

Une alternative est de considérer l'agent délibératif comme étant composé internement de quatre agents, comme illustré à la figure 5.11. Cette modélisation est basée sur les agents croyances, désirs, intentions et communication. Les agents croyances, désirs et intentions sont de type logique, comme mentionné au paragraphe précédent. L'agent communication peut être de type logique ou procédural, il suffit qu'il puisse interagir convenablement avec les agents croyances et intentions et traiter adéquatement les communications évoluées. Cette modélisation distribuée a l'avantage de profiter du parallélisme. D'autres types d'agents délibératifs, comme ceux mentionnés à la section 2.4.1, peuvent aussi être recréés.

La division de l'agent délibératif en quatre modules internes peut être une division logique ou une division physique. Dans le cas d'une division logique, l'agent est alors structuré et organisé de manière à refléter correctement l'organisation proposée. Les différentes parties logiques évoluent alors à l'intérieur d'un même agent physique, chacune dans leur *thread*. Dans le cas d'une division physique, chaque module correspond réellement à un agent autonome.

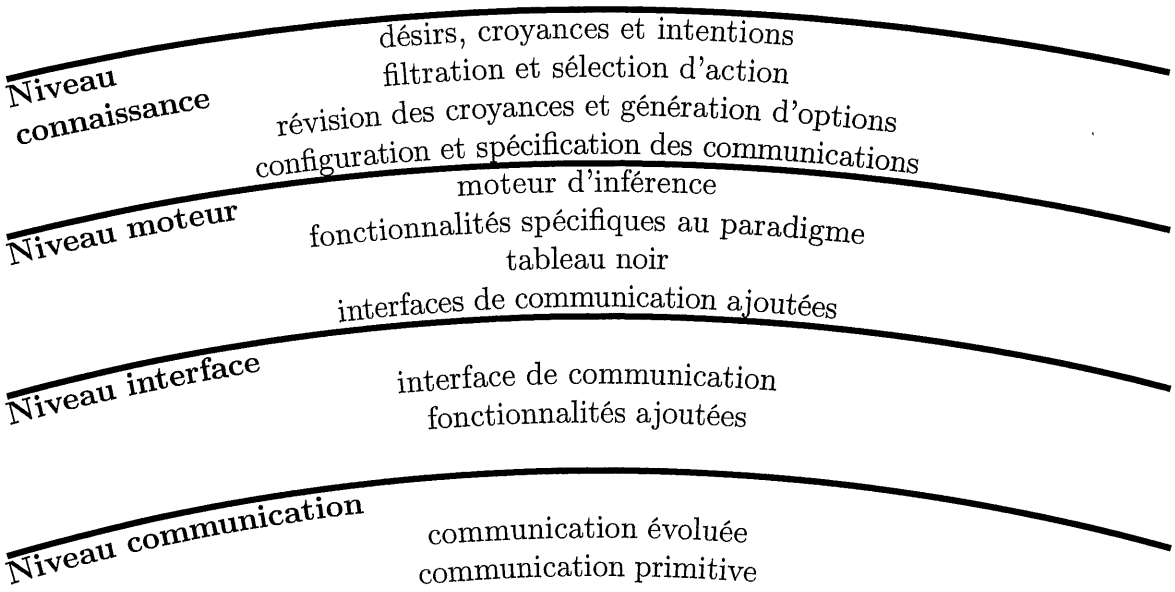


Figure 5.12 – Spécialisation du modèle GAM pour un agent délibératif.

Bien que les concepteurs puissent réaliser le tableau noir et concevoir un agent délibératif à partir de l'agent logique, il est préférable que le tableau noir soit fourni par l'environnement de développement. Ainsi, les concepteurs évitent des problèmes d'implémentation et surtout économisent du temps de développement.

D'autres extensions peuvent être ajoutées à l'agent logique afin de rendre l'agent délibératif plus performant et intéressant. Par exemple, des mécanismes d'ordonnancement, comme dans FLEX [123], peuvent faciliter le travail des concepteurs et rendre l'agent délibératif plus facile à implémenter.

5.5.2 Agent réactif

Contrairement aux agents délibératifs, la modélisation d'un agent réactif est assez uniforme dans la littérature. Elle est constituée de plusieurs niveaux de compétences qui s'exécutent en parallèle et qui tentent de contrôler le système. Ce type d'agent peut aussi être recréé à partir des agents spécialisés de base. La figure 5.13 illustre de quelle manière cette modélisation peut être effectuée.

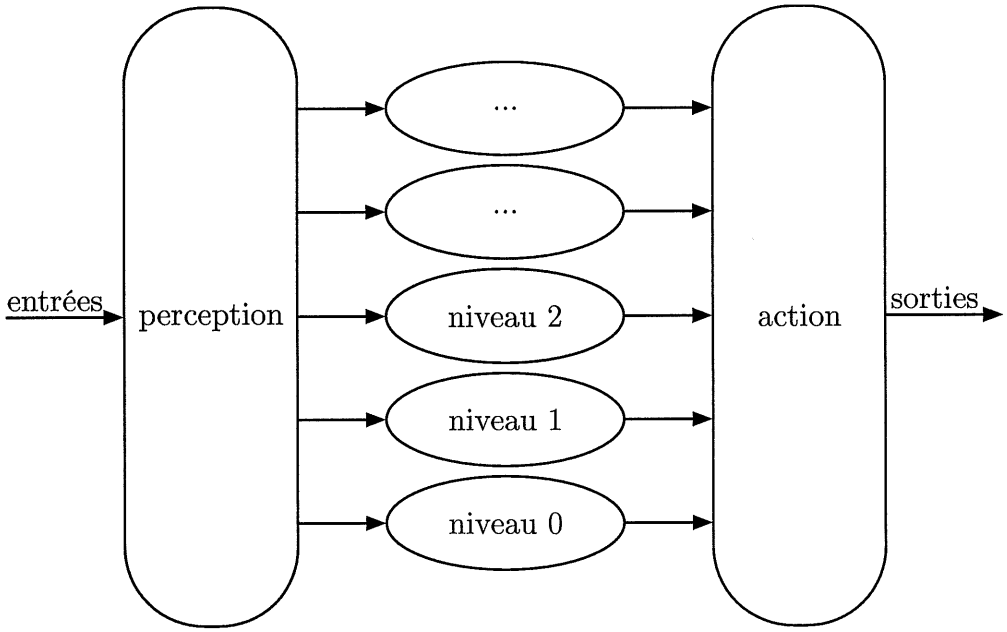


Figure 5.13 – Modélisation d'un agent réactif.

Le module perception s'occupe de la réception des communications et le module action s'occupe des communications sortantes. Généralement, les communications des agents réactifs sont primitives, mais elles peuvent aussi être évoluées, car les modules perception et action peuvent être de tout type. Les modules représentant les niveaux de compétences sont du type procédural, car le traitement des données est simple, comme le prescrit la philosophie des architectures réactives.

La modélisation proposée à la figure 5.13 peut être interprétée de deux manières. Elle peut être une division logique interne d'un agent, tout comme avec l'agent délibératif. À ce moment, l'architecture interne de l'agent est composée de modules homogènes de type procédural. Les *threads* sont utilisés pour tirer profit des avantages du parallélisme à l'intérieur d'un même agent. Si la modélisation est une division physique, alors tous les modules composant l'agent réactif sont des agents autonomes qui évoluent en parallèle, comme le prescrit l'architecture réactive.

Cette modélisation d'un agent réactif permet aux modules représentant les différents niveaux de compétences d'être hétérogènes. Ceci est une caractéristique intéressante, car rien n'empêche que chaque niveau de compétences soit basé sur des paradigmes différents. Elle

est également intéressante par le fait que le module action peut adopter des mécanismes de contrôle appropriés afin de gérer les commandes venant des différents niveaux de compétences, un des problèmes de cette architecture.

5.5.3 Agent hybride

Un agent hybride est composé d'une partie délibérative et d'une partie réactive qui collaborent. La figure 5.14 présente cette architecture basée sur les agents délibératifs et réactifs des sections précédentes. Elle est structurée afin que l'agent délibératif et l'agent réactif communiquent de manière adéquate. Les modules perception et action de l'agent réactif communiquent avec le module communication de l'agent délibératif.

La modélisation de l'agent hybride illustrée à la figure 5.14 permet d'exploiter le parallélisme, car elle est composée d'au moins deux agents autonomes : l'agent délibératif et l'agent réactif. Les modèles plus complexes d'agents hybrides, comme les TouringMachines [47], peuvent aussi être modélisés en combinant des agents délibératifs et des agents réactifs, et d'autres agents si nécessaire.

5.5.4 Remarques

Les agents spécialisés de base issus du modèle GAM permettent de générer les types d'agents évolués les plus courants. Les concepteurs peuvent donc intégrer tout type d'agent dans un système.

La modélisation des agents évolués de cette section représente uniquement une approche possible parmi plusieurs. D'autres modélisations plus complètes et plus complexes peuvent être utilisées pour recréer les agents évolués, surtout pour les agents délibératifs et hybrides.

Il est aussi à remarquer que les concepteurs de SMA peuvent non seulement utiliser les agents spécialisés de base et évolués mentionnés dans les sections précédentes, mais ils peuvent aussi créer leurs propres agents évolués et les intégrer au SMA.

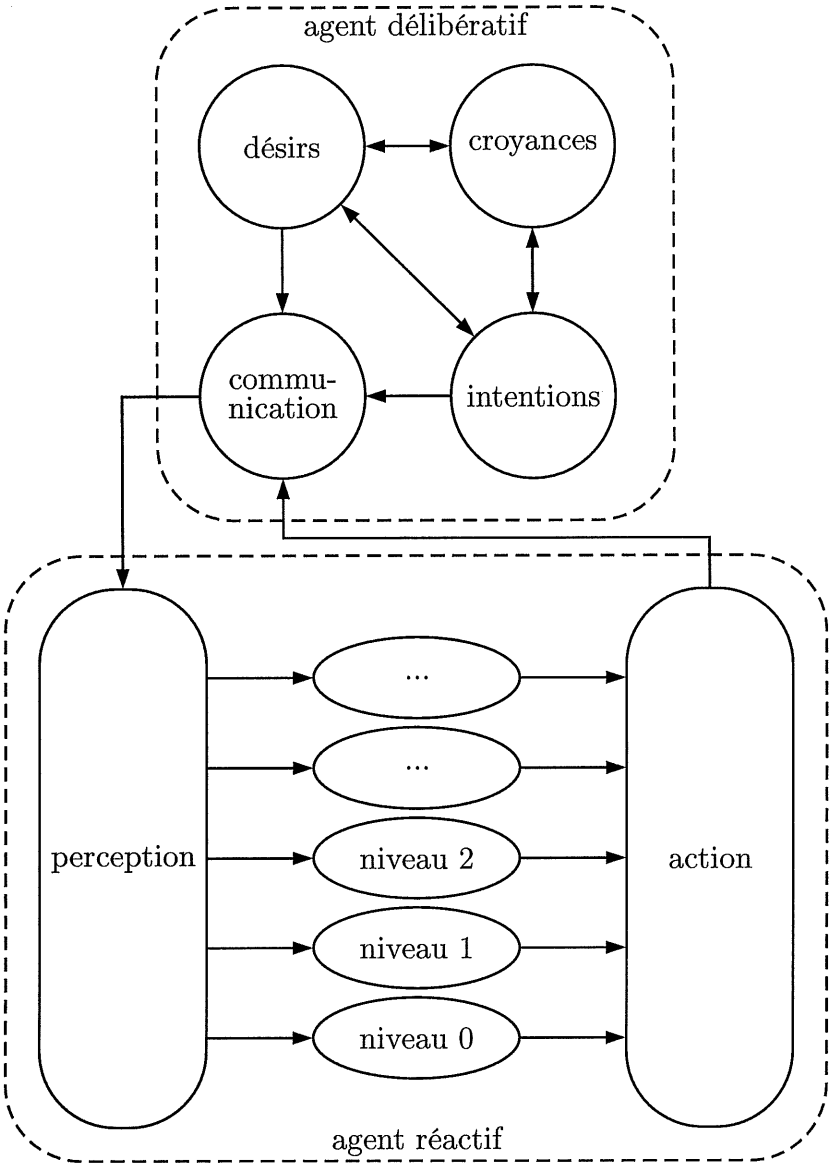


Figure 5.14 – Modélisation d'un agent hybride.

5.6 Développement de systèmes multi-agents

Bien que le modèle GAM génère des agents hétérogènes, les concepteurs ont besoins d'outils qui aident au développement de SMA. L'environnement doit aider les concepteurs en s'occupant de plusieurs des considérations de développement. Plusieurs tâches doivent être accomplies par l'environnement, car les concepteurs ne sont pas des experts dans tous les domaines et dans toutes les disciplines.

Selon Martin et al. [99], l'environnement de développement doit aider les concepteurs de plusieurs manières. Il doit encourager la conformité, supporter l'hétérogénéité, offrir du support pour la construction d'une communauté d'agents, aider à l'exécution et au débogage du système et faciliter l'usage d'agents de support. Les concepteurs ont des choix à faire sur les types d'agents ainsi que sur leurs caractéristiques. Le tableau 5.2 montre les plus importants, selon Farhoodi et Graham [45].

CARACTÉRISTIQUES
– degré d'adoption de la notion d'agent
– architecture interne des agents
– granularité des agents
– modèle de distribution
– méthode de communication
– type d'interaction
– technique de partage des connaissances
– méthode de contrôle
– méthode de coordination
– style de coopération

Tableau 5.2 – Choix importants pour la conception de systèmes multi-agents.

Le modèle GAM sert bien la cause des concepteurs concernant les caractéristiques mentionnées dans le tableau 5.2. Les concepteurs sont libres de choisir le degré d'adoption de la notion d'agent. Certains des agents spécialisés de base de l'environnement correspondent tout juste à la définition pratique d'agent et certains, comme l'agent délibératif, se rapprochent de la définition théorique. Il en est de même pour l'architecture interne des agents, car les types d'agents les plus courants sont déjà offerts par l'environnement. Le modèle de distribution des agents est fixé en définissant les connexions appropriées selon le modèle que les concepteurs ont choisi.

L'environnement de développement doit aussi aider les concepteurs d'une manière plus globale concernant l'architecture du système et les capacités de communication et de coordination, deux caractéristiques importantes d'un SMA. Ces caractéristiques sont importantes selon Müller [108], car elles sont reliées aux critères de base pour une approche multi-agents,

une distributivité naturelle, des interactions flexibles et un environnement dynamique. L'environnement de développement doit libérer les concepteurs sur ces points, car ils doivent prendre des décisions et faire des choix sur plusieurs autres sujets à propos des agents. Le tableau 5.3 résume ces choix.

SUJET	CHOIX
tâches simples ou collaboratrices	est-ce qu'il y a des activités parallèles d'exécution de tâches ?
raisonnements ouverts ou connaissances fermées	est-ce que le raisonnement sur d'autres agents doit être inclus ?
actions séquentielles ou parallèles	est-ce que des activités parallèles sont permises ou préférées ?
autarcie ou planification interactive	est-ce que la collaboration est nécessaire pour atteindre un but ?
planification individuelle ou collective	est-ce qu'il y a des activités de planification parallèle pour une tâche globale ?

Tableau 5.3 – Choix importants de conception d'agents [108].

Le modèle GAM et les agents spécialisés de base et évolués sont bien adaptés à la conception de SMA, car les concepteurs sont libres de faire les choix qu'ils désirent. Les agents issus du modèle GAM peuvent être utilisés selon les besoins des concepteurs. Ils peuvent distribuer les tâches selon leurs souhaits et les actions peuvent être séquentielles ou parallèles. L'architecture GAM n'impose pas de restrictions quant au degré de parallélisme à utiliser. Le type de planification ou de collaboration est laissé au choix des concepteurs. Le modèle GAM permet tout type d'organisation architecturale.

Dans le cas où le paradigme choisi ou le type d'agent désiré ne serait pas présent dans l'ensemble des agents offerts par l'environnement, les concepteurs peuvent eux-mêmes concevoir le type d'agent manquant à partir de ceux existants ou à partir de l'agent abstrait. Le nouvel agent peut ensuite être ajouté à l'ensemble de ceux disponibles.

Le modèle GAM, satisfait les considérations importantes de développement de SMA du tableau 5.3. Il peut donc être considéré comme bien adapté pour servir de base à un environnement générique de développement de SMA. Par contre, d'un point de vue pratique, une facette importante du développement de SMA reste à détailler et à approfondir afin d'assu-

rer que le modèle GAM soit bien adapté à tous les niveaux. Cette facette est le débogage de SMA.

5.7 Environnement de développement et débogage

Une des fonctions premières d'un environnement de développement est de faciliter l'implémentation de chaque agent d'un SMA et de faciliter son intégration dans le système. Il doit aussi faciliter la tâche des concepteurs en ce qui concerne la mise au point du SMA lorsque le système est dans des situations de simulations ou des situations réelles. Le débogage est une étape très importante dans le cycle de vie d'un système logiciel, surtout pour des systèmes complexes comme les SMA. Au niveau du développement, les outils de débogage sont une composante importante des outils de diagnostic de systèmes intelligents.

La discussion sur le débogage de SMA à la section 3.7 a permis de distinguer le mode hors-ligne et le mode en-ligne de débogage. La figure 3.6 relative à ces explications est reproduite ici à la figure 5.15. Les outils de débogage doivent supporter ces deux modes pour offrir une flexibilité d'opération aux concepteurs. Ils peuvent alors choisir le mode le plus approprié selon les circonstances. Les points de vue de l'agent et des outils de débogage sont distingués par l'environnement GEMAS.

5.7.1 Point de vue de l'agent

Du point de vue de l'agent, l'environnement GEMAS distingue les états actif et inactif de débogage. Aucune trace n'est générée lorsque l'état est inactif. Lorsque l'état est actif, des traces sont générées et elles peuvent alors être examinées par les observateurs. La séance de débogage peut être hors-ligne ou en-ligne, comme illustré à la figure 5.15. Dans le cas d'une séance hors-ligne, les traces sont locales et elles sont inscrites dans une base de données locale. Dans le cas d'une séance en-ligne, les traces doivent être envoyées à un outil de collection de traces. Dans les deux cas, les concepteurs décident de l'endroit précis où vont les traces.

L'agent peut être en mode statique ou en mode dynamique de débogage. Si le mode est statique, l'état de débogage de l'agent et la destination des traces ne peuvent pas être mo-

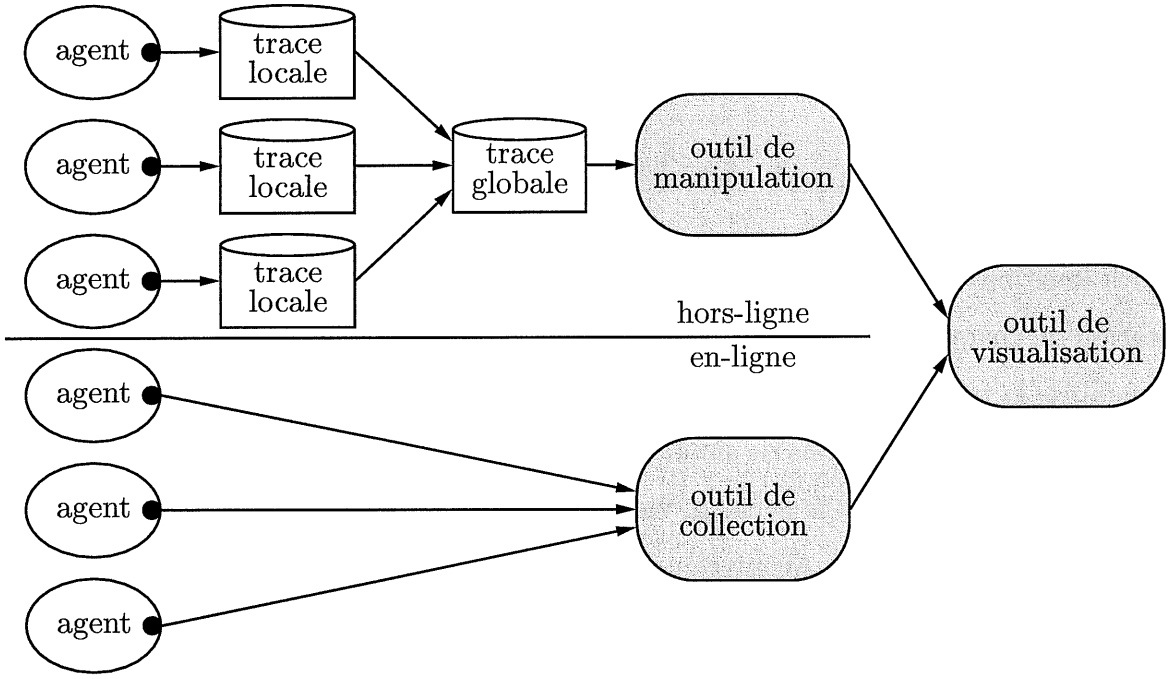


Figure 5.15 – Mode statique de débogage d'un système multi-agents [148].

difiés pendant l'exécution, car ils sont fixés lors de l'instanciation de l'agent. Si le mode est dynamique, l'état de débogage de l'agent peut être changé à tout moment. Dans ce cas, l'état de l'agent et la destination des traces peuvent être modifiés par des messages spécialement dédiés à cet usage. La figure 5.15 montre le débogage hors-ligne et en-ligne en mode statique. Le mode dynamique est illustré à la figure 5.16. La différence principale entre le mode dynamique et le mode statique est que les outils de manipulation et de collection envoient des messages aux agents ; ils ne font plus que recevoir des traces.

Dans une application réelle, le mode dynamique peut causer des problèmes de sécurité ou de confidentialité des données. En effet, des informations internes confidentielles peuvent être divulguées si l'état de débogage est activé par un agent inconnu et que des traces lui sont envoyées. Le mode statique et l'état inactif de débogage sont préférables dans une version finale d'un système. Par contre, dans certaines applications, il peut être désirable de garder le mode dynamique. Dans ce cas, des mécanismes d'encryptage pour le transfert des données et des mécanismes de mots de passe pour les changements de mode de débogage et de lieu des traces pourraient être utilisés.

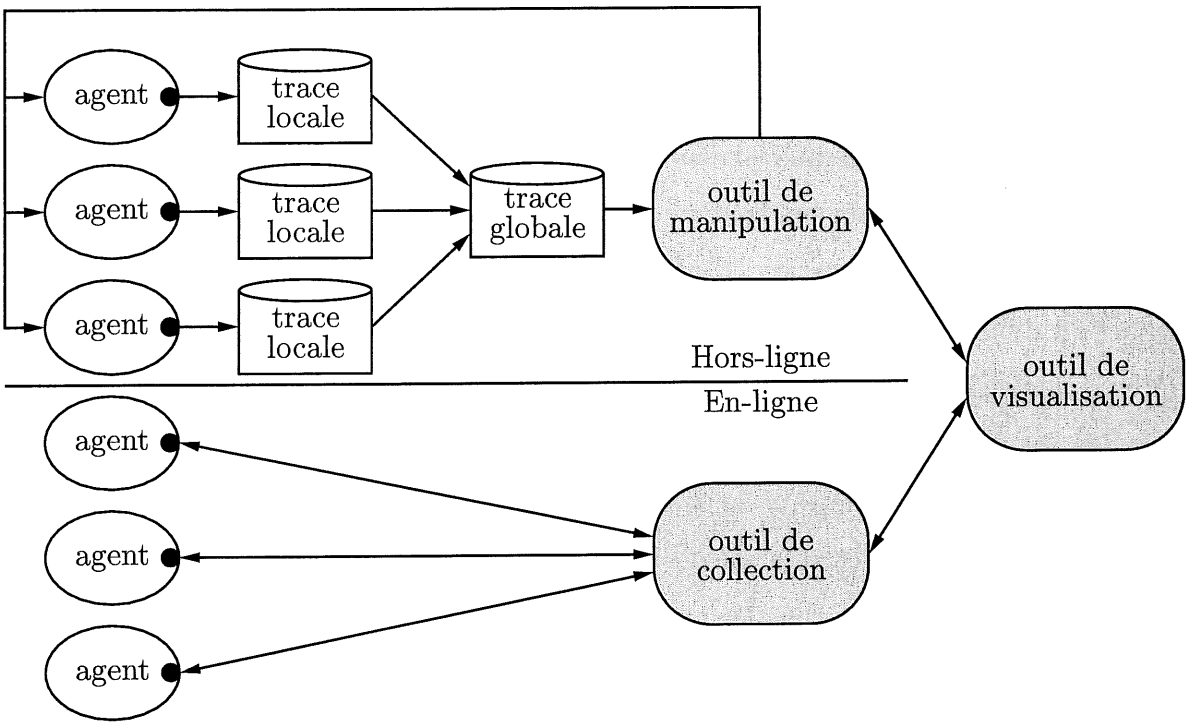


Figure 5.16 – Mode dynamique de débogage d'un système multi-agents.

En ce qui concerne la capacité d'un agent de laisser des traces, différentes possibilités sont offertes selon le niveau du modèle GAM considéré. À chaque niveau, il est possible d'utiliser les fonctionnalités de traçage. Les concepteurs de chacun des niveaux sont libres de décider des messages qui vont apparaître dans les traces et même d'avoir différentes catégories de traces s'ils le désirent.

La capacité de laisser des traces au niveau communication et au niveau interface est fixée par les concepteurs de l'environnement. Ainsi, il est possible d'assurer non seulement l'uniformité des fonctionnalités de communication, mais aussi l'uniformité des capacités de débogage des agents.

De la même manière, au niveau moteur et au niveau connaissance, la capacité de laisser des traces est la responsabilité des concepteurs de ces niveaux. Cette liberté au niveau des traces est une caractéristique importante. Au niveau moteur, elle permet d'uniformiser et de standardiser les fonctionnalités de traces de chaque paradigme. Ainsi, tous les paradigmes peuvent avoir les mêmes capacités de traçage ou ils peuvent avoir leurs propres caractéris-

tiques particulières. Au niveau connaissance, cette liberté permet aux concepteurs de laisser les traces qu'ils désirent pour leurs applications.

Les fonctionnalités ajoutées aux niveaux communication et interface du modèle GAM afin de supporter le mode débogage sont peu nombreuses. Au moment de l'instanciation d'un agent, les niveaux communication et interface doivent pouvoir connaître le mode de l'agent, (statique ou dynamique) le mode de débogage (actif ou inactif) et l'identité de la base de données locale ou de l'agent servant d'outil de collection d'information. Pendant l'exécution, l'agent doit avoir la capacité d'envoyer des traces. En mode dynamique, l'agent doit pouvoir accepter des messages permettant de passer d'un état à l'autre (actif ou inactif) et de changer l'identité de la base de données ou de l'agent recevant les traces. Ces fonctionnalités de débogage sont disponibles à tous les niveaux du modèle GAM afin d'assurer que le contrôle du débogage soit disponible globalement dans l'agent. Les fonctionnalités de débogage sont résumées au tableau 5.4.

FONCTIONNALITÉ
– détection de l'état de débogage (actif ou inactif)
– détection du mode de débogage (statique ou dynamique)
– changement de l'état de débogage (actif ou inactif)
– changement de destination des traces (locales ou distantes)
– génération d'une trace (enregistrement)

Tableau 5.4 – Fonctionnalités supplémentaires des agents pour le débogage.

5.7.2 Point de vue des outils de débogage

Du point de vue de l'environnement de développement, les outils de débogage peuvent être perçus comme des agents supplémentaires au SMA. Les agents constituant le SMA communiquent leurs traces avec des messages. Les traces sont récupérées, en-ligne ou hors-ligne, par des agents adaptés à la présentation de traces. Ces agents de présentation sont, en accord avec les figures 5.15 et 5.16, l'outil de manipulation de traces, l'outil de collection de traces et l'outil de visualisation de traces. Les outils de débogage sont des agents adaptés pour la

manipulation de traces d'agents et aussi capables de manipuler les modes de débogage des agents du SMA.

L'ensemble des outils de débogage doit offrir les fonctionnalités nécessaires afin de manipuler les fonctionnalités de débogage des agents. Dans le mode statique et dynamique, ils permettent de récupérer les traces locales ou les traces en-ligne et de les afficher. Les outils peuvent permettre d'avoir différentes vues conceptuelles des traces, comme expliqué à la section 3.7. Dans le mode dynamique, les outils permettent de commander aux agents de changer l'état de débogage et la destination des traces.

En plus des outils et des fonctionnalités associées aux traces, un mécanisme standardisé d'estampage temporel doit être mis en oeuvre afin de pouvoir ordonner les traces adéquatement. Deux situations peuvent survenir selon que le SMA est distribué ou non sur plusieurs machines. Dans le cas d'un SMA sur une seule machine, l'heure locale du système est suffisante afin d'obtenir une estampille temporelle, car tous les agents ont la même référence de temps. Dans le cas d'un SMA distribué, un mécanisme centralisé d'estampage temporel doit être mis en place, car les différentes machines impliquées dans le SMA ne sont pas nécessairement synchronisées. À ce moment, un agent est dédié à la gestion des estampilles temporelles. Les agents du SMA générant des traces doivent communiquer avec lui pour obtenir des estampilles, comme illustré à la figure 5.17. L'agent d'estampage temporel peut aussi être utilisé dans le cas d'un SMA non-distribué. Le tableau 5.5 résume la liste des fonctionnalités principales des outils de débogage.

FONCTIONNALITÉ
– récupérer les traces
– changer le mode de débogage des agents
– changer la destination des traces
– fournir des estampilles temporelles

Tableau 5.5 – Fonctionnalités principales des outils de débogage.

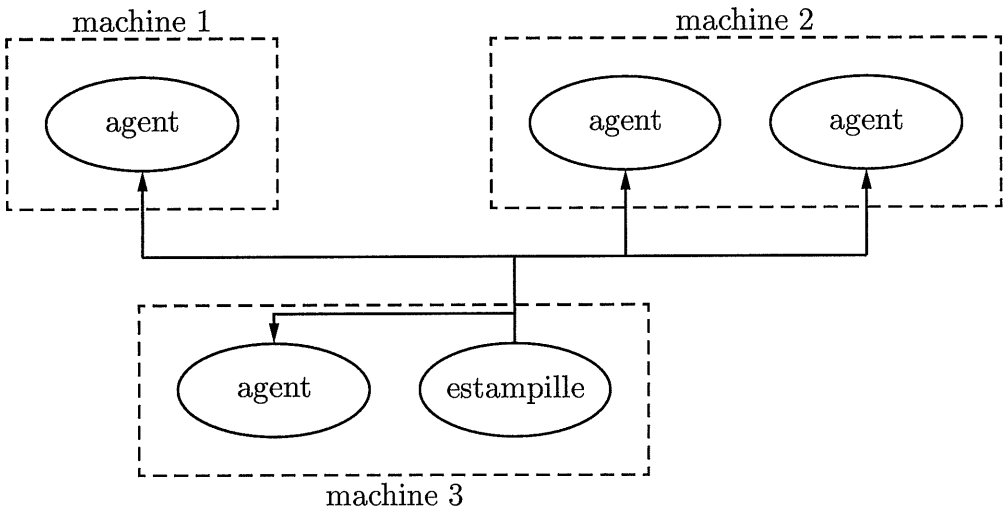


Figure 5.17 – Gestion centralisée d'estampage temporel pour un SMA distribué.

5.8 Discussion

D'un point de vue conceptuel, l'environnement GEMAS est basé sur le modèle GAM, un modèle générique d'agent à quatre niveaux : communication, interface, moteur et connaissance. Les différentes spécialisations du niveau moteur permettent d'obtenir des agents hétérogènes qui sont appelés agents spécialisés de base. Le modèle GAM permet d'obtenir des agents basés sur les paradigmes les plus courants : la logique, la logique floue, les réseaux de neurones artificiels, les algorithmes génétiques et les langages procéduraux.

Le modèle GAM permet d'engendrer des agents hétérogènes en spécialisant le niveau moteur de différentes manières et il permet de les intégrer dans un même système. L'intégration d'un nouvel agent est faite de manière plus uniforme, car tous les agents du système sont basés sur le même modèle. De plus, le modèle GAM assure une bonne portabilité de l'application, car le modèle est générique et indépendant du système d'opération et de la composante matérielle du système. Ces avantages ne sont pas négligeables pour les concepteurs et pour la maintenance.

Les agents spécialisés sont utilisés pour obtenir les agents évolués qui sont couramment rencontrés : agents délibératifs, agents réactifs et agents hybrides. De plus, le modèle GAM permet de prendre en considération le problème très concret qu'est le débogage. Les fonctionnalités nécessaires pour le débogage, phase importante de la durée de vie d'un système

logiciel, est une partie intégrante de la modélisation des agents de l'environnement GEMAS.

Le modèle GAM se distingue des autres modèles d'agents de deux manières. Premièrement, il se distingue par le fait qu'il engendre des agents hétérogènes. Deuxièmement, il est orienté communication et non orienté paradigme. Le modèle GAM est centré sur les communications ; les paradigmes sont ajoutés et instanciés selon les besoins. Ceci est en opposition avec les modèles classiques qui sont centrés sur le paradigme. Cette distinction du modèle GAM par rapport aux autres est ce qui permet de réaliser des agents hétérogènes à partir d'un même modèle conceptuel.

Le modèle GAM ouvre aussi une porte sur une possibilité intéressante, soit des agents à personnalités multiples (APM). Le modèle GAM instancie différemment le paradigme selon le type d'agent et rien n'empêche de le changer dynamiquement à l'exécution pour ainsi changer complètement les niveaux moteur et connaissance. Ceci a comme effet de changer complètement la personnalité de l'agent. Par exemple, un APM peut passer du paradigme logique au paradigme de la logique floue s'il juge que ce paradigme est plus approprié et efficace pour la tâche qu'il doit accomplir. Les APM ouvrent une nouvelle avenue pour la conception d'agents.

D'un point de vue pratique, l'environnement GEMAS est extensible et portable. Il encourage une conception modulaire et favorise donc la maintenance. Les agents spécialisés de base et évolués de l'environnement GEMAS sont indépendants de l'application et ils peuvent être configurés et exploités comme les concepteurs le jugent pertinent. L'architecture du SMA n'est pas imposée par GEMAS ; elle est plutôt fixée par les concepteurs selon l'application et leurs besoins.

L'introduction de ce chapitre mentionne trois défis importants à relever en ce qui concerne le domaine des SMA. Le modèle GAM et l'environnement GEMAS solutionnent directement le besoin d'outils de développement réutilisables et génériques et le besoin d'intégrer des agents hétérogènes, soit les deux premiers défis. Ils ouvrent une voie vers le troisième qui est le rapprochement de la théorie et de la pratique.

Éventuellement, une base commune de développement, comme l'environnement GEMAS, pourrait permettre de rassembler les efforts de recherche sur les SMA. Elle permettrait de cumuler plus rapidement les réalisations sous une même modélisation et ainsi faire avancer le domaine des SMA.

Le modèle GAM et l'environnement GEMAS étant spécifiés, il s'agit maintenant de les valider à l'aide d'une implémentation et d'applications de test. Le chapitre 6 présente cette étape de validation.

CHAPITRE 6

GEMAS : RÉALISATION DE L'ENVIRONNEMENT

Le chapitre 5 a permis d'établir les concepts de base d'un environnement générique de développement de SMA nommé GEMAS. L'implémentation des agents issus du modèle GAM et des outils de l'environnement GEMAS est le sujet de ce chapitre. Des applications de test ont été réalisées afin de confirmer la validité des concepts et des implémentations qui sont proposés.

Le chapitre 5 a aussi fixé la structure des agents qui composent l'environnement GEMAS et il a aussi identifié les outils qui en font partie. Il s'agit maintenant de les implémenter dans le but d'obtenir un prototype de l'environnement GEMAS.

6.1 Introduction

L'analyse qui a été faite au chapitre 5 identifie ce qui doit être implémenté afin de réaliser l'environnement GEMAS : l'agent abstrait, les agents spécialisés de base et les outils de débogage. Les agents spécialisés évolués ne sont pas réalisés, car ils ne sont pas nécessaires pour démontrer la validité de l'environnement GEMAS. Ils sont une extension ou une composition des agents spécialisés de base.

Des choix doivent être faits afin de pouvoir procéder à l'implémentation : quelle approche de conception utiliser ? Quel langage de programmation choisir ? Quelle plate-forme de développement utiliser ?

Choix de l'approche de conception logicielle

L'approche de conception logicielle favorisée doit être bien adaptée à l'architecture à niveaux du modèle GAM ; elle doit permettre de bien encapsuler les mécanismes de chaque niveau. Elle doit permettre de facilement étendre et augmenter les fonctionnalités à chacun des niveaux. Parmi les approches existantes de conception de logiciels, l'approche orientée objet est bien adaptée pour ce type de développement. L'approche orientée objet se prête naturellement à l'encapsulation, à une architecture hiérarchique et à la spécialisation. La composition et l'héritage sont bien adaptés pour l'extension d'outils de base et pour une architecture logicielle hiérarchique comme le modèle GAM. Le formalisme utilisé dans cette thèse pour les représentations de la conception orientée objet est UML [14, 127].

Choix du langage de programmation

La conception orientée objet suggère le choix d'un langage orienté objet. Le langage choisi doit maximiser la portabilité, la capacité de communication, et aussi permettre de prendre avantage du parallélisme. Le langage C++ [140] et le langage JAVA [6] sont deux candidats de choix.

Le langage retenu est JAVA. La raison principale est que toutes les caractéristiques souhaitées sont une partie intégrante standardisée du langage et, en plus, il est totalement portable sur différents systèmes d'opération. Ceci augmente la portabilité de l'environnement, un critère important. De plus, contrairement au C++, les interfaces graphiques sont aussi une partie standardisée du langage. Les interfaces graphiques sont nécessaires pour les outils de débogage de l'environnement et pour des agents qui interagissent avec des humains. Le langage JAVA assure une meilleure portabilité de l'environnement tout en assurant une implémentation qui se rapproche de la conception.

De plus, si dans certaines situations le C++ est plus avantageux, il peut être utilisé, car le langage JAVA prévoit une interface standardisée avec le C++. Cela représente un autre avantage que le C++ ne peut pas offrir, car le C++ n'a pas d'interface standard avec JAVA. JAVA offre donc le meilleur des deux mondes et c'est pourquoi il est le langage retenu. Aussi,

si des considérations de temps réel surviennent, des appels au langage C++ peuvent être utilisés ou encore les extensions temps réel du langage JAVA [12].

Choix de la plate-forme de développement

Le choix du langage étant JAVA, différentes plates-formes de développement sont disponibles, dont celles basées sur les systèmes d'opération UNIX, Microsoft Windows et MacOS. Celle choisie doit supporter un environnement complet de développement JAVA. Le choix d'une plate-forme est un choix uniquement pratique et non conceptuel, car le langage JAVA est complètement portable sur toutes ces plates-formes. L'accessibilité aux ressources, la disponibilité immédiate des outils de développement et la possibilité d'exploiter le parallélisme sont les critères les plus importants. La plate-forme UNIX est celle retenue, car elle est la seule qui répond à tous les critères dans le contexte actuel de cette recherche.

Maintenant que les paramètres globaux de la réalisation sont fixés, le reste du chapitre est dédié à la réalisation. Le chapitre est organisé selon l'architecture du modèle GAM et de l'environnement GEMAS. Il procède du plus bas niveau vers le plus haut. La section 6.2 expose la conception de l'agent abstrait du modèle GAM. La section 6.3 discute du concept d'agent avec paradigme. Les sections 6.4 à 6.8 montrent comment les agents spécialisés de base sont conçus et réalisés à partir de l'agent avec paradigme. Les outils de débogage sont abordés à la section 6.9. La section 6.10 permet de discuter des réalisations effectuées et des résultats obtenus des applications de test.

6.2 Agent abstrait

Les sections 5.3 et 5.7 définissent les fonctionnalités principales de l'agent abstrait ainsi que ses fonctionnalités de débogage ; elles sont listées dans les tableaux 5.1 et 5.4. La conception et la réalisation de l'agent abstrait doivent respecter et intégrer toutes ces fonctionnalités.

La section 5.3 identifie que l'agent abstrait sert de fondation pour la conception et la réalisation des agents spécialisés de base. Il ne peut pas être instancié, car seuls les niveaux

communication et interface sont présents. L'agent abstrait sert de classe de base aux agents spécialisés.

L'agent abstrait est situé au niveau interface du modèle GAM et il offre aux niveaux supérieurs un accès aux fonctionnalités des niveaux inférieurs. L'agent abstrait offre une interface qui cache les détails sous-jacents à la communication. La figure 6.1 illustre le diagramme de classes de l'agent abstrait.

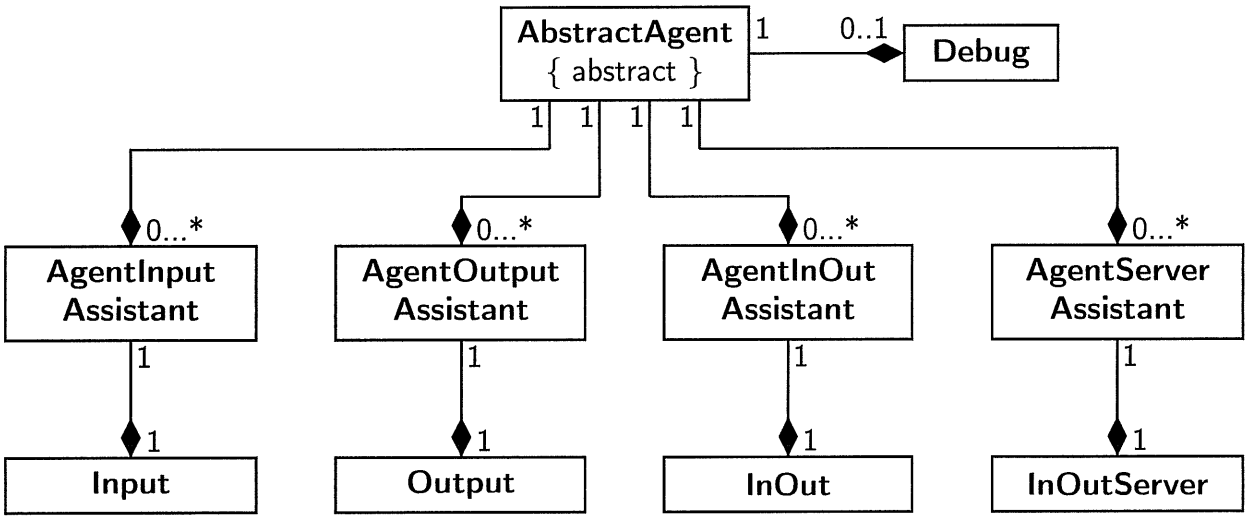


Figure 6.1 – Diagramme de classes de l'agent abstrait.

L'agent abstrait est implémenté par la classe abstraite[1] nommée AbstractAgent et, comme le prescrit le modèle GAM, elle ne peut pas être instanciée, car deux de ses méthodes sont abstraites. Une d'elles permet d'instancier le paradigme et l'autre permet de lancer l'exécution du paradigme. Ces deux méthodes abstraites sont implémentées par les classes dérivées de la classe AbstractAgent, car ce sont elles qui doivent spécifier le paradigme et la manière de lancer son exécution. Au niveau moteur, ces classes dérivées sont celles qui implémentent les agents spécialisés de base.

La classe AbstractAgent possède toutes les fonctionnalités permettant de gérer les liens de communication de l'agent. Ces fonctionnalités sont divisées en deux catégories :

[1]Une description détaillée de toutes les classes est disponible dans la documentation en-ligne de l'environnement GEMAS.

Catégorie ajout d'un lien : Cette catégorie de fonctionnalités permet d'ajouter un lien de communication à ceux qui sont déjà gérés par l'agent. Le lien ajouté peut être en entrée (classe Input), en sortie (classe Output) ou les deux (classe InOut). Un **assistant de lien** gère la réception et l'envoi des messages associés au lien ajouté. Les assistants de liens sont expliqués un peu plus loin.

Catégorie retrait d'un lien : Cette catégorie de fonctionnalités permet de retirer un lien de communication de la liste des liens gérés par l'agent. Le lien peut être coupé lorsqu'il est retiré.

L'agent abstrait comporte aussi des fonctionnalités qui permettent de gérer des demandes externes de connexions ; l'agent abstrait a les capacités d'agir comme serveur de liens. De manière identique aux liens de communication, deux catégories de fonctionnalités de type serveur sont disponibles :

Catégorie ajout d'un serveur : Cette catégorie de fonctionnalités permet d'ajouter un serveur de liens (classe InOutServer) à ceux qui sont déjà gérés par l'agent. Comme avec la catégorie ajout d'un lien, un assistant est associé au serveur afin de gérer les nouveaux liens acceptés et créés.

Catégorie retrait d'un serveur : Cette catégorie de fonctionnalités permet de retirer un serveur de la liste des serveurs qui sont gérés par l'agent. Le serveur de lien peut être stoppé lorsqu'il est retiré.

L'agent abstrait possède aussi les fonctionnalités de base de débogage qui sont identifiées à la section 5.7. Ces fonctionnalités de débogage sont rendues disponibles à l'aide de la classe Debug qui permet de générer les traces, de gérer le mode et le statut de débogage et d'identifier l'agent d'estampage temporel. L'agent abstrait peut créer et détruire un objet de type Debug et ainsi contrôler ses capacités de débogage. L'agent abstrait possède un seul objet de type Debug à la fois.

L'agent abstrait peut entretenir plusieurs liens de communication simultanément. Des messages peuvent être reçus et envoyés de manière asynchrone et concurrente. Ceci est accompli en ayant un *thread* pour chaque lien de communication. Les assistants de liens assurent

une gestion sécuritaire de la réception et de l'envoi de messages dans un environnement *multi-threaded*. De la même manière, plusieurs serveurs de liens peuvent être entretenus.

Les assistants de liens sont des intermédiaires qui gèrent l'envoi et la réception de messages. Ils sont des gestionnaires spécialisés de listes de messages. La figure 6.2 illustre leur fonctionnement pour la réception et l'envoi de messages.

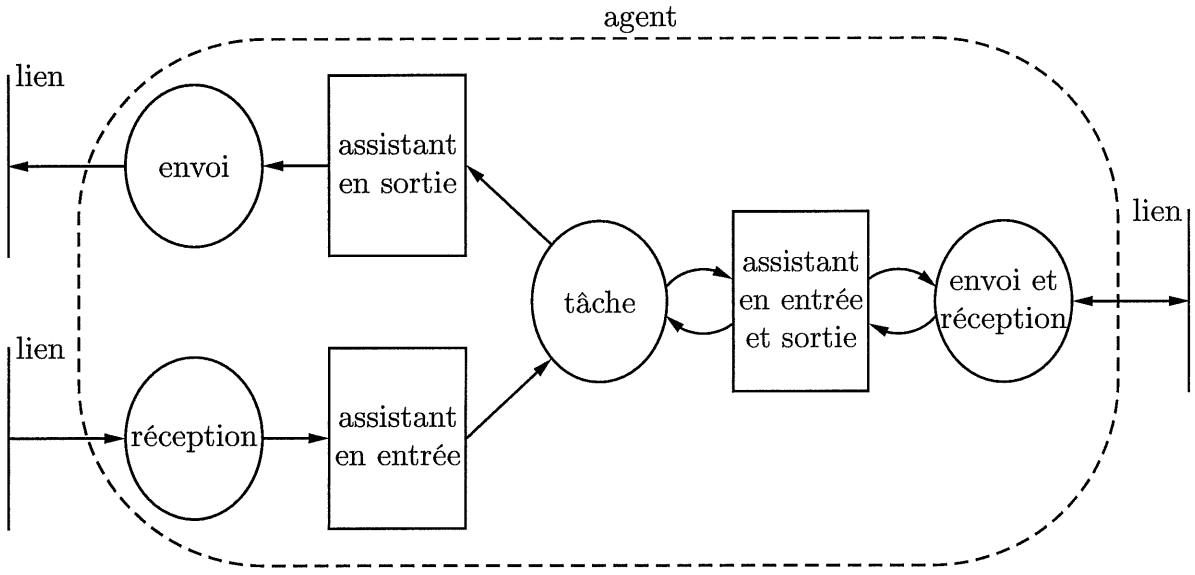


Figure 6.2 – Fonctionnement d'assistants de liens en entrée et en sortie.

Le fonctionnement de l'assistant en entrée est le suivant. Le *thread* réception reçoit un message sur le lien de communication qui lui est associé et l'ajoute dans sa liste de messages reçus. Lorsqu'il le juge nécessaire, le *thread* tâche récupère ce message en sollicitant l'assistant. L'assistant assure le traitement correct des accès concurrentiels à la liste de messages de la part des *threads* réception et tâche.

Le fonctionnement de l'assistant en sortie est similaire à celui en entrée. Le *thread* tâche ajoute un message dans la liste gérée par l'assistant en sortie. Le *thread* envoi récupère les messages à envoyer de l'assistant en sortie et les transmet sur le lien qui lui est associé.

En plus des assistants en entrée et des assistants en sortie, il existe aussi un assistant pour les liens de communication qui sont en entrée-sortie. Cet assistant fait la gestion interne de deux listes de messages, une liste pour les messages en entrée et une liste pour les messages en sortie. Aussi, un assistant pour les fonctionnalités de serveur est défini. Cet assistant gère

une liste de nouvelles connexions en entrée-sortie qui ont été acceptées et qui doivent être prises en charge. Les classes AgentInputAssistant, AgentOutputAssistant, AgentInOutAssistant et AgentServerAssistant réalisent les assistants qui sont utilisés par l'agent abstrait.

L'agent abstrait cache aux niveaux supérieurs les détails et les problèmes de gestion des communications et de débogage à l'aide d'une interface de haut niveau et d'assistants. Un ensemble de classes supporte l'agent abstrait dans ses fonctions ; elles se situent au niveau interface et au niveau communication du modèle GAM. Ces classes de support sont expliquées dans les sections suivantes.

6.2.1 Niveau interface

Le niveau interface, comme indiqué à la section 5.2, permet de découpler les fonctionnalités de base du niveau communication des niveaux supérieurs et il ajoute des fonctionnalités de communication plus évoluées. Afin de respecter cette philosophie, les classes du niveau interface encapsulent les classes du niveau communication et ajoutent à leurs fonctionnalités. La figure 6.3 illustre le diagramme de classes du niveau interface.

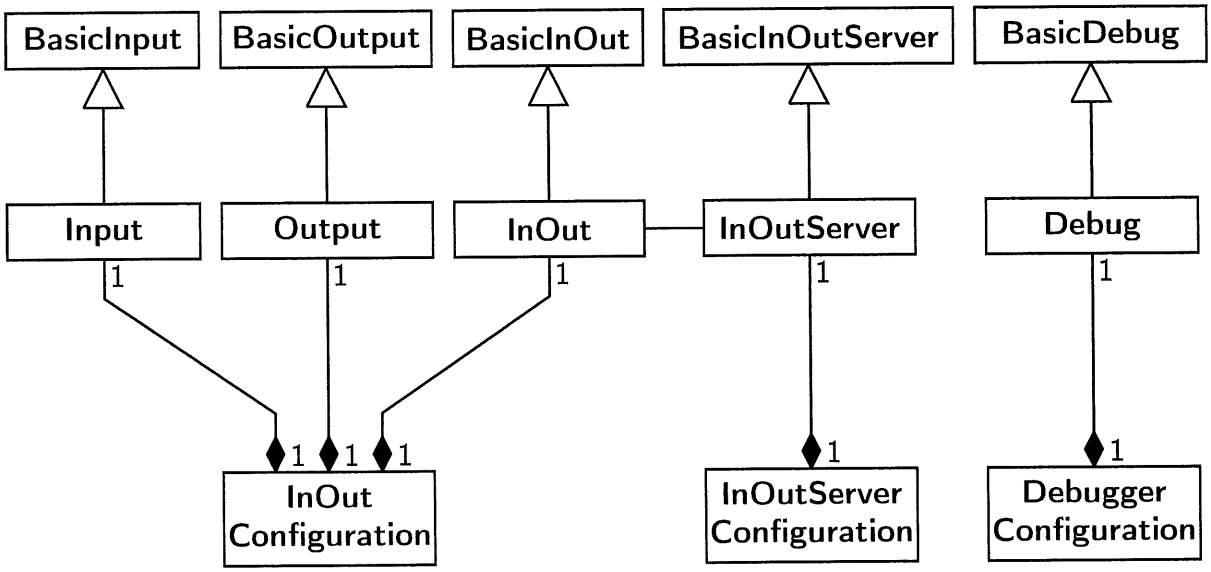


Figure 6.3 – Diagramme de classes du niveau interface.

Les classes qui composent le coeur du niveau interface permettent de créer des liens de communication, de créer des serveurs de liens et de fournir des fonctionnalités de débogage.

Voici une description de chacune de ces classes :

Classe Input : Cette classe permet de créer un lien de communication en entrée. La configuration du lien est spécifiée à l'aide de la classe InOutConfiguration. La classe Input permet à un agent de recevoir des messages sur ce lien. Elle est une extension de la classe BasicInput du niveau communication.

Classe Output : Cette classe permet de créer un lien de communication en sortie. La configuration du lien est spécifiée à l'aide de la classe InOutConfiguration. Elle permet à un agent d'envoyer des messages sur ce lien. Elle est une extension de la classe BasicOutput du niveau communication.

Classe InOut : Cette classe permet de créer un lien de communication en entrée-sortie. La configuration du lien est spécifiée à l'aide de la classe InOutConfiguration. Elle permet à un agent d'envoyer et de recevoir des messages sur le même lien. Elle est une extension de la classe BasicInOut du niveau communication. De manière conceptuelle, la classe InOut peut être vue comme une combinaison des classes Input et Output.

Classe InOutServer : Cette classe permet de créer un serveur de liens de communication. La configuration du serveur est spécifiée à l'aide de la classe InOutServerConfiguration. Les liens acceptés et créés sont en entrée-sortie (classe InOut). Elle est une extension de la classe BasicInOutServer du niveau communication.

Classe Debug : Cette classe permet de créer un objet fournissant les capacités de débogage de l'agent abstrait. La configuration du débogueur est spécifiée à l'aide de la classe DebugConfiguration. Elle est une extension de la classe BasicDebug du niveau communication.

Une instance des classes Input, Output ou InOut est créée lorsqu'un agent doit initier une communication. Ensuite, à l'aide d'un assistant, il peut envoyer ou recevoir des messages selon le type de lien créé.

Les classes du niveau interface offrent des fonctionnalités plus évoluées que celles du niveau communication et elles servent d'intermédiaire aux niveaux supérieurs du modèle GAM.

La plupart des fonctionnalités des classes de ce niveau sont basées sur celles du niveau communication.

6.2.2 Niveau communication

Le niveau communication est composé de classes qui implémentent les fonctionnalités de base de communication et de débogage. Les classes sont divisées en deux catégories, la catégorie communication et la catégorie débogage. La catégorie communication est illustrée à la figure 6.4 et la figure 6.5 illustre la catégorie débogage.

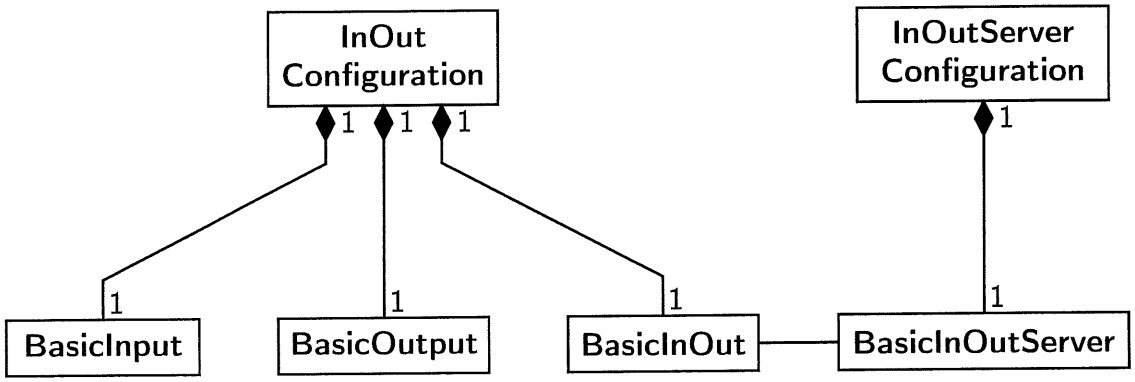


Figure 6.4 – Diagramme de classes du niveau communication.

Parmi les classes de la figure 6.4, les classes BasicInput, BasicOutput, BasicInOut sont celles qui permettent de créer des liens de communication. La classe BasicInOutServer permet de créer un serveur de liens. Afin d'assurer l'indépendance des niveaux supérieurs des niveaux inférieurs, ces classes sont instanciées et manipulées uniquement par les classes du niveau interface et ne doivent pas être manipulées directement par les niveaux supérieurs. Voici une description de ces classes :

Classe BasicInput : Cette classe permet de créer un lien de communication en entrée ; elle permet à un agent de recevoir des messages sur ce lien. La configuration du lien est spécifiée à l'aide de la classe InOutConfiguration.

Classe BasicOutput : Cette classe permet de créer un lien de communication en sortie ; elle permet à un agent d'envoyer des messages sur ce lien. La configuration du lien est spécifiée à l'aide de la classe InOutConfiguration.

Classe BasicInOut : Cette classe permet de créer un lien de communication en entrée-sortie ; elle permet à un agent d'envoyer et de recevoir des messages sur ce lien. La configuration du lien est spécifiée à l'aide de la classe InOutConfiguration.

Classe BasicInOutServer : Cette classe permet de créer un serveur de liens de communication. La configuration du serveur est spécifiée à l'aide de la classe InOutServerConfiguration. Les liens acceptés et créés sont en entrée-sortie (classe BasicInOut).

Les capacités de débogage d'un agent sont réalisées par la classe BasicDebug et, tout comme les classes précédentes, elle n'est instanciée et manipulée directement que par les classes du niveau interface.

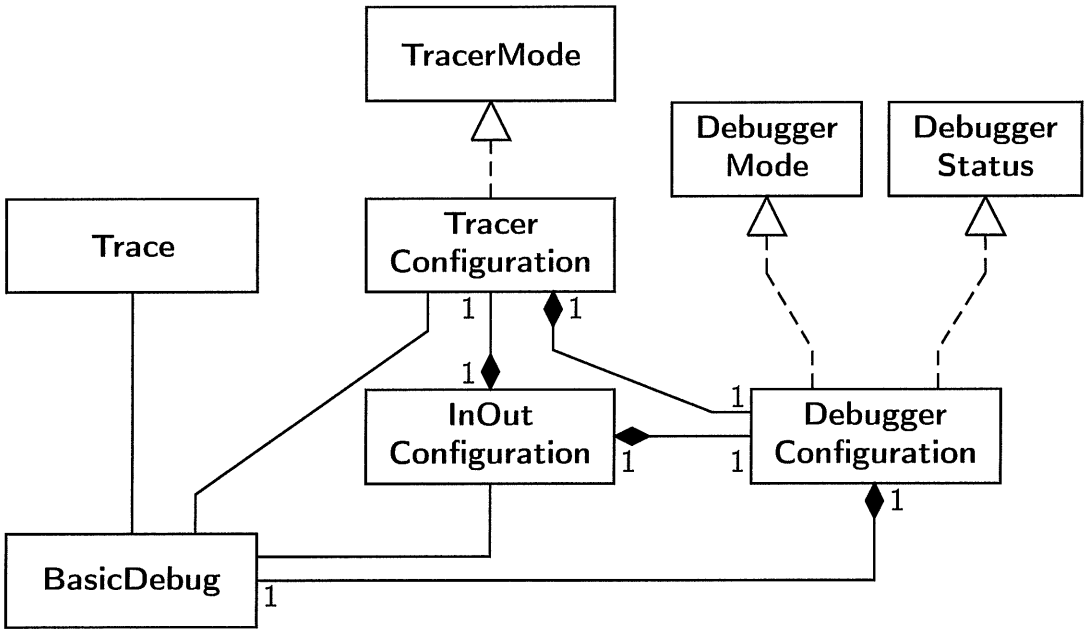


Figure 6.5 – Diagramme de classes des fonctionnalités de débogage.

Lorsqu'un agent requiert des capacités de débogage, une instance de la classe BasicDebug est créée en spécifiant ses caractéristiques à l'aide de la classe DebuggerConfiguration. À ce moment, des traces (classe Trace) peuvent être laissées par l'agent.

La configuration du débogueur permet de spécifier quel est le mode de débogage (classe DebuggerMode) et quel est le statut de débogage (classe DebuggerStatus). L'endroit où les traces sont enregistrées est spécifié par la classe TracerConfiguration qui indique le mode du

traceur (classe TracerMode) et la destination des traces (classe InOutConfiguration). Aussi, si un agent d'estampage temporel est utilisé, la configuration de la communication avec cet agent est spécifiée à l'aide de la classe InOutConfiguration.

6.2.3 Remarques

Cette section a présenté les interrelations et les principes du fonctionnement des classes des niveaux communication et interface du modèle GAM. Le fonctionnement interne des classes et leurs détails d'implémentation n'est pas abordé ici. Par exemple, si le lien a été coupé, la lecture ou l'écriture d'un message peut provoquer une erreur. Les exceptions levées et beaucoup d'autres détails de réalisation ne sont intentionnellement pas discutés. La documentation en-ligne et le code source de l'environnement GEMAS permettent d'obtenir plus de détails sur ces sujets.

Les classes qui composent les niveaux interface et communication implémentent toutes les fonctionnalités de communication et de gestion nécessaires à l'agent abstrait. Ces fonctionnalités sont divisées dans les niveaux communication et interface, comme spécifié aux sections 5.2 et 5.3.

Les classes des niveaux communication et interface sont facilement extensibles afin de favoriser l'extensibilité de l'environnement GEMAS. La hiérarchie des classes des niveaux communication et interface et le modèle d'agent abstrait sont basés sur les mécanismes d'interface du langage JAVA afin d'assurer cette extensibilité. Les figures 6.3, 6.4 et 6.5 sont en fait une implémentation particulière du contrat général spécifié par les interfaces JAVA de l'environnement GEMAS. Toutes les classes qui implémentent une interface peuvent non seulement être étendues ou redéfinies par le mécanisme d'héritage et ainsi étendre l'environnement GEMAS, mais une implémentation complètement différente des interfaces peut aussi être faite sans que l'agent abstrait ou les niveaux supérieurs en soient affectés. Ces caractéristiques facilitent la maintenance et la portabilité de l'environnement GEMAS.

6.3 Agent avec paradigme

Les agents du niveau moteur du modèle GAM possèdent un paradigme ; ce concept est réalisé explicitement par l'agent avec paradigme. L'agent avec paradigme est dérivé de l'agent abstrait. Il est implémenté par la classe ParadigmedAgent qui est dérivée de la classe AbstractAgent du niveau interface. Cette hiérarchie est illustrée à la figure 6.6.

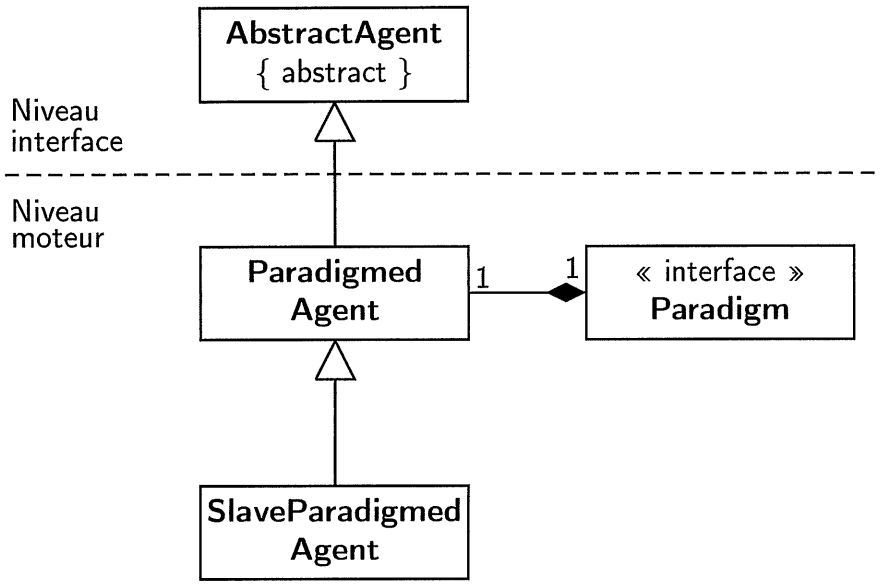


Figure 6.6 – Diagramme de classes d'un agent avec paradigme.

Les agents spécialisés de la section 5.4 sont tous réalisés à l'aide de l'agent avec paradigme. Les paradigmes de base (logique, logique floue, réseaux de neurones artificiels et algorithmes génétiques) réalisent les fonctionnalités spécifiées par l'interface Paradigm. Le cas particulier de l'agent procédural est discuté plus loin.

L'interface Paradigm spécifie les fonctionnalités minimales que tout paradigme doit réaliser. En généralisant les fonctionnalités de base d'un paradigme, celui-ci devient une tâche interne bien définie de l'agent et les communications peuvent alors se faire de manière uniforme avec l'aide d'assistants.

Plus précisément, l'interface Paradigm spécifie des fonctionnalités de base afin que tout paradigme puisse être manipulé de manière standardisée. Des manipulations sont, par exemple,

le passage de données, la récupération de résultats, le démarrage et l'arrêt. Le format du niveau connaissance est spécifié par le paradigme, car les connaissances sont représentées différemment pour chacun des paradigmes. Par exemple, la logique floue n'a pas la même représentation que les réseaux de neurones.

L'environnement GEMAS comporte l'agent avec paradigme non-esclave (classe ParadigmedAgent) et l'agent avec paradigme esclave (classe SlaveParadigmedAgent). Ces deux types d'agents se distinguent par la manière de traiter les communications.

L'agent avec paradigme non-esclave traite les communications explicitement avec les fonctionnalités offertes par l'agent abstrait. Il offre un maximum de contrôle aux concepteurs sur la gestion des communications de l'agent. L'agent avec paradigme esclave spécifie complètement ses communications lors de son instanciation. Une fois instancié, les communications ne peuvent plus être changées. C'est dans ce sens qu'il est esclave. L'agent esclave règle le problème de communication pour les paradigmes qui ne prévoyaient pas les communications avec d'autres agents.

La manière dont chaque paradigme exploite les connaissances liées aux communications est expliquée dans les sections suivantes. La syntaxe des connaissances liées aux communications est donnée dans la documentation en ligne de la classe SlaveParadigmedAgent et des exemples d'utilisation sont disponibles dans les applications de test des agents spécialisés.

Le niveau connaissance d'un agent est spécifié à l'aide d'un **fichier de connaissances** et cela peu importe le paradigme utilisé et si l'agent est esclave ou non. Le fichier de connaissances contient toutes les informations concernant l'exploitation du paradigme, et dans le cas d'un agent esclave, il contient ceux des communications. Le format du fichier de connaissances est particulier à chaque paradigme. Le contenu du fichier de connaissances est discuté en détail un peu plus loin dans les sections décrivant chacun des agents spécialisés.

Comme expliqué à la section 5.4.5, l'agent procédural est un cas particulier. Contrairement aux autres agents spécialisés, il n'a pas de paradigme à instancier, car il l'est déjà. Dans

l'environnement GEMAS, le modèle d'agent avec paradigme ne s'applique pas a l'agent procédural. La section 6.8 discute de ce type d'agent spécialisé avec plus de détails.

En définissant et réalisant les agents avec paradigme esclave et non-esclave, la réalisation des agents spécialisés consiste uniquement en l'implémentation de leur paradigme et de la spécification respective de leur niveau connaissance.

6.4 Agent logique

L'agent logique est un agent basé sur un langage logique comme Prolog. Le paradigme logique doit être implémenté et son niveau connaissance doit être défini afin de réaliser un tel type d'agent.

6.4.1 Conception

Le paradigme logique implémenté utilise le langage Prolog. Le paradigme comporte le moteur d'inférence Prolog, les primitives de base du langage et la définition du niveau connaissance. Afin de respecter le modèle GAM, le niveau moteur de l'agent logique est composé du moteur d'inférence Prolog et des primitives du langage, et le niveau connaissance est composé de l'exploitation du paradigme, c'est-à-dire du code Prolog et de la spécification des communications.

Comme le prescrit le modèle d'agent avec paradigme, la conception de l'agent logique est basée sur la classe ParadigmedAgent, et pour l'agent logique esclave, sur la classe SlaveParadigmedAgent. La classe implémentant le paradigme logique est nommée LogicParadigm. Le diagramme de classes de l'agent logique est illustré à la figure 6.7. Voici le rôle de chacune de ces classes :

Classe LogicAgent : Cette classe offre un agent logique prêt à être utilisé seul ou prêt à être intégré dans un SMA.

Classe SlaveLogicAgent : Cette classe est semblable à la classe LogicAgent, mais l'agent logique est esclave.

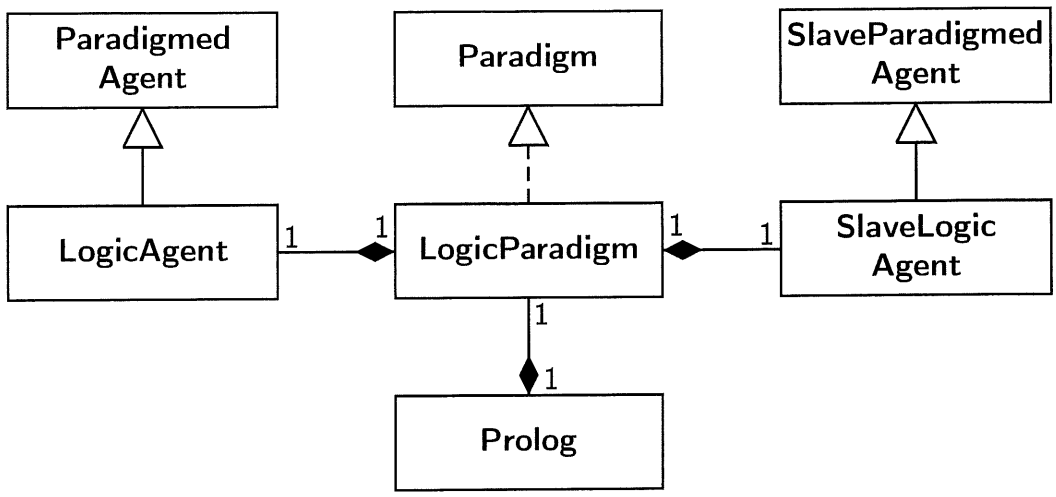


Figure 6.7 – Diagramme de classes du niveau moteur de l'agent logique.

Classe LogicParadigm : Cette classe implémente le paradigme logique. C'est elle qui manipule le moteur d'inférence Prolog. Elle définit aussi le niveau connaissance de l'agent.

Classe Prolog : Cette classe implémente le moteur d'inférence Prolog et ses fonctionnalités de base.

6.4.2 Réalisation

Niveau moteur

Le moteur d'inférence Prolog est basé sur celui présenté dans [147]. Plus précisément, il s'agit d'une adaptation de celui utilisé dans le compilateur COP [21, 22] qui à été traduit du langage C++ au langage JAVA.

Lorsqu'un lien de communication entre l'agent logique et un autre agent est établi, les échanges d'information se passent de la manière suivante. Un agent soumet une requête à l'agent logique. Cette requête de traitement est ajoutée dans la liste des requêtes. Lorsque la requête est traitée, l'agent est informé des résultats de la requête soumise.

Les messages échangés entre l'agent logique et les autres agents sont des chaînes de caractères. Les requêtes de traitement et les résultats en langage Prolog sont représentables par des chaînes de caractères.

Niveau connaissance

Le niveau connaissance de l'agent logique est spécifié par un fichier de connaissances qui spécifie complètement l'exploitation du paradigme et les communications. Lors de l'instanciation de l'agent logique, le fichier de connaissances spécifie le code Prolog, les requêtes initiales, et si l'agent est esclave, les communications. Les connaissances sont spécifiées à l'aide de deux types de blocs de connaissances :

Type codeFiles : Bloc de connaissances spécifiant l'exploitation du paradigme logique. Il spécifie tous les fichiers de code Prolog nécessaires à l'instanciation de l'agent.

Type interpreterInput : Bloc de connaissances optionnel qui permet de soumettre des requêtes initiales à l'interpréteur Prolog lors de l'instanciation de l'agent. Ces requêtes seront exécutées avant toutes requêtes externes de traitement.

Si l'agent logique est en mode esclave, le fichier de connaissances contient alors des blocs de connaissances spécifiant les communications. Les blocs de connaissances liés aux communications sont de quatre types :

Type inputLink : Bloc de connaissances spécifiant complètement les caractéristiques d'un lien en entrée qui doit être créé et géré par l'agent lors de son instanciation.

Type outputLink : Bloc de connaissances spécifiant complètement les caractéristiques d'un lien en sortie qui doit être créé et géré par l'agent lors de son instanciation.

Type inOutLink : Bloc de connaissances spécifiant complètement les caractéristiques d'un lien en entrée-sortie qui doit être créé et géré par l'agent lors de son instanciation.

Type linkServer : Bloc de connaissances spécifiant complètement les caractéristiques d'un serveur de liens qui doit être créé et géré par l'agent lors de son instanciation.

La documentation en ligne de la classe LogicParadigm donne une description complète de la syntaxe et de la sémantique du fichier de connaissances. Elle donne aussi, dans la classe SlaveParadigmedAgent, une description complète de la syntaxe et de la sémantique des blocs de connaissances liés aux communications.

L'agent logique, esclave ou non, est maintenant spécifié aux quatre niveaux du modèle GAM. Une application de test à été réalisée afin de valider la conception et la réalisation de l'agent logique.

6.4.3 Application de test

L'application de test réalisée permet de valider l'agent logique en mode esclave ou non et évoluant seul ou dans un SMA. L'application choisie porte sur les ancêtres. Elle est un exemple classique de la littérature portant sur le langage Prolog.

L'application réalisée a permis de vérifier le bon fonctionnement du niveau moteur de l'agent logique en mode normal et esclave et dans toutes les configurations de communications. L'agent logique est sollicité par d'autres agents avec des requêtes de traitement et il communique des résultats corrects aux agents concernés.

6.5 Agent flou

L'agent flou est un agent basé sur le paradigme de la logique floue. Il faut procéder de manière similaire à l'agent logique afin de réaliser le paradigme de cet agent.

6.5.1 Conception

Le paradigme de la logique floue requiert la définition et la réalisation du contrôleur flou. Les concepteurs doivent spécifier les fonctions d'appartenance et les règles floues. Afin de respecter le modèle GAM, le niveau moteur est composé du contrôleur flou et le niveau connaissance de l'exploitation du paradigme, c'est-à-dire, des règles floues, des fonctions d'appartenance et de la spécification des communications.

La conception de l'agent flou, comme le prescrit le modèle d'agent avec paradigme, est basée sur la classe ParadigmedAgent, et pour l'agent flou esclave, elle est basée sur la classe SlaveParadigmedAgent. La classe implémentant le paradigme flou est nommée FuzzyLogicParadigm. Le diagramme de classes de l'agent flou est illustré à la figure 6.8. Voici le rôle de chacune de ces classes :

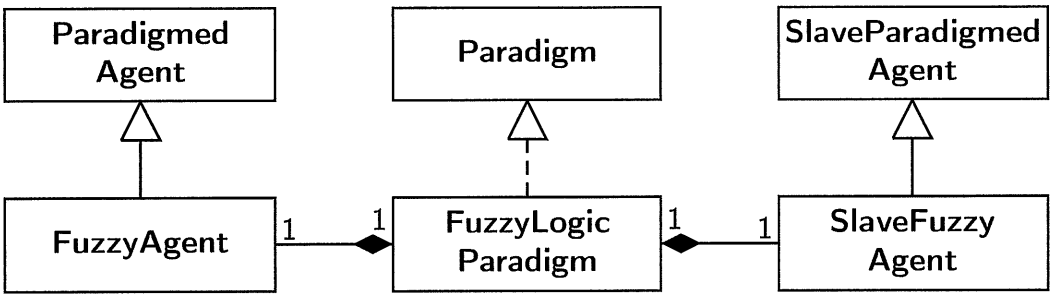


Figure 6.8 – Diagramme de classes du niveau moteur de l'agent flou.

Classe FuzzyAgent : Cette classe offre un agent flou prêt à être utilisé seul ou prêt à être intégré dans un SMA.

Classe SlaveFuzzyAgent : Cette classe est semblable à la classe FuzzyAgent, mais l'agent flou est esclave.

Classe FuzzyLogicParadigm : Cette classe implémente le paradigme de la logique floue. C'est elle qui manipule le contrôleur flou. Elle définit aussi le niveau connaissance de l'agent flou.

6.5.2 Réalisation

Niveau moteur

Le contrôleur flou est basé sur les idées et concepts présentés dans [35, 157]. Le contrôleur flou est réalisé avec le langage JAVA.

Lorsqu'un lien de communication entre l'agent flou et un autre agent est établi, les échanges d'informations se passent de la manière suivante. Un agent soumet de nouvelles valeurs en entrée. L'agent flou ajuste alors ses entrées en accord avec celles reçues. Les sorties du système sont calculées et les nouveaux résultats sont envoyés aux agents intéressés. Les entrées et les sorties sont de type FuzzyLogicData.

Les messages échangés entre l'agent flou et les autres agents sont du type FuzzyLogicData. Ce type de donnée permet de représenter complètement les entrées et les sorties d'un système basé sur la logique floue.

Niveau connaissance

Le niveau connaissance de l'agent flou, tout comme l'agent logique, est spécifié par un fichier de connaissances. Ce fichier spécifie complètement l'exploitation du paradigme et les communications dans le cas d'un agent esclave. Lors de l'instanciation de l'agent flou, le fichier de connaissances permet de spécifier les définitions des fonctions d'appartenance, les règles floues, la configuration du contrôleur flou, et si l'agent est esclave, les communications. Les connaissances sont spécifiées à l'aide de trois types de blocs de connaissances :

Type fuzzySets : Bloc de connaissances spécifiant les fonctions d'appartenance du système.

Type fuzzyRules : Bloc de connaissances spécifiant les règles floues faisant partie du système.

Type configuration : Bloc de connaissances optionnel qui permet de spécifier certains paramètres utilisés par le contrôleur flou, comme la méthode de défuzzification.

La documentation en ligne de la classe FuzzyLogicParadigm donne une description complète de la syntaxe et de la sémantique d'un fichier de connaissances. Si l'agent flou est en mode esclave, des blocs de connaissances sont ajoutés au fichier de connaissances. Ce sont les mêmes blocs de connaissances que ceux présentés pour l'agent logique de la section 6.4.

L'agent flou, esclave ou non, est maintenant spécifié aux quatre niveaux du modèle GAM. Une application de test à été réalisée afin de valider la conception et la réalisation de l'agent flou.

6.5.3 Application de test

L'application de test réalisée permet de valider l'agent flou en mode esclave ou non, et évoluant seul ou dans un SMA. L'application floue choisie est la balance inversée. Elle est un exemple classique de la littérature portant sur la logique floue.

L'application réalisée a permis de vérifier le bon fonctionnement du niveau moteur de l'agent flou en mode normal et esclave et dans toutes les configurations de communication. L'agent

flou est sollicité par d'autres agents avec des données et il communique les bons résultats du calcul des sorties aux agents concernés.

6.6 Agent neural

L'agent neural est un agent basé sur le paradigme des réseaux de neurones artificiels. Il faut procéder de manière similaire à l'agent flou pour réaliser le paradigme de cet agent.

6.6.1 Conception

Le paradigme des réseaux de neurones artificiels requiert la définition et la réalisation du calculateur qui ajuste les sorties du réseau en accord avec les entrées. Les concepteurs doivent spécifier la topologie du réseau, les poids associés aux connexions et les fonctions d'activation des neurones. Afin de respecter le modèle GAM, le niveau moteur est composé du calculateur neural et le niveau connaissance de la définition complète du réseau et de la spécification des communications.

La conception de l'agent neural, comme le prescrit le modèle d'agent avec paradigme, est basée sur la classe ParadigmedAgent, et pour l'agent neural esclave, sur la classe SlaveParadigmedAgent. La classe implémentant le paradigme des réseaux de neurones est nommée NeuralNetworkParadigm. Le diagramme de classes de l'agent neural est illustré à la figure 6.9. Voici le rôle de chacune de ces classes :

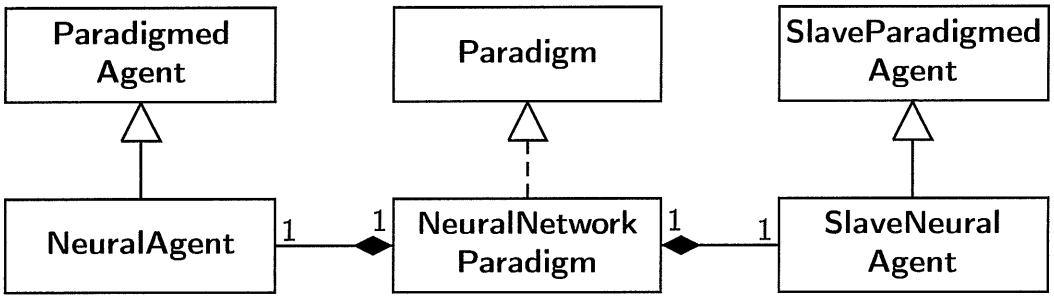


Figure 6.9 – Diagramme de classes du niveau moteur de l'agent neural.

Classe NeuralAgent : Ctte classe offre un agent neural prêt à être utilisé seul ou prêt à être intégré dans un SMA.

Classe SlaveNeuralAgent : Cette classe est semblable à la classe NeuralAgent, mais dans ce cas l'agent neural est esclave.

Classe NeuralNetworkParadigm : Cette classe implémente le paradigme des réseaux de neurones. C'est elle qui manipule le calculateur neural. Elle définit aussi le niveau connaissance de l'agent neural.

6.6.2 Réalisation

Niveau moteur

Le calculateur neural est réalisé avec le langage JAVA et permet de traiter des réseaux de neurones de type *feed-forward* [128] ayant autant de couches internes que les concepteurs le désirent.

Lorsqu'un lien de communication entre l'agent neural et un autre agent est établi, les échanges d'information se passent de la manière suivante. Un agent soumet de nouvelles valeurs en entrée à l'agent neural qui ajuste alors ses entrées en accord avec celles reçues. Les sorties du réseau sont calculées et les nouveaux résultats sont envoyés aux agents intéressés.

Les messages échangés entre l'agent neural et les autres agents sont du type NeuralNetworkInputs et NeuralNetworkData. Le type NeuralNetworkInputs permet aux agents clients de spécifier de nouvelles valeurs en entrée au réseau alors que le type NeuralNetworkData permet à l'agent neural de spécifier l'état courant des entrées et sorties du réseau aux agents clients intéressés.

Niveau connaissance

Le niveau connaissance de l'agent neural est spécifié par un fichier de connaissances. Ce fichier spécifie complètement l'exploitation du paradigme et les communications dans le cas d'un agent esclave. Lors de l'instanciation de l'agent neural, le fichier de connaissances permet de spécifier les définitions des fonctions d'activation, la morphologie du réseau, et si l'agent est esclave, les communications. Les connaissances sont spécifiées à l'aide de deux types de blocs de connaissances :

Type neuralFunction : Bloc de connaissances spécifiant les fonctions d'activation du système neural.

Type neuralNetwork : Bloc de connaissances spécifiant la morphologie du réseau.

La documentation en ligne de la classe NeuralNetworkParadigm donne une description complète de la syntaxe et de la sémantique d'un fichier de connaissances. Si l'agent neural est en mode esclave, des blocs de connaissances sont ajoutés au fichier de connaissances. Ce sont les même blocs de connaissances que ceux présentés pour l'agent logique de la section 6.4.

L'agent neural, esclave ou non, est maintenant spécifié aux quatre niveaux du modèle GAM. Une application de test à été réalisée afin de valider la conception et la réalisation de l'agent neural.

6.6.3 Application de test

L'application de test réalisée permet de valider l'agent neural en mode esclave ou non, et évoluant seul ou dans un SMA. L'application neurale choisie est un exemple classique de la littérature portant sur la représentation d'équations logiques à partir de neurones artificiels [128].

L'application réalisée a permis de vérifier le bon fonctionnement du niveau moteur de l'agent neural en mode normal et esclave, et dans toutes les configurations de communication. L'agent neural est sollicité par d'autres agents avec des données de type NeuralNetworkInputs et il communique les résultats du calcul des sorties aux agents concernés avec le type NeuralNetworkData.

6.7 Agent génétique

L'agent génétique est un agent basé sur le paradigme des algorithmes génétiques. Il faut procéder de manière similaire à l'agent neural afin de réaliser le paradigme de cet agent.

6.7.1 Conception

Le paradigme des algorithmes génétiques requiert la définition et la réalisation d'un simulateur de sélection naturelle, d'un chromosome et d'une fonction d'évaluation. Afin de respecter le modèle GAM, le niveau moteur est composé du simulateur et le niveau connaissance de la définition d'un chromosome, de la fonction d'évaluation d'un chromosome et de la spécification des communications.

La conception de l'agent génétique, comme le prescrit le modèle d'agent avec paradigme, est basée sur la classe ParadigmedAgent, et pour l'agent génétique esclave, sur la classe SlaveParadigmedAgent. La classe implémentant le paradigme des algorithmes génétiques est nommée GeneticAlgorithmParadigm. Le diagramme de classes de l'agent génétique est illustré à la figure 6.10. Voici le rôle de chacune de ces classes :

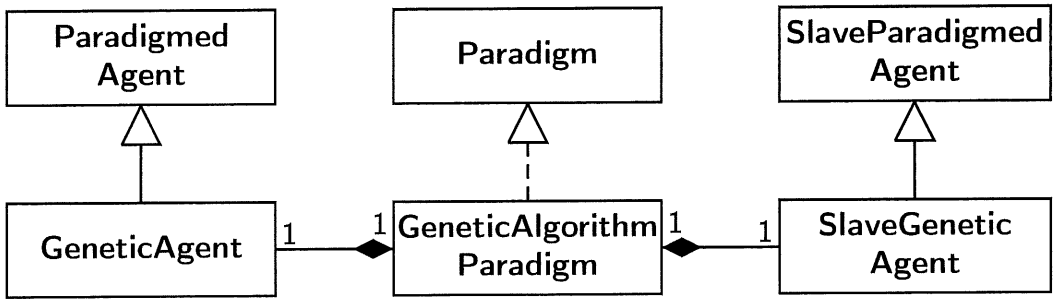


Figure 6.10 – Diagramme de classes du niveau moteur de l'agent génétique.

Classe GeneticAgent : Cette classe offre un agent génétique prêt à être utilisé seul ou prêt à être intégré dans un SMA.

Classe SlaveGeneticAgent : Cette classe est semblable à la classe GeneticAgent, mais l'agent génétique est esclave.

Classe GeneticAlgorithmParadigm : Cette classe implémente le paradigme des algorithmes génétiques. C'est elle qui manipule le simulateur de sélection naturelle. Elle définit aussi le niveau connaissance de l'agent génétique.

6.7.2 Réalisation

Niveau moteur

Le simulateur génétique est réalisé avec le langage JAVA et permet de traiter des requêtes de simulation d'un algorithme génétique.

Lorsqu'un lien de communication entre l'agent génétique et un autre agent est établi, les échanges d'information se passent de la manière suivante. Un agent soumet une nouvelle configuration de simulation. L'agent génétique ajoute cette requête dans sa liste des requêtes de simulation. Lorsque la requête est traitée, la simulation correspondante est effectuée et le résultat est communiqué aux agents intéressés.

Les messages échangés entre l'agent génétique et les autres agents sont du type GeneticAlgorithmConfiguration et Chromosome. Le type GeneticAlgorithmConfiguration permet aux agents clients de spécifier une requête de simulation alors que le type Chromosome permet à l'agent génétique de spécifier le résultat d'une simulation, soit le chromosome retenu, aux agents clients intéressés.

Niveau connaissance

Le niveau connaissance de l'agent génétique est spécifié par un fichier de connaissances. Ce fichier spécifie complètement l'exploitation du paradigme et les communications dans le cas d'un agent esclave. Lors de l'instanciation de l'agent génétique, le fichier de connaissances permet de spécifier la définition d'un chromosome, la fonction d'évaluation des chromosomes, et si l'agent est esclave, les communications. Les connaissances sont spécifiées à l'aide de deux types de blocs de connaissances :

Type chromosome : Bloc de connaissances spécifiant le chromosome utilisé par le simulateur génétique.

Type fitnessFunction : Bloc de connaissances spécifiant la fonction d'évaluation utilisée par le simulateur génétique pour évaluer les chromosomes.

La documentation en ligne de la classe GeneticAlgorithmParadigm donne une description complète de la syntaxe et de la sémantique d'un fichier de connaissances. Si l'agent génétique est en mode esclave, des blocs de connaissances sont ajoutés au fichier de connaissances. Ce sont les mêmes blocs de connaissances que ceux présentés pour l'agent logique de la section 6.4.

L'agent génétique, esclave ou non, est maintenant spécifié aux quatre niveaux du modèle GAM. Une application de test à été réalisée afin de valider la conception et la réalisation de l'agent génétique.

6.7.3 Application de test

L'application de test réalisée permet de valider l'agent génétique en mode esclave ou non, et évoluant seul ou dans un SMA. L'application génétique choisie est le commis voyageur [92]. Elle est un exemple classique de la littérature et est une variante de la recherche du chemin le plus court.

L'application réalisée a permis de vérifier le bon fonctionnement du niveau moteur de l'agent génétique en mode normal et esclave et dans toutes les configurations de communication. L'agent génétique est sollicité par d'autres agents avec des requêtes de simulation de type GeneticAlgoritmConfiguration et il communique les résultats de la simulation aux agents concernés avec le type Chromosome.

6.8 Agent procédural

L'agent procédural est un cas particulier d'agent spécialisé, car son paradigme est le même que celui de l'agent abstrait, ce qui veux dire que son paradigme a été instancié par les niveaux inférieurs. L'agent procédural est directement dérivé de l'agent abstrait et n'est pas basé sur le modèle d'agent avec paradigme de la section 6.3. Le diagramme de classes de l'agent procédural est illustré à la figure 6.11.

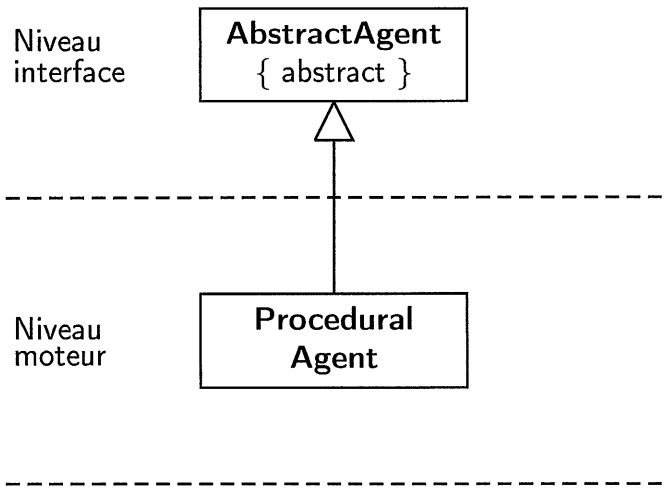


Figure 6.11 – Diagramme de classes du niveau moteur de l'agent procédural.

Une seule classe compose la hiérarchie, la classe ProceduralAgent. Cette classe ne fait que définir les deux méthodes abstraites de l'agent abstrait afin qu'elle devienne concrète. Ces deux fonctions n'exécutent aucune tâche particulière, car c'est aux concepteurs de définir le travail que doit accomplir l'agent procédural.

Avec l'agent procédural, les concepteurs doivent gérer les communications directement à partir des fonctionnalités offertes par l'agent abstrait. Les types d'informations échangées sont au choix des concepteurs. Ils ont le contrôle total sur le travail effectué par l'agent procédural et sur ses communications.

L'agent procédural est en fait l'agent abstrait rendu concret sans aucune modification. Il n'y a pas d'application de test, car les outils de débogage ont été réalisés avec cet agent (voir la section suivante) et ils testent de manière extensive ses capacités.

6.9 Outils de débogage

Les sections 3.7 et 5.7 ont identifié des outils qui aident au débogage de SMA. La figure 3.6 illustre les outils en question. Un outil d'estampage temporel fait aussi partie des outils souhaitables pour le débogage. Les sections suivantes discutent des outils de débogage réalisés dans l'environnement GEMAS.

6.9.1 Outils de manipulation, de collection et de visualisation de traces

L'environnement GEMAS offre les outils de collection et de visualisation de traces. L'outil de manipulation de traces n'a pas été réalisé, car son rôle n'est pas essentiel pour démontrer la validité de l'environnement GEMAS et de ses outils.

Les outils de collection et de visualisation de traces ont été réalisés à partir de l'agent procédural. Ils sont intégrés dans un seul outil nommé *TraceViewerTool* (TVT). La figure 6.12 montre l'interface de cet outil en situation réelle.

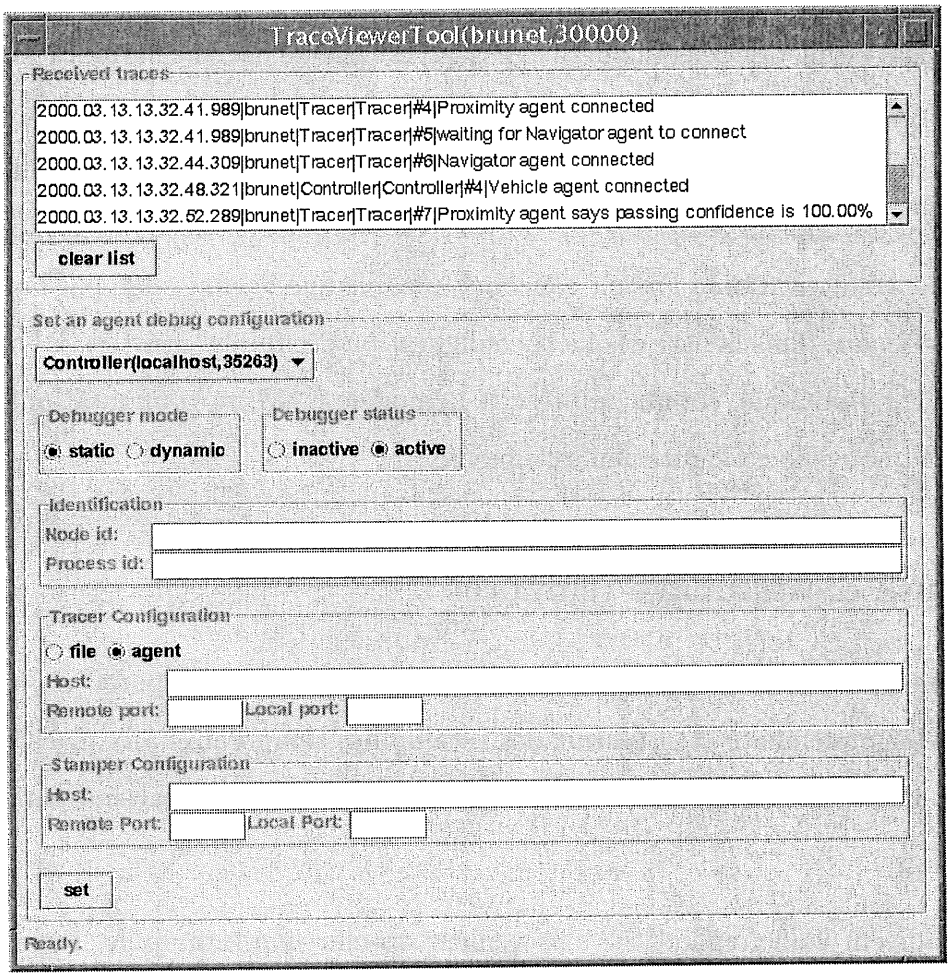


Figure 6.12 – Outil de collection et de visualisation de traces : le *TraceViewerTool*.

La barre de titre du TVT permet d'identifier la machine sur laquelle il s'exécute ainsi que le port serveur de liens qu'il utilise. La fenêtre du TVT est divisée en deux sections, la section trace identifiée par l'encadrement nommé *Received traces* et la section configuration identifiée par l'encadrement nommé *Set an agent debugger configuration.* Voici une description de ces deux sections :

Section trace : Elle est composée d'une aire d'affichage et d'un bouton. L'aire d'affichage est utilisée pour présenter les traces reçues des agents qui sont connectés. Le bouton nommé *clear list* permet de vider la liste des traces reçues. Les traces sont affichées dans l'ordre reçu.

Section configuration : Elle permet de modifier la configuration de débogage d'un agent. Lorsqu'un agent établit une connexion avec le TVT, il est ajouté dans la liste d'agents. Cette liste permet de sélectionner l'agent qui est affecté par une modification éventuelle. La nouvelle configuration de l'agent est spécifiée par les différentes zones en bas de la liste : le mode de débogage, le statut de débogage, la configuration du TVT et de l'agent d'estampage temporel qui doivent être contactés. Le bouton *set* transmet la nouvelle configuration à l'agent concerné une fois que l'agent ciblé par la modification est sélectionné dans la liste et que la configuration désirée est ajustée. Si l'agent visé est en mode dynamique, comme le prescrit le modèle GAM, sa configuration est changée, sinon, la nouvelle configuration est ignorée.

6.9.2 Outil d'estampage temporel

L'environnement GEMAS offre aussi un mécanisme d'estampage temporel. Un agent dédié à cette tâche est implémenté et il fournit des estampilles temporelles aux agents qui en font la demande. Cet outil est nommé *TimeStampTool* (TST) et a été réalisé à partir de l'agent procédural.

Les agents doivent solliciter le TST s'ils veulent obtenir une estampille. Pour ce faire, un agent intéressé établit une connexion avec le TST et une estampille sera fournie sur demande de l'agent client.

Le TST n'offre pas d'interface graphique, mais il peut afficher toutes ses actions, si la demande en est faite lors de son instanciation. La figure 6.13 illustre un extrait des messages affichés par le TST. Deux types de messages sont émis. Le premier type, le type connexion, fournit des informations sur les connexions et déconnexions des agents avec le TST. Après l'établissement d'une connexion, le TST attend des demandes de l'agent client. Le deuxième type, le type estampille, fournit des informations sur les agents qui demandent une estampille. Il est ainsi possible de vérifier si les agents se connectent et font effectivement des demandes d'estampilles au TST.

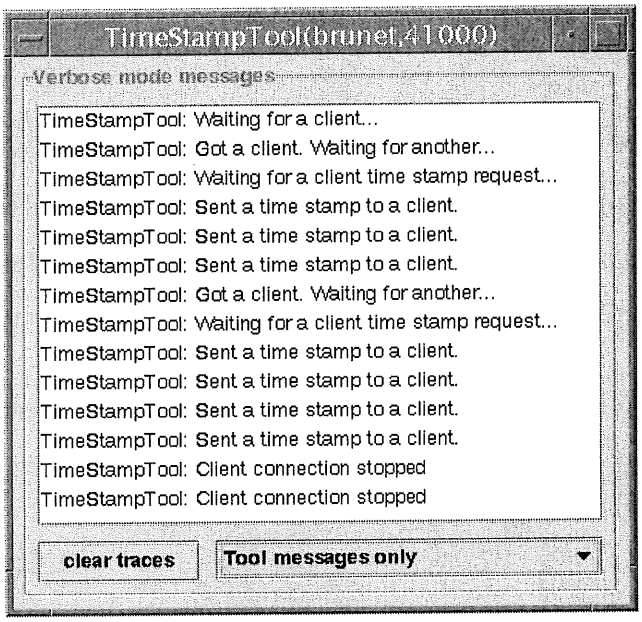


Figure 6.13 – Exemple de messages du *TimeStampTool*.

6.9.3 Remarques

Les outils TVT et TST offrent des services qui aident à la mise au point de SMA. Ils offrent la plupart des fonctionnalités identifiées dans les chapitres 3 et 5. Leur pertinence a été vérifiée en les utilisant pour le développement des agents spécialisés de ce chapitre. Entre autres, la capacité d'avoir plusieurs TVT est très avantageuse. Par exemple, ceci permet de dédier un TVT par agent ou par groupe d'agents. Cela permet aussi de faciliter le travail lorsque plusieurs personnes sont impliquées dans la mise au point du système. Chacun peut avoir son TVT et les traces d'agents peuvent être dirigées vers des TVT particuliers.

Les outils TVT et TST représentent une base sur laquelle peuvent évoluer des outils plus complets. En effet, des travaux futurs permettraient de les rendre encore plus utiles. Par exemple, pour le TVT, la réalisation de l'outil de manipulation permettrait de récupérer les traces laissées localement sur différentes machines. Aussi, la manipulation des traces dans le TVT est inexistante ; elles sont affichées dans l'ordre reçu. Des fonctionnalités de filtrage et de tri au niveau de l'affichage seraient très utiles afin de permettre à l'usager de se concentrer uniquement sur certaines catégories de traces.

6.10 Discussion

Ce chapitre a montré la réalisation du modèle GAM et de l'environnement GEMAS. Il a permis de valider les concepts énoncés dans le chapitre 5. Afin de réaliser l'environnement GEMAS, une conception orientée objet, le langage JAVA, et la plate-forme UNIX ont été retenus pour le développement.

L'agent abstrait de l'environnement GEMAS ainsi que toutes les fonctionnalités des niveaux communication et interface ont été réalisées. La réalisation de l'agent abstrait sert de base aux agents spécialisés.

Le concept d'agent avec paradigme généralise la structure des agents spécialisés et assure ainsi une uniformité dans les fonctionnalités et les interfaces avec les différents paradigmes. L'agent avec paradigme esclave généralise les fonctionnalités de communication pour les paradigmes ne prévoyant pas de mécanismes de communication.

Le concept d'agent avec paradigme est un concept d'implémentation qui s'avère une généralisation pratique très importante lors du codage. Il permet un traitement uniforme des paradigmes, et dans le cas d'agents esclaves, il permet un traitement uniforme des communications.

Les agents spécialisés sont basés sur l'agent avec paradigme. Tous les agents spécialisés de l'environnement GEMAS ont été réalisés : logique, logique floue, réseaux de neurones artificiels, algorithmes génétiques et procédural. Des applications de tests ont été réalisées

pour chacun des paradigmes afin de valider l'implémentation et les comportements dans un SMA.

Les agents spécialisés évolués n'ont pas été réalisés, car il a été démontré que ces agents peuvent être construits à partir des agents spécialisés de base. Si les agents spécialisés de base sont fonctionnels, par extension, les agents spécialisés évolués le sont aussi.

Les outils de débogage de l'environnement GEMAS ont aussi été réalisés. L'outil de visualisation de traces et l'outil d'estampage temporel s'avèrent être des outils d'une utilité indéniable. Ils ont d'ailleurs été parmi les premières réalisations de l'environnement afin de pouvoir être utilisés pour développer les agents spécialisés et mettre au point l'environnement GEMAS et ses applications de test.

L'environnement GEMAS, les outils de débogage, les agents spécialisés de base et les applications de test représentent plus de 21000 lignes de code JAVA et sont totalement portables. Des tests ont été complétés avec succès sur une plate-forme UNIX et sur une plate-forme PC sous Windows sans aucune modification.

Le modèle GAM et sa conception à niveaux ne constitue pas seulement une division conceptuelle, elle est aussi une excellente division fonctionnelle lors de l'implémentation. Le modèle a été réalisé couche par couche, de la couche communication à la couche connaissance. La répartition des concepts à l'intérieur des couches du modèle permet vraiment une conception modulaire, et les différents niveaux sont vraiment indépendants.

Il est clair que la conception orientée objet et la réalisation présentée dans ce chapitre ne représentent qu'une implémentation possible. Le langage JAVA et les choix de réalisation auraient pu être différents. La réalisation montrée n'est qu'une démonstration que les concepts avancés au chapitre 5 sont valides. Une autre réalisation complètement différente basée sur les concepts du modèle GAM peut être tout aussi acceptable. D'ailleurs, la réalisation présentée souffre de certaines limitations qui méritent d'être abordées.

Les mécanismes d'exceptions JAVA sont utilisés pour annoncer des problèmes de communication. Un seul type d'exception est lancé, ce qui n'est pas assez discriminatoire dans

certaines applications afin de récupérer avantageusement de l'exception. Une plus grande variété d'exceptions serait souhaitable.

Bien que les agents spécialisés puissent laisser des traces de leurs actions, il n'est pas possible de vraiment contrôler leur exécution à distance afin de gérer plus efficacement des séances de débogage multi-agents. Ces séances deviennent rapidement complexes lorsque plusieurs agents sont impliqués. Un modèle générique d'événement pour la gestion et le contrôle externe d'agent avec paradigme serait un outil de développement très utile. Par exemple, faire du pas à pas avec un agent et visualiser certaines données internes intermédiaires. Le problème est de définir ce qu'est un pas pour un paradigme et de définir comment un agent et un paradigme peuvent être inspectés internement.

Les communications primitives n'ont pas été abordées et les classes qui les réalisent ne sont pas implémentées, car elles n'étaient pas nécessaires pour la validation de l'environnement GEMAS et du modèle GAM. Les communications primitives devraient quand même être implémentées dans des travaux futurs.

Il serait souhaitable que les outils de débogage offrent des fonctionnalités supplémentaires. Par exemple, il serait utile d'avoir la possibilité de sauvegarder dans un fichier les traces affichées dans la fenêtre du TVT ou encore d'avoir un outil permettant de faire la récupération automatique des traces laissées localement sur différentes machines.

Bien que certaines fonctionnalités seraient avantageuses ou pourraient être ajoutées, les réalisations effectuées atteignent leur but principal qui est de démontrer la validité des concepts du modèle GAM et de l'environnement GEMAS.

Une application aux véhicules autonomes beaucoup plus imposante que les tests effectués dans ce chapitre est réalisée afin de valider l'environnement GEMAS. Elle est le sujet des chapitres qui suivent.

TROISIÈME PARTIE

APPLICATION AUX VÉHICULES AUTONOMES

CHAPITRE 7

VERS UN MODÈLE COMPLET DE PILOTE

Les concepteurs d'un modèle complet de pilote doivent intégrer dans un même système des tâches de nature très différente qui évoluent sur des échelles de temps qui varient de plusieurs ordres de grandeurs. Comme indiqué au chapitre 4, la modélisation complète d'un pilote pour un véhicule autonome est un problème intéressant à plusieurs niveaux et des défis restent à surmonter. L'architecture est généralement le thème central, car le système doit avoir la capacité d'intégrer des tâches de nature hétérogène dans un même système et elle doit être extensible et portable.

7.1 Introduction

Différentes architectures ont été proposées par les chercheurs pour modéliser un pilote ; les plus courantes sont les architectures à niveaux. Elles semblent faire l'unanimité auprès des chercheurs, car elles ont plusieurs avantages par rapport aux autres :

- modélisation distribuée, donc plus facilement réalisée en parallèle ;
- modularité qui facilite la maintenance et l'extension du système ;
- facilité de traitement des différentes échelles de temps impliquées dans les tâches : le temps réel, le moyen terme et le long terme ;
- facilité de gestion des redondances ;
- activation simultanée de plusieurs tâches ou de comportements différents.

Ce type de modélisation cadre bien avec les problématiques qui sont associées aux SMA, c'est pourquoi les SMA sont choisis comme base de modélisation pour un nouveau modèle de pilote. Dans le cadre des recherches présentées, la définition et la réalisation d'un modèle complet de pilote a deux objectifs principaux :

1. **Définir une architecture complète de pilote** qui est un objectif à deux facettes. La première facette est de proposer une architecture complète de pilote pour véhicule autonome basée sur les SMA. Elle est basée sur les SMA afin qu'elle soit extensible, portable et capable d'intégrer des agents hétérogènes. La deuxième facette est de valider le modèle de pilote proposé en l'implémentant.
2. **Valider l'environnement** GEMAS, un objectif qui a une orientation très pratique. Il s'agit de valider l'environnement GEMAS en réalisant une application réelle. La réalisation du modèle donnera des résultats concrets sur la validité des principes de l'environnement.

Ce chapitre présente une architecture de pilote pour véhicule autonome. L'architecture générale du système est présentée à la section 7.2 ; elle discute aussi des détails des agents. La section 7.3 discute du modèle simplifié de pilote qui permettra de valider l'architecture du modèle complet et l'environnement GEMAS. La section 7.4 termine le chapitre en discutant des modèles proposés.

7.2 Architecture complète de pilote

Le chapitre 4 a permis de faire une analyse des architectures courantes pour la modélisation de pilote. Les architectures présentées sont l'architecture réactive de Brooks, l'architecture TMI de Saridis et l'architecture RCS de Albus. Chacune de ces architectures possède des avantages et des inconvénients et aucune n'est parfaite dans tous les cas. Malgré tout, une architecture doit être choisie.

L'architecture retenue est la TMI de Saridis [129], car elle est un bon compromis entre l'architecture réactive et l'architecture RCS dans le cadre précis de la modélisation d'un pilote de véhicule autonome. Elle représente un bon compromis pour plusieurs raisons :

- Elle permet d'intégrer des composants délibératifs, contrairement à l'architecture réactive qui est purement réactive.
- Elle est beaucoup moins complexe que l'architecture RCS. L'architecture RCS est une architecture générale et complexe pour une théorie de l'intelligence. La complétude et la complexité de cette architecture ne sont pas nécessaires pour l'application réalisée.
- Elle distingue les échelles de temps, tout comme l'architecture RCS.

– Elle peut être adaptée localement, si nécessaire, afin d'avoir une hiérarchie qui peut ressembler à une architecture de type réactive ou de type RCS. Un groupe d'agents peut adopter une hiérarchie ressemblant à l'une ou l'autre afin de satisfaire à leurs besoins.

La section 4.3 a permis d'identifier les tâches principales reliées au pilotage de véhicule autonome ; un modèle complet doit permettre de toutes les intégrer. La figure 7.1 illustre un nouveau modèle de pilote. L'architecture est inspirée de celle de Saridis ; elle conserve les trois niveaux et le principe de **précision grandissante avec intelligence décroissante**. Afin de simplifier cette figure, les tâches à chacun des niveaux ont été regroupées par module, un module étant un agent ou un regroupement d'agents. Une description générale des flots de données de la figure est donnée dans le tableau 7.1. Voici une description de chacun des modules selon les niveaux ainsi qu'une description du fonctionnement du modèle.

Nom	Description
commande	commande du volant et de la vitesse
connaissance	connaissance extraite ou ajoutée
déduction	déduction faite à partir des connaissances
détection	informations sur les objets dans l'environnement
entente	entente de coordination des tâches à accomplir
état	état du véhicule : position, orientation, etc.
informations	informations sur l'environnement
modération	facteurs environnementaux modérant la conduite
prévision	manoeuvre locale prévue
recommandation	manoeuvre locale recommandée
sélection	trajet sélectionné par l'usager
statut	statut sur l'environnement
tracé	position désirée sur la route
trajet	routes à suivre et vitesses maximales associées
validation	validation de manoeuvre locale
vitesse désirée	vitesse désirée du véhicule
vitesse maximum	vitesse maximum permise du véhicule

Tableau 7.1 – Définition des flots de données du modèle complet de pilote.

7.2.1 Niveau exécution

Le niveau exécution assure la manipulation des commandes du véhicule. Il est composé des modules commande, filtrage et urgence. Voici la description de ces modules.

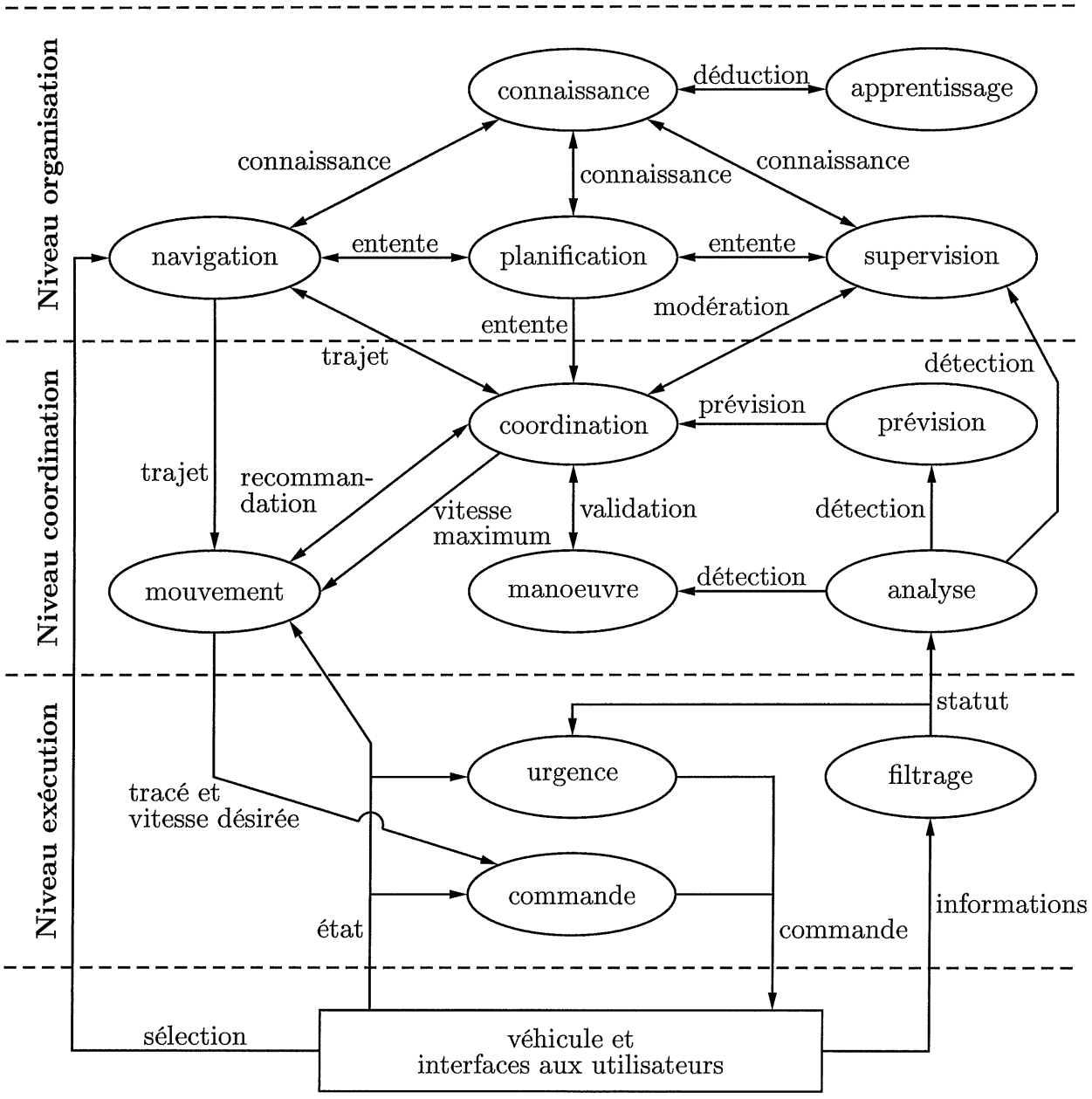


Figure 7.1 – Vers un modèle complet de pilote de véhicule autonome.

Module commande

Ce module s'occupe des commandes latérale et longitudinale du véhicule. Il commande le véhicule en mode normal de conduite, c'est-à-dire lorsque des manoeuvres sévères ne sont pas impliquées. Au niveau exécution, c'est le module qui commande les actuateurs du véhicule la très grande majorité du temps. Le module commande suit le tracé désiré sur la route et ajuste la vitesse désirée du véhicule en accord avec le trajet fourni par le module mouvement du niveau coordination.

Module filtrage

Ce module fait un prétraitement sur les informations dans l'environnement du véhicule. Ce prétraitement sert à extraire rapidement les éléments clé, comme les obstacles, afin que le module urgence puisse les utiliser. Les résultats du prétraitement sont aussi fournis au module analyse du niveau coordination afin que ce dernier puisse faire des traitements plus poussés.

Module urgence

Ce module commande le véhicule lors de manoeuvres sévères du véhicule, comme le dérapage ou l'évitement de collision. Ce module commande le véhicule uniquement en situation d'urgence.

Le module urgence tente de récupérer le plus rapidement possible de la situation dangereuse. Généralement, lorsque le module urgence est activé, la priorité est de garder le véhicule sur la route, d'éviter une collision ou, dans le pire des cas, de minimiser les dommages. La sécurité des passagers ou de la cargaison est la considération première, la direction du véhicule et le confort sont des considérations secondaires.

Les modules commande et urgence sont nécessaires pour la conduite, car la modélisation n'est pas la même pour des manoeuvres sévères que pour des manoeuvres normales. Ces deux modules sont en configuration réactive, comme expliqué dans la section 4.5. Un seul

de ces modules manipule les commandes du véhicule à un moment précis. À tout moment, le module urgence peut prendre le contrôle du véhicule sans le consentement du module commande. Le module urgence redonne le contrôle du véhicule au module commande lorsque la situation d'urgence est terminée.

Le niveau exécution du modèle de pilote s'exécute en temps réel. La commande du véhicule est une tâche de tout instant et ne peut pas être reportée ou arrêtée. Idéalement, ce niveau a un système informatique dédié et indépendant des autres afin d'assurer la disponibilité des ressources du système. Si ce niveau partage les ressources système avec d'autres niveaux, il doit avoir la priorité lorsque la charge du système est grande afin d'assurer une commande adéquate du véhicule.

7.2.2 Niveau coordination

Le niveau coordination est dédié aux tâches et à la planification à moyen terme. Il est composé des modules mouvement, analyse, prévision, coordination et manoeuvre. Voici une description de ces modules.

Module mouvement

Ce module calcule le tracé désiré et la vitesse désirée du véhicule en accord avec le trajet fourni par le module navigation du niveau organisation. Le tracé et la vitesse sont décidés en tenant compte de l'état courant du véhicule et le tracé résultant est fourni au module commande du niveau exécution. La plupart du temps, ce module décide seul le tracé désiré et la vitesse désirée que le véhicule doit suivre. Généralement, il n'y a pas de manoeuvre particulière à exécuter autre que de suivre la route selon le trajet qui a été décidé.

Par contre, lorsque d'autres véhicules ou d'autres objets se rapprochent, des manoeuvres locales peuvent être exécutées. À ce moment, une manoeuvre, comme un dépassement, peut être recommandée par le module coordination. Si la recommandation est acceptée par le mo-

dule mouvement, une nouvelle vitesse désirée et un nouveau tracé désiré sont calculés et sont ensuite communiqués au module commande du niveau exécution.

La manoeuvre locale suggérée par le module coordination au module mouvement est vraiment une recommandation et non un ordre. Le module mouvement peut toujours refuser la recommandation, car l'état du véhicule peut la proscrire.

Le module mouvement peut aussi recevoir du module coordination une nouvelle vitesse maximum. Dans ce cas, cette vitesse maximum ne doit pas être dépassée, car la sécurité du véhicule peut alors être mise en cause. Cette vitesse est déduite des conditions environnementales, comme la condition des routes et la proximité des autres véhicules, et elle n'est pas discutable.

Module analyse

Ce module analyse l'environnement du véhicule pour en extraire des informations qui peuvent être utilisées par d'autres modules du niveau coordination. Par exemple, il extrait des informations permettant de détecter la présence de véhicules ou d'obstacles d'intérêt.

Les informations sont déduites à partir d'images représentant le champ de vision. Ces images sont prétraitées par le module filtrage du niveau exécution afin d'alléger la tâche du module analyse. Les résultats des analyses sont fournis au module prévision, au module manoeuvre et au module supervision du niveau organisation.

Module prévision

Ce module tente de prévoir des manoeuvres particulières qui peuvent survenir à moyen terme. Par exemple, si le véhicule en rattrape un autre dans la même voie, une diminution de la vitesse du véhicule ou un dépassement peut être suggéré.

Les prévisions des situations à moyen terme sont faites à partir des informations extraites par le module analyse. Lorsque des manoeuvres sont avantageuses, elles sont suggérées au module coordination.

Module coordination

Ce module coordonne les choix possibles des manoeuvres locales soumises par le module prévision. Les suggestions sont évaluées par le module manoeuvre et selon les évaluations reçues en retour, la plus appropriée pour la situation présente est choisie et elle est soumise au module mouvement.

Les manoeuvres retenues sont confrontées aux modérations du module supervision, aux tâches du module planification et au trajet fixé par le module navigation. Par exemple, si la manoeuvre retenue est le dépassement, il peut quand même être dangereux de l'exécuter si la voie de gauche est enneigée ou glacée. Le module coordination s'assure que toutes les conditions favorables sont réunies pour une manoeuvre locale en tenant compte de tous les facteurs.

Le module coordination peut solliciter un nouveau trajet au module navigation dans le cas où le trajet spécifié antérieurement ne pourrait pas être respecté. Par exemple, cette situation peut survenir lorsqu'une route est bloquée par un accident ou un embouteillage.

Module manoeuvre

Ce module fait l'évaluation de manoeuvres locales à partir d'une banque de manoeuvres, comme le dépassement par exemple. Ce module ne fait pas concurrence au module urgence du niveau exécution. Les manoeuvres de ce module ne sont pas des manoeuvres extrêmes, mais plutôt des manoeuvres planifiées et exécutées en toute sécurité.

Le module coordination soumet à ce module des manoeuvres à évaluer. Toutes les manoeuvres soumises sont évaluées et le résultat de chacune des évaluations est retourné au module coordination. Une manoeuvre peut être jugée pertinente et être validée. Par contre, une manoeuvre peut être jugée trop risquée et être invalidée ou encore être simplement inconnue et refusée. Une manoeuvre est jugée en fonction des informations obtenues du module analyse.

Le niveau coordination est divisé logiquement en deux : une section s'occupe du mode normal et une autre s'occupe des manoeuvres locales. En mode normal, le module mouvement reçoit le trajet du module navigation, calcule le tracé désiré et la vitesse désirée et les fournit au mo-

dule commande. Lorsque survient une situation qui requiert une manoeuvre locale, les autres modules du niveau coordination, analyse, prévision, coordination et manoeuvre, s'impliquent dans le processus de la sélection d'action à court terme du véhicule.

Les échelles de temps impliquées à ce niveau sont typiques du niveau coordination de la TMI ; elles sont de l'ordre de quelques secondes. Les ressources système exigées par ce niveau sont généralement ponctuelles. Lorsque le système est en mode normal, la plupart des modules sont inactifs. Par contre, lorsque survient une situation de manoeuvre locale, plusieurs modules peuvent entrer en action simultanément et ainsi causer une charge appréciable sur le système. Une priorité d'exécution doit exister à ce niveau afin d'assurer qu'une situation inoffensive et courante ne deviendra pas une situation d'urgence, comme le rapprochement constant d'un véhicule lointain.

7.2.3 Niveau organisation

Le niveau organisation s'occupe des tâches à long terme ou à tendance cognitive. Il est composé des modules connaissances, apprentissage, navigation, supervision et planification. Voici une description de ces modules.

Module connaissances

Ce module contient des connaissances qui sont offertes à tous les modules du niveau organisation. Tous les modules nécessitant ces connaissances afin d'accomplir leurs tâches peuvent solliciter ce module afin de satisfaire à leurs besoins. Ce module contient principalement trois types de connaissances :

Connaissances générales : Ce sont des connaissances globales et générales sur les véhicules et le pilotage de véhicules. Certaines connaissances sont immuables, peu importe le type de véhicule. Deux exemples de ce type de connaissance sont qu'un véhicule peut accélérer et freiner et que la vitesse doit être réduite si la route est glacée ou enneigée.

Connaissances particulières : Ce sont des connaissances particulières sur le véhicule et le système de commande du véhicule qui est effectif. Ce sont généralement des connaissances qui sont de nature plus technique, comme les distances de freinage par rapport à la vitesse. Ce type de données est important, car il permet d'assurer une conduite sécuritaire.

Connaissances apprises : Ce sont des connaissances qui sont acquises par les expériences passées de conduite. Par exemple, les distances de freinage qui sont requises doivent être ajustées pour tenir compte de l'usure des freins et des pneus. Ce type de données est particulier au véhicule malgré que l'adaptation au vieillissement soit une caractéristique générale à la conduite.

Module apprentissage

Ce module analyse les données du module connaissances dans le but de faire des déductions et permettre au système de faire un apprentissage, d'optimiser certaines actions ou de s'adapter. Suite aux manoeuvres du véhicule, ce module analyse les résultats et tente d'améliorer les performances de conduite. Par exemple, si la distance de freinage qui a été requise dans une situation a été plus grande que prévue, il faut donc prévoir une plus grande distance pour la prochaine fois. Ce module peut modifier ou adapter les connaissances particulières et apprises du véhicule à partir des connaissances générales de la conduite.

Module navigation

Ce module planifie un trajet à suivre pour se rendre à une destination choisie. Il décide du parcours pour se rendre à destination en utilisant une base de données routières. Il choisit les routes à emprunter et la vitesse maximale à respecter sur chacune d'elles selon la loi. Le trajet calculé est communiqué au module mouvement du niveau coordination.

Le calcul d'un trajet est fait sur demande. Les modules du niveau organisation peuvent en faire la demande afin de planifier ou d'accomplir certaines activités. Le module coordination en fait la demande suite à la rencontre d'un blocage sur la route. Dans ce cas, le nouveau trajet est communiqué au module mouvement.

Le module navigation peut utiliser le module connaissances et planification afin de vérifier si le trajet choisi est adéquat à certains égards. Par exemple, certains gros véhicules lourds ne peuvent pas utiliser de petites avenues.

Module supervision

Ce module, comme son nom l'indique, supervise le pilotage afin de s'assurer que certaines règles de base de conduite sont observées, comme réduire la vitesse maximum du véhicule lorsque la route est enneigée. Ce module fait une supervision de haut niveau et modère la conduite afin d'augmenter la sécurité ou le confort.

Le module supervision utilise le module connaissances afin d'obtenir des renseignements sur le véhicule et sur la conduite et il utilise aussi les informations fournies par le module analyse du niveau coordination. Le module supervision fait ses recommandations aux autres modules à partir des informations obtenues et de ses capacités d'analyse. Ce module fait des recommandations qui peuvent être précises, comme dans le cas de la vitesse maximale. Il peut aussi faire des suggestions plus générales comme dans le cas où il y a tempête de neige ; il peut alors suggérer au module planification de favoriser les routes principales, car elles sont généralement déneigées plus rapidement que les routes secondaires.

Le module supervision fait ses suggestions de modération au module coordination. Il peut les faire automatiquement lorsque l'environnement change, comme une pluie soudaine. Il peut aussi les faire sur demande pour des besoins précis. Par exemple, si le module coordination opte pour un dépassement, il valide la manoeuvre auprès du module supervision pour s'assurer qu'il n'y a pas de contre-indications.

Module planification

Ce module permet de faire de la planification à long terme et de la prise de décisions. Ce module aide dans certaines situations où plusieurs alternatives sont possibles. Les objectifs de conduite ou les préférences liées à une mission particulière peuvent alors être prises en compte.

Le module planification permet aussi de résoudre des conflits qui peuvent survenir au niveau des objectifs de conduite, comme minimiser le temps du trajet et maximiser la sécurité. Le module permet alors de faire un choix en tenant compte des objectifs de conduite et des contraintes globales établies par le module supervision.

Ce module peut être sollicité par les autres modules du niveau organisation afin de résoudre des conflits entre différentes solutions. Le module planification connaissant les priorités et les objectifs de conduite, il peut s'avérer un excellent conseiller pour les autres modules afin d'améliorer les performances générales du système.

Un peu à la façon du niveau coordination, le niveau organisation est divisé logiquement : une section est dédiée au mode normal et une section dédiée est à l'apprentissage et la planification. En mode normal, les modules navigation, supervision et planification font leur travail respectif afin d'assurer le déplacement du véhicule d'un point à un autre, et cela en toute sécurité. L'autre section s'occupe d'optimiser les performances à long terme du système.

Les échelles de temps impliquées à ce niveau sont de quelques secondes, pour la partie en mode normal, à quelques minutes ou quelques heures pour la partie apprentissage. Dans plusieurs cas, l'apprentissage n'a pas besoin d'être fait sur-le-champ, il peut être fait plus tard à un moment opportun, comme lorsque le véhicule est au repos.

7.2.4 Remarques

Le modèle de pilote présenté est adéquat pour un AHV. Il contrôle un véhicule et le déplace d'un point à un autre sur des routes connues. Le modèle peut s'adapter à différentes situations dynamiques, comme le dépassement d'un véhicule ou calculer un nouveau trajet suite à un blocage sur la route. Il peut même tenter de récupérer d'une situation d'urgence.

Par contre, l'architecture doit être adaptée pour les AGV ou les ALV. Par exemple, les AGV sont des véhicules qui ont souvent des tâches à accomplir, comme livrer le courrier ou déplacer des objets. Dans ce cas, des modules doivent être ajoutés au niveau exécution afin de pouvoir manipuler les bras ou les pinces. D'autres modules devront aussi être ajoutés

aux niveaux coordination et organisation afin d'accomplir les tâches spécifiques. Le modèle présenté doit être étendu et adapté selon les besoins de l'application.

Le modèle présenté comporte plusieurs avantages. Premièrement, c'est un modèle qui est basé sur une architecture à niveaux, un type d'architecture qui semble faire l'unanimité chez les chercheurs. Deuxièmement, le modèle étant divisé en modules et étant basé sur les SMA, il est facilement modifiable et il est extensible, portable et capable d'intégrer des agents hétérogènes. Le modèle résout certains des problèmes importants liés à l'architecture.

La configuration réactive des modules commande et urgence pourrait être substituée par un agent à personnalités multiples, comme discuté à la section 5.8. Dans ce cas, un seul agent serait nécessaire pour implémenter les modules commande et urgence. Ce nouveau module changerait de personnalité lorsque la situation le requiert en passant d'un module de commande en mode normal (paradigme procédural) à un module de commande pour la situation d'urgence (paradigme réactif).

Le premier objectif derrière la proposition d'un modèle de pilote était d'obtenir une architecture qui est complète. Ce premier objectif est atteint par le modèle présenté dans cette section. Le deuxième objectif était de valider l'environnement GEMAS. Dans ce cas, le modèle complet n'a pas besoin d'être réalisé, un sous-ensemble des modules présentés est suffisant. La section suivante expose le modèle simplifié qui sera implémenté.

7.3 Modèle simplifié de pilote

Le premier objectif du modèle simplifié de pilote est de réaliser une partie du modèle complet de la section précédente afin de valider de manière pratique l'environnement GEMAS. Afin d'atteindre cet objectif, le modèle simplifié intègre des agents hétérogènes dans un même système qui sera implémenté. Le deuxième objectif est de démontrer la validité de l'architecture complète de pilote. Cet objectif est atteint en intégrant des agents à chacun des niveaux de l'architecture complète pour ainsi valider sa structure.

Les modules du modèle complet qui sont retenus doivent permettre d'obtenir un système qui réagit et qui accomplit une tâche afin que des tests concluants soient obtenus. Les modules

commande, mouvement, prévision, navigation et supervision sont ceux qui sont retenus. La figure 7.2 illustre l'architecture du modèle simplifié de pilote et le tableau 7.2 donne une définition des flots de données de cette figure.

Module commande : Les fonctionnalités complètes de ce module sont retenues. Les commandes latérale et longitudinale du véhicule sont implémentées par un agent autonome nommé contrôleur.

Module mouvement : Les fonctionnalités de ce module ne sont retenues qu'en partie. Le choix du tracé en fonction du trajet fourni par le module navigation est retenu, mais le traitement des recommandations du module coordination n'est pas implémenté. Les fonctionnalités de ce module sont réalisées par un agent autonome nommé traceur.

Module prévision : Seule la capacité de détecter la proximité des objets a été conservée. Cette capacité est implémentée par un agent autonome nommé proximité.

Module navigation : Seule la capacité de sélectionner un trajet a été conservée. Les capacités de ce module sont implémentées par un agent autonome nommé navigateur.

Module supervision : Seule la capacité d'adapter la conduite aux conditions météorologiques a été conservée. Les capacités de ce module sont implémentées par un agent autonome nommé superviseur.

Le modèle illustré à la figure 7.2 est une variante d'un modèle qui a déjà été présenté dans [23, 24]. Il permet de suivre un trajet fixé par l'agent navigateur. Une fois que le trajet est décidé, l'agent traceur fixe le tracé désiré et la vitesse désirée qui sont ensuite fournis à l'agent contrôleur. L'agent contrôleur ajuste la vitesse et guide le véhicule avec une marge d'erreur acceptable. Ce fonctionnement est identique au modèle complet de pilote.

Par contre, certaines choses sont différentes, car les modules du modèle complet ne sont pas tous implémentés. L'agent proximité détecte les objets qui sont trop près et fait ses recommandations directement à l'agent traceur, plutôt qu'au module coordination comme dans le modèle complet. Aussi, les agents proximité et superviseur communiquent directement l'un avec l'autre étant donné que le module coordination du modèle complet n'est pas im-

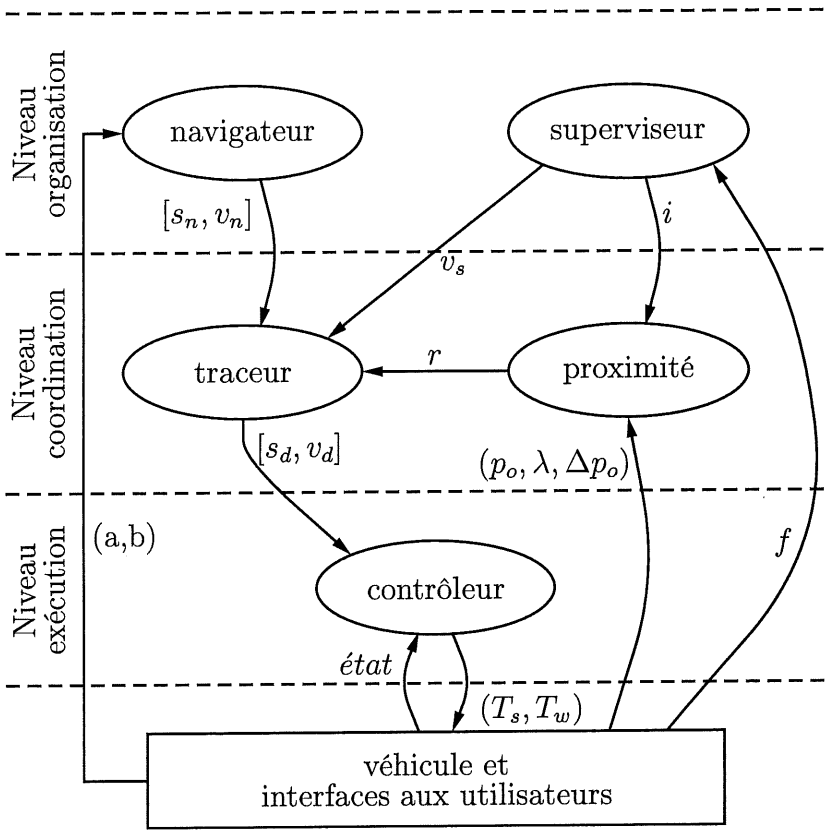


Figure 7.2 – Modèle simplifié de pilote.

plémenté. Malgré les simplifications et les changements, les agents proximité et superviseur peuvent quand même suggérer des manoeuvres locales ou changer la vitesse maximale du véhicule.

Cette version simplifiée du modèle complet atteint les objectifs de réalisation fixés, car tous les niveaux du modèle complet sont implémentés afin de le valider et une application avec des agents hétérogènes est réalisée afin de tester l'environnement GEMAS avec une application réelle. Voici une description plus détaillée de chacun des agents composant le modèle simplifié de pilote.

7.3.1 Agent contrôleur

Cet agent, situé au niveau exécution du modèle simplifié de la figure 7.2, réalise les commandes latérale et longitudinale du véhicule. Plusieurs systèmes de commandes ont été ex-

Nom	Unité	Description
a		départ d'un trajet
b		destination d'un trajet
$état$		$(x, y, \varphi, \theta, \delta)$
f		facteurs environnementaux
i		information environnementale
p_o	m	position relative de l'objet
r		recommandation
s_d	m	segment de tracé désiré du traceur
s_n	m	segment de trajet désiré du navigateur
T_w	Nm	couple sur les roues
T_s	Nm	couple sur le volant
v_d	m/s	vitesse désirée du traceur
v_n	m/s	vitesse maximale du navigateur
v_s	m/s	vitesse maximale du superviseur
x	m	coordonnée réelle longitudinale
y	m	coordonnée réelle latérale
θ	rad	angle de rotation des roues
δ	rad	angle de direction des roues avant
Δp_o	m	changement de position relative de l'objet
λ	rad	angle de l'obstacle
φ	rad	angle d'orientation du véhicule
[...]		une liste de ...

Tableau 7.2 – Définition des flots de données du modèle simplifié de pilote.

posés à la section 4.3. Parmi ces modèles, ceux réalisant le contrôle latéral et longitudinal du véhicule sont ceux retenus, plus particulièrement celui de Ehsani et al. [44].

Une analyse du modèle de commande présenté dans [44] permet de déduire les entrées et les sorties du système. Les flux de données étant identifiés par le modèle de commande et le contexte d'utilisation de l'agent dans le SMA étant connu, il est possible d'identifier une structure de l'agent. Elle est illustrée à la figure 7.3.

Le fonctionnement de l'agent est le suivant. Le tracé désiré et la vitesse désirée $[s_d, v_d]$ sont reçus de l'agent traceur par le processus réception et ils sont stockés dans les données dynamiques de l'agent. L'état courant du véhicule est reçu par le processus détection et il est stocké dans les données dynamiques de l'agent. Le processus modèle de commande implémente le modèle de contrôle du véhicule et génère les sorties en accord avec la configuration du

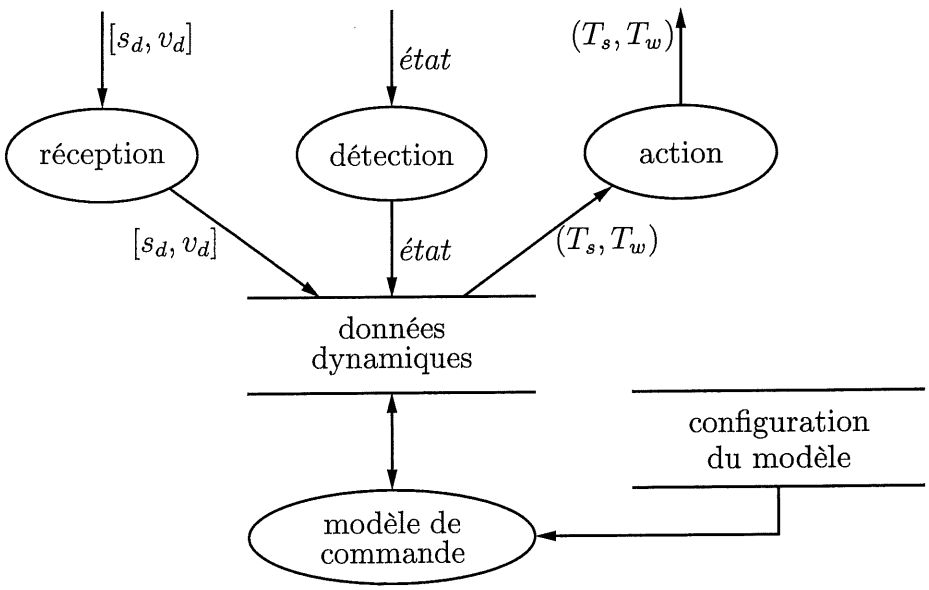


Figure 7.3 – Structure de l'agent contrôleur.

modèle et des données dynamiques du système. Le processus action prend les sorties stockées (T_s, T_w) et contrôle les commandes du véhicule.

Le paradigme choisi pour l'implémentation de cet agent est de type procédural ou orienté objet. La nature de la modélisation du contrôle est essentiellement mathématique ; elle consiste en l'implémentation d'équations. Aussi, la nature temps réel de cet agent exige qu'il soit le plus efficace possible lors de l'exécution. L'agent spécialisé de type procédural de l'environnement GEMAS est donc sélectionné pour l'implémentation de l'agent contrôleur.

7.3.2 Agent traceur

Cet agent, situé au niveau coordination du modèle simplifié de la figure 7.2, calcule le tracé désiré qui doit être suivi. Le tracé est calculé à partir du trajet fourni par l'agent navigateur au niveau organisation, puis il est communiqué à l'agent contrôleur. L'agent traceur ne tient pas compte des recommandations faites par l'agent proximité, même si elles sont pertinentes, car le traitement des recommandations n'est pas une fonctionnalité retenue dans le modèle simplifié de pilote. L'agent navigateur fournit aussi la vitesse maximale à respecter sur les routes qu'il a retenues pour le trajet. Cette vitesse maximale v_n peut être diminuée par

l'agent superviseur afin de tenir compte des conditions environnementales. Lors du choix de son tracé, l'agent traceur s'assure de tenir compte de la vitesse v_s de l'agent superviseur. Malgré les simplifications, l'agent traceur est nécessaire afin de fusionner le trajet donné par l'agent navigateur et la vitesse suggérée par l'agent superviseur.

La littérature sur la modélisation de choix de tracés, comme [79, 84, 85, 142], expose plusieurs modèles de comportements de pilotes. Le tracé généralement suivi par un conducteur est celui qui minimise l'accélération latérale [63], mais, étant donné les objectifs de réalisation de l'application, le tracé sélectionné par l'agent traceur est celui qui suit le centre de la route.

Les tâches de l'agent et ses communications étant identifiées, il est possible d'établir son architecture ; elle est illustrée à la figure 7.4. Le fonctionnement de l'agent se passe comme suit. Le processus réception reçoit le trajet $[s_n, v_n]$ de l'agent navigateur et le stocke en mémoire. Le calcul du tracé désiré et de la vitesse désirée $[s_d, v_d]$ est effectué par le processus calcul du tracé à l'aide du trajet stocké et de la vitesse v_s reçue de l'agent superviseur, puis le résultat est envoyé à l'agent contrôleur par le processus envoi. Les recommandations r reçues de l'agent proximité ne sont pas stockées par le processus réception et sont ignorées, car cette fonctionnalité n'est pas implémentée.

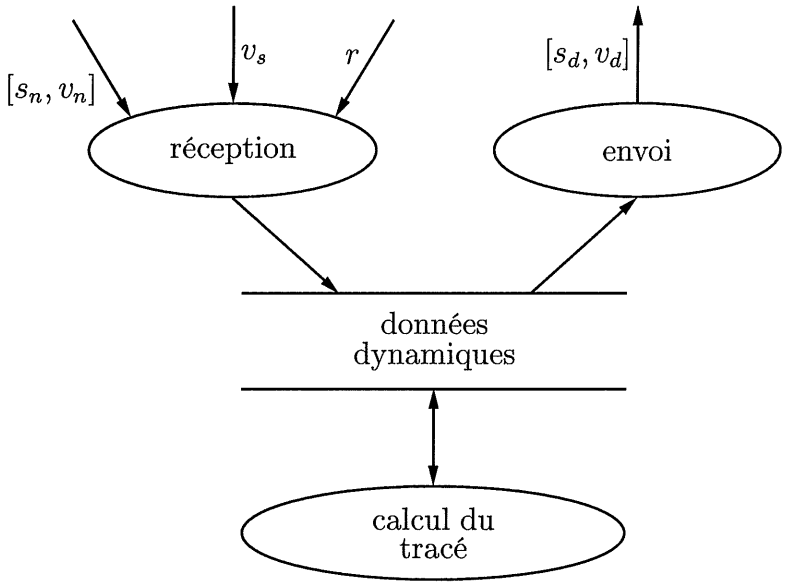


Figure 7.4 – Structure de l'agent traceur.

De la même manière qu'avec l'agent contrôleur, cet agent est implémenté avec un paradigme procédural. La nature mathématique de la tâche, le peu de connaissances et le peu de traitement de haut niveau justifient ce choix de paradigme. Ainsi, l'agent spécialisé de type procédural de l'environnement GEMAS est sélectionné pour l'implémentation de l'agent traceur.

7.3.3 Agent proximité

Cet agent, situé au niveau coordination du modèle simplifié de la figure 7.2, détecte la proximité des autres véhicules et suggère une manoeuvre lorsque des véhicules ou des objets se rapprochent trop près. En suggérant une manoeuvre, cet agent tente de prévenir les situations où des entités environnantes pourraient devenir une menace de collision à moyen terme.

Selon Zapata [160], l'évitement de collision ou d'obstacle est généralement solutionné par une approche réactive. Ceci est différent de ce que l'agent proximité tente de faire. Il ne tente pas d'éviter une collision, mais plutôt d'empêcher qu'une entité détectée et qui est assez éloignée ne se rapproche trop d'un périmètre de sécurité. Ce type de situation est bien géré avec la logique floue, car des réactions typiques d'un pilote sont du type « si le véhicule devant est moyennement près alors faire un dépassement » ou encore « si le véhicule est près alors ralentir un peu ».

La description des tâches et des communications de l'agent permet d'établir son architecture. Elle est illustrée à la figure 7.5. L'agent proximité obtient des détecteurs du véhicule les informations des entités environnantes $(p_o, \lambda, \Delta p_o)$. À partir de ces informations, le processus sélection décide d'une manoeuvre. La manoeuvre sélectionnée devient une recommandation r qui est envoyée à l'agent traceur avec l'aide du processus envoi.

Le processus sélection tient compte dans ses mécanismes de décisions des informations environnementales i fournies par l'agent superviseur. Ces informations sont envoyées automatiquement par l'agent superviseur.

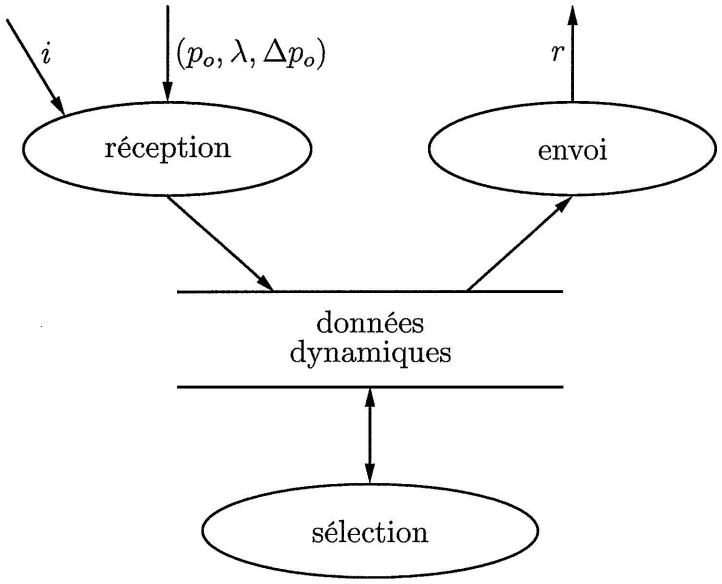


Figure 7.5 – Structure de l'agent proximité.

Cet agent traitant avec différents degrés de proximité et de vitesse des véhicules avoisinants est réalisé avec la logique floue. L'agent spécialisé de type flou de l'environnement GEMAS est donc sélectionné pour l'implémentation de l'agent proximité.

7.3.4 Agent navigateur

Cet agent, situé au niveau organisation du modèle simplifié de la figure 7.2, utilise une base de données cartographique pour décider du trajet afin d'atteindre une destination. Le calcul d'un trajet se fait sur demande. Tout agent nécessitant un trajet pour accomplir une tâche peut solliciter l'agent navigateur pour un trajet.

La problématique liée à la recherche d'un trajet ou d'un chemin est très étudiée en intelligence artificielle. Ce problème de recherche est exposé, sous une forme ou une autre, dans la plupart des livres d'introduction à l'intelligence artificielle, par exemple [92, 128]. Une variante de ce problème est souvent présentée comme le problème du commis voyageur (*travelling salesperson*).

La description des tâches et des communications de l'agent permet d'établir son architecture. Elle est illustrée à la figure 7.6. L'agent navigateur reçoit une demande de trajet (a, b) qui

est stockée par le processus réception. Le trajet est ensuite calculé par le processus calcul du trajet à l'aide d'une base de données cartographiques. Le trajet calculé $[s_n, v_n]$ est alors envoyé à l'agent traceur par le processus envoi.

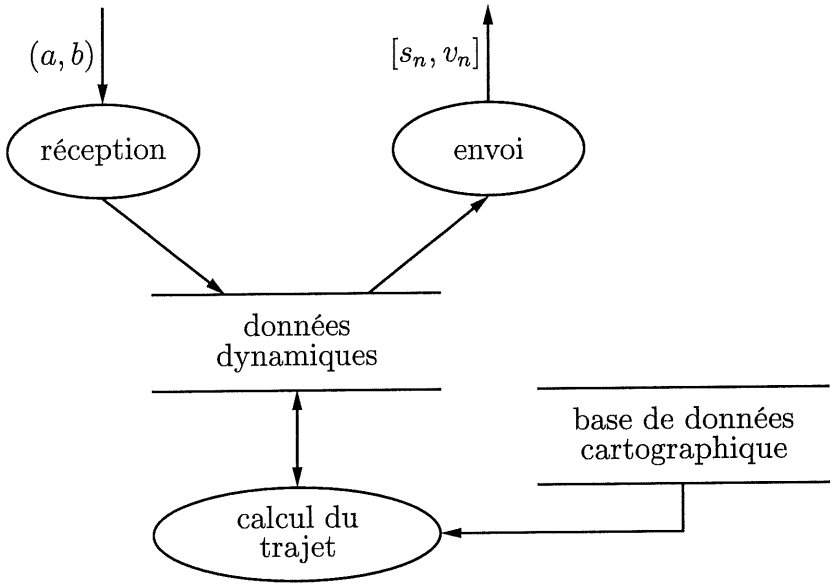


Figure 7.6 – Structure de l'agent navigateur.

Les techniques généralement utilisées pour résoudre les problèmes liés à la problématique de la recherche de chemin sont la logique et les algorithmes génétiques pour des problèmes très complexes. La logique est le paradigme retenu dans ce cas-ci, car les besoins de l'application réalisée ne sont pas assez complexes pour justifier l'utilisation des algorithmes génétiques. Ainsi, l'agent spécialisé de type logique de l'environnement GEMAS est sélectionné pour l'implémentation de l'agent navigateur.

7.3.5 Agent superviseur

Cet agent, situé au niveau organisation du modèle simplifié de la figure 7.2, agit comme un modérateur afin d'assurer une conduite sécuritaire. Il fait une supervision de la conduite afin de suggérer une limite de vitesse qui tienne compte des facteurs environnementaux, comme la neige et la glace. Cet agent agit comme le **bon sens** d'un pilote humain, et sa tâche est de type cognitif.

Le **bon sens** (*common sense*), n'est pas si simple à implémenter que les chercheurs le croyaient *a priori*. La modélisation du **bon sens** n'est pas une problématique triviale ; les travaux du projet CYC [61] le montrent bien. Par contre, pour les fins de l'application choisie, la logique permet d'implémenter le raisonnement simple nécessaire à cet agent.

La description des tâches et des communications de l'agent permet d'établir son architecture. Elle est illustrée à la figure 7.7. L'agent superviseur obtient des capteurs du véhicule des informations environnementales f. Ces informations sont reçues et stockées par le processus réception. Ces informations environnementales amènent le processus raisonnement à réévaluer la situation courante. Selon ses connaissances, il suggère une nouvelle vitesse maximale v_s à l'agent traceur.

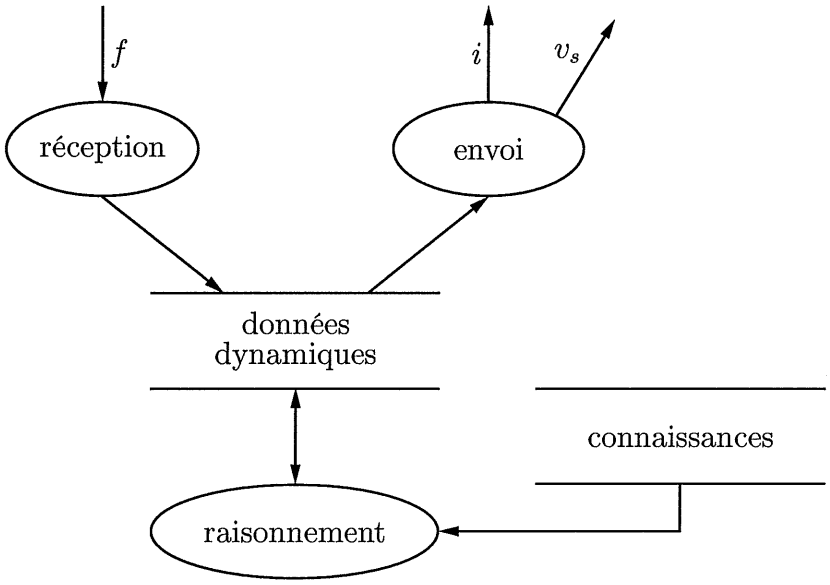


Figure 7.7 – Structure de l'agent superviseur.

La logique est le paradigme retenu. Ainsi, l'agent spécialisé de type logique de l'environnement GEMAS est sélectionné pour l'implémentation de l'agent superviseur.

7.3.6 Remarques

L'implémentation du modèle simplifié n'est pas faite avec un véhicule réel, mais avec un véhicule simulé. Tout le système est simulé, ainsi le véhicule et les interfaces aux utilisateurs

sont aussi simulés. Ils font partie du système présenté à la figure 7.2 et ils sont simulés dans un agent nommé simulateur. L'agent simulateur remplace le véhicule pour les simulations et il est de type procédural. Sa structure est illustrée à la figure 7.8.

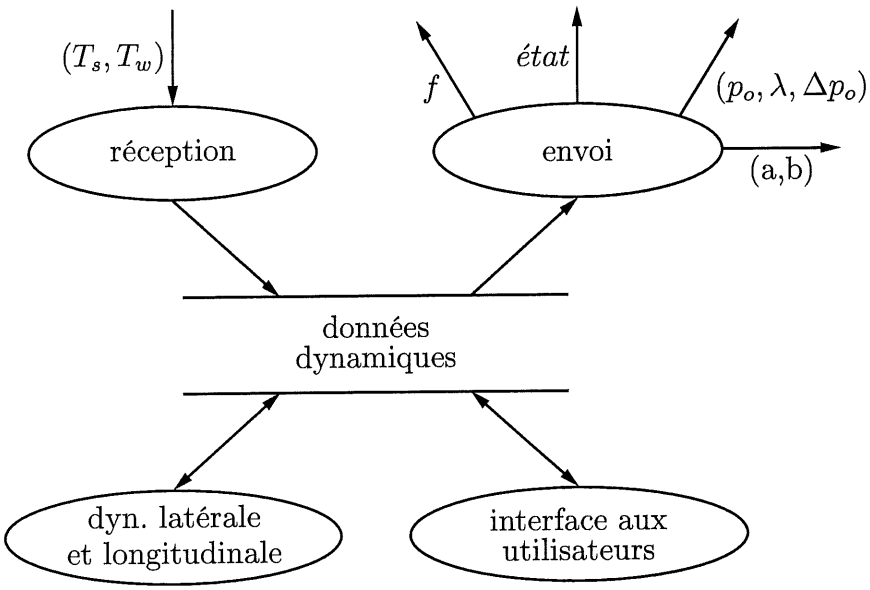


Figure 7.8 – Structure de l'agent simulateur.

Le fonctionnement de l'agent simulateur est le suivant. Les passagers fournissent une destination au système b. Un processus nommé interfaces aux utilisateurs permet d'obtenir interactivement le trajet désiré et de le fournir à l'agent navigateur à l'aide du processus envoi.

La simulation du véhicule et de ses capteurs est une situation moins simple et elle est divisée en deux parties. La première partie concerne les détecteurs. La détection d'entités dans le champ de vision $(p_o, \lambda, \Delta p_o)$ et le changement des facteurs environnementaux f doivent être simulés afin de les générer et de les envoyer aux agents proximité et superviseur. Ces informations peuvent être générées interactivement et de manière imprévisible, du point de vue du système, avec une interface aux utilisateurs. Le processus interface aux utilisateurs s'occupe de cette partie et les données sont envoyées à l'aide du processus envoi.

La deuxième partie concerne la simulation du comportement dynamique du véhicule suite à la réception des commandes (T_s, T_w) de l'agent contrôleur. La réaction est simulée à partir du modèle mathématique du véhicule. Ces modèles simplifiés des comportements dynamiques sont souvent utilisés dans la littérature afin de valider les systèmes de commandes de véhi-

cules, par exemple [79, 105]. Le processus dynamiques latérale et longitudinale s'occupe de la partie dynamique du véhicule. L'état calculé à partir du modèle est ensuite envoyé à l'agent contrôleur à l'aide du processus envoi.

Ainsi, le modèle simplifié de pilote qui est implémenté est composé de cinq agents et de l'agent simulateur qui vient compléter le SMA.

7.4 Discussion

Ce chapitre a exposé un modèle complet de pilote. Ce modèle permet d'intégrer des agents hétérogènes dans un SMA qui est extensible et portable. Le premier objectif fixé au début du chapitre est donc atteint.

Un modèle simplifié de pilote a aussi été présenté. Il définit un SMA à trois niveaux composé d'agents hétérogènes. Ce SMA, basé sur l'environnement GEMAS, est l'application qui sert à valider l'environnement GEMAS et à vérifier que l'architecture du modèle complet est valable. Le deuxième objectif énoncé au début du chapitre, soit la validation de l'environnement GEMAS, est atteint en implémentant le modèle simplifié. Cette implémentation est faite au chapitre 8.

CHAPITRE 8

RÉALISATION DU MODÈLE DE PILOTE

Le chapitre 7 a permis de faire une conception de haut niveau du modèle complet de pilote et de définir un modèle simplifié. Le modèle simplifié est réalisé afin de valider l'architecture du modèle complet, le modèle GAM et l'environnement GEMAS, avec une application multi-agents hétérogènes réelle.

Afin d'alléger la discussion, seul le niveau connaissance des agents réalisés dans un paradigme autre que procédural est inclus. Les parties procédurales réalisées en JAVA peuvent être consultées dans le code de l'application.

8.1 Modèle simplifié de pilote

La figure 8.1 représente le modèle simplifié de pilote. Ce modèle fixe l'architecture et les interactions du SMA hétérogène à implémenter. La section 7.3 expose la conception de chaque agent ; elle définit les paradigmes, spécifie les données qui sont échangées sur les liens de communication et définit la structure interne de chaque agent. Il ne reste qu'à faire l'implémentation de chacun de ces agents avec les outils offerts par l'environnement GEMAS.

8.2 Agent contrôleur

L'agent contrôleur est réalisé à partir de l'agent procédural. Essentiellement, il reçoit les tracés désirés et les vitesses désirées de l'agent traceur et il donne des commandes au véhicule. La réalisation de cet agent consiste en l'implémentation de formules mathématiques réalisant le modèle de commande du véhicule exposé dans [43] et [143] et en la réception de messages servant de données en entrée aux formules mathématiques.

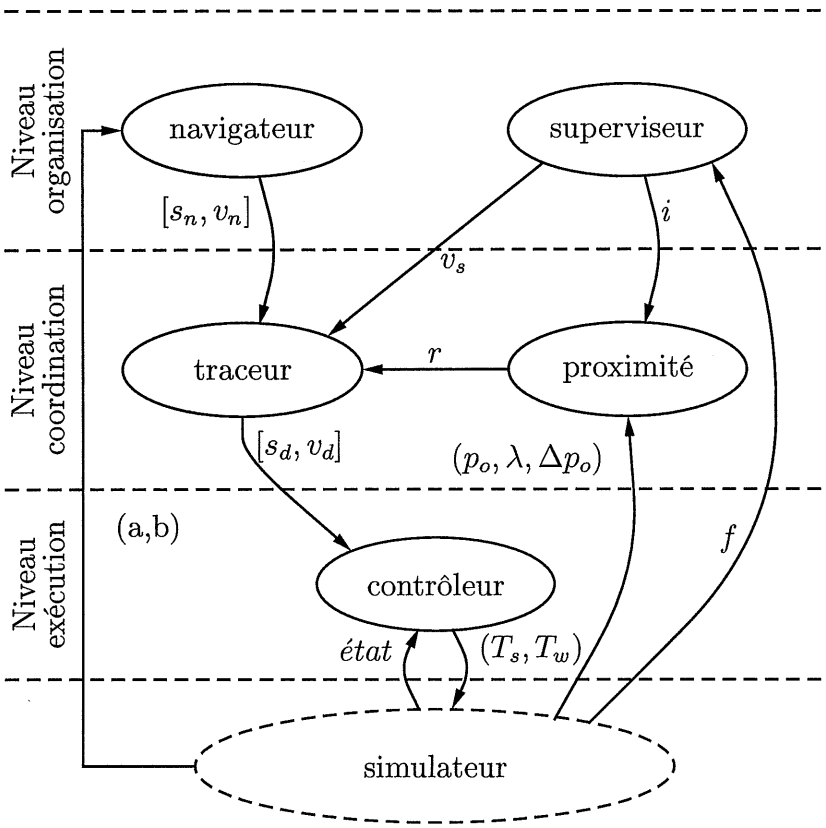


Figure 8.1 – Modèle simplifié du pilote simulé.

Ce modèle de commande utilise une représentation de l'état du véhicule appelée x. Cette représentation est classique pour la conception d'un système de commande de véhicule :

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} \text{direction roue avant } (\delta) \\ \text{dérivée de } x_1 \\ \text{rotation roue avant } (\theta) \\ \text{dérivée de } x_3 \\ \text{position réelle en y} \\ \text{position réelle en x} \\ \text{orientation du véhicule } (\varphi) \end{bmatrix}$$

La figure 8.2 [44] illustre certaines des variables d'état.

Le système de commande comprend un modèle du véhicule et quatre modules pour former une boucle de rétroaction. Ce système est illustré à la figure 4.2 qui est reproduite ici à la figure 8.3. La conception du système de commande est basée sur une méthode classique appelée linéarisation par retour d'état. Les sections suivantes exposent le modèle non-linéaire

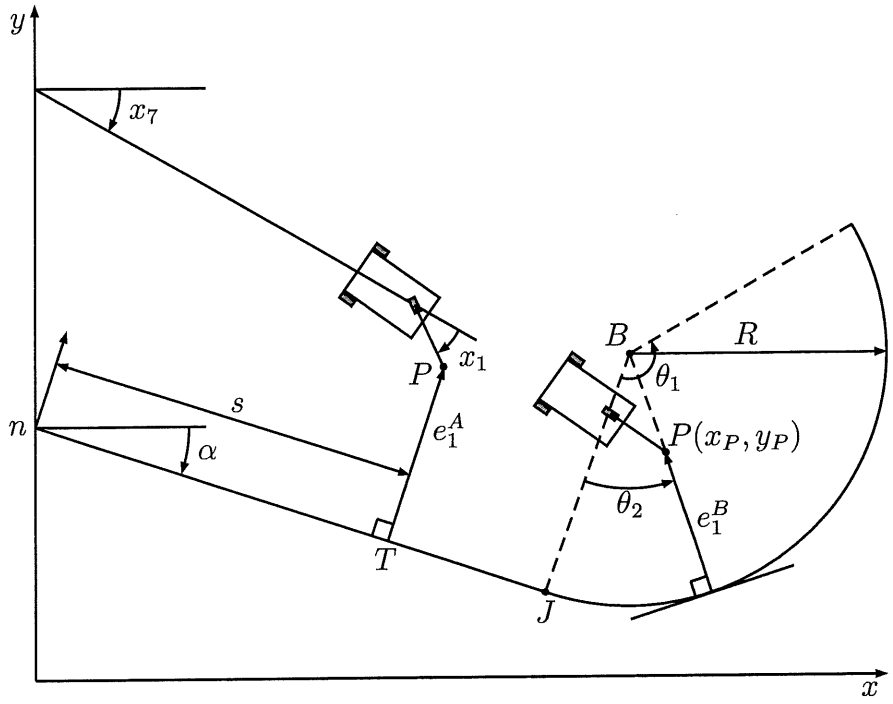


Figure 8.2 – Géométrie des tracés linéaires et circulaires [44].

du véhicule et les quatre modules servant à la commande. Les constantes et les formules intermédiaires sont données à l'annexe A.

8.2.1 Module véhicule

En sachant que $T1$ est le couple appliqué sur le volant et que $T2$ est le couple appliqué sur les roues avant, alors la dérivée $\dot{x}$ de l'état courant x du véhicule est calculée par les équations suivantes :

$$\dot{x} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \\ \dot{x}_7 \end{bmatrix} = \begin{bmatrix} x_2 \\ f_1 x_2 x_4 + g_1 T_1 + g_2 T_2 \\ x_4 \\ f_2 x_2 x_4 + g_2 T_1 + g_3 T_2 \\ x_4(K_1 K_{15} \sin x_1 \cos x_7 - \rho \cos x_1 \sin x_7) \\ x_4(K_1 K_{15} \sin x_1 \sin x_7 + \rho \cos x_1 \cos x_7) \\ K_1 x_4 \sin x_1 \end{bmatrix} \tag{8.1}$$

Des intégrations numériques sont faites pendant la simulation afin de trouver l'état x du véhicule. Les méthodes de Simpson 1/3 et Simpson 3/8 sont utilisées [59].

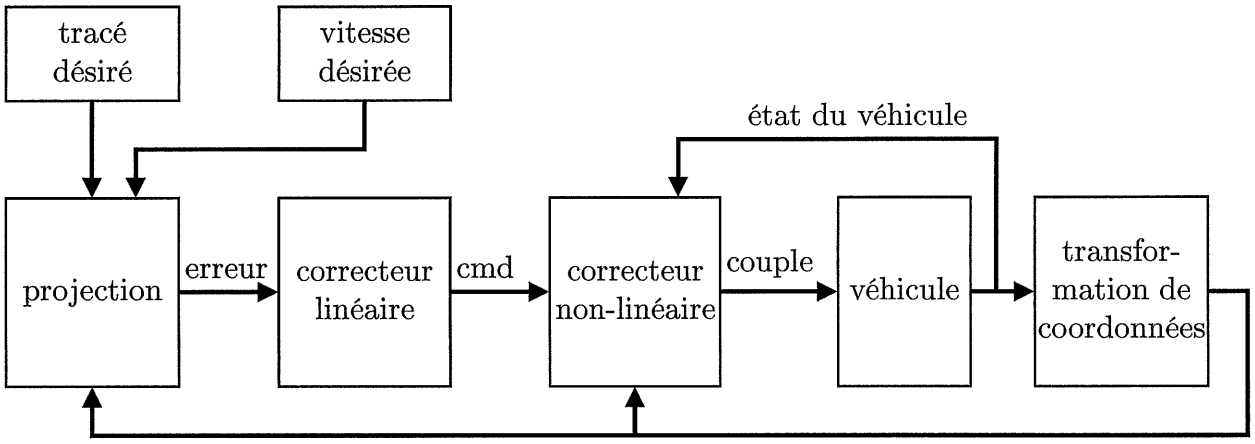


Figure 8.3 – Architecture du système de commande [44].

Ces équations sont implémentées dans l'agent simulateur à la section 8.7, car la dynamique du véhicule est simulée dans cet agent, en accord avec le modèle de pilote.

8.2.2 Module transformation de coordonnées

Ce module exécute la transformation de coordonnées de la position courante du véhicule à la position courante du point de commande P sur la figure 8.2. Il calcule les valeurs de x_P et de y_P et leurs dérivées. Les calculs sont faits à partir de l'état courant x du véhicule :

$$\begin{bmatrix} x_P \\ y_P \\ \dot{x}_P \\ \dot{y}_P \end{bmatrix} = \begin{bmatrix} x_6 + l_1 \cos x_7 + L_p \cos(x_1 + x_7) \\ x_5 - l_1 \sin x_7 - L_p \sin(x_1 + x_7) \\ N_1 x_2 + M_1 x_4 \\ N_2 x_2 + M_2 x_4 \end{bmatrix} \tag{8.2}$$

8.2.3 Module projection

Le module de projection calcule la déviation latérale e_1 du véhicule par rapport au tracé désiré, l'erreur par rapport à la vitesse longitudinale désirée e_2 et la dérivée de la déviation latérale $\dot{e}_1$. Le vecteur des erreurs, composé de e_1, e_2 et $\dot{e}_1$, est calculé différemment selon que le tracé courant est linéaire ou circulaire.

Dans le cas d'un tracé linéaire, le vecteur des erreurs est défini comme suit :

$$erreur = \begin{bmatrix} e_1 \\ e_2 \\ \dot{e}_1 \end{bmatrix} = \begin{bmatrix} x_P \sin\alpha + (y_P - n)\cos\alpha \\ \rho x_4 \cos x_1 - V_d \\ \dot{x}_P \sin\alpha + \dot{y}_P \cos\alpha \end{bmatrix} \tag{8.3}$$

Dans le cas d'un tracé circulaire, le vecteur des erreurs est défini comme suit :

$$erreur = \begin{bmatrix} e_1 \\ e_2 \\ \dot{e}_1 \end{bmatrix} = \begin{bmatrix} R - R_c \\ \rho x_4 \cos x_1 - V_d \\ -\frac{\dot{x}_P \Delta x + \dot{y}_P \Delta y}{R_c} \end{bmatrix} \tag{8.4}$$

Avec :

$$R_c = \sqrt{\Delta x^2 + \Delta y^2} \tag{8.5}$$
$$\Delta x = x_P - x_B \tag{8.6}$$
$$\Delta y = y_P - y_B \tag{8.7}$$

D'un point de vue mathématique, le tracé doit être une fonction continue et dérivable par partie. Aussi, certaines manoeuvres complexes ne peuvent pas être représentées par une seule fonction. Dans ce cas, le tracé doit alors être divisé en plusieurs segments. Ceci n'est pas une limitation, car le tracé désiré se représente bien en une suite de segments constitués de droites et d'arcs de cercle. Une discussion plus détaillée de ce point se trouve dans [44].

8.2.4 Module correcteur linéaire

La commande donnée par le contrôleur linéaire afin de minimiser l'erreur est calculée de la manière suivante :

$$cmd = \begin{bmatrix} \nu_1 \\ \nu_2 \end{bmatrix} = \begin{bmatrix} \eta_1 e_1 + \eta_2 \dot{e}_1 \\ \eta_3 e_2 \end{bmatrix} \tag{8.8}$$

où η_1, η_2 et η_3 sont des constantes servant à obtenir les caractéristiques désirées sur la réponse du système de commande, comme le temps de réponse par exemple.

8.2.5 Module correcteur non-linéaire

Ce module calcule le couple effectivement appliqué sur le véhicule à partir de la commande linéaire. Le calcul se fait différemment selon que la partie courante du tracé est linéaire ou circulaire.

Dans le cas d'un tracé linéaire, le couple est calculé comme suit :

$$couple = \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} = \begin{bmatrix} \frac{\zeta_6(\nu_1-\Psi_1)-\zeta_2(\nu_2-\Psi_3)}{\zeta_1\zeta_6-\zeta_2\zeta_5} \\ \frac{-\zeta_5(\nu_1-\Psi_1)+\zeta_1(\nu_2-\Psi_3)}{\zeta_1\zeta_6-\zeta_2\zeta_5} \end{bmatrix} \tag{8.9}$$

Dans le cas d'un tracé circulaire, le couple est calculé comme suit :

$$couple = \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} = \begin{bmatrix} \frac{\zeta_6(\nu_1-\Psi_2)-\zeta_4(\nu_2-\Psi_3)}{\zeta_3\zeta_6-\zeta_4\zeta_5} \\ \frac{-\zeta_5(\nu_1-\Psi_2)+\zeta_3(\nu_2-\Psi_3)}{\zeta_3\zeta_6-\zeta_4\zeta_5} \end{bmatrix} \tag{8.10}$$

Deux conditions sont imposées afin d'assurer la stabilité du système. La première est que l'angle x_1 de direction de la roue avant soit dans l'intervalle $[-\frac{\pi}{2}, \frac{\pi}{2}]$. La deuxième est que l'axe des roues ne doit pas être parallèle au tracé désiré. La simulation est stoppée et une indication visuelle de l'état invalide est donnée lorsque la première condition n'est pas rencontrée. Cette erreur survient dans l'exemple de condition extrême expliqué à la section 8.8.2.

8.2.6 Remarques

Un trajet d'un point à un autre est divisé en plusieurs segments composés de droites et d'arcs de cercle. Ceci implique que différentes formules doivent être appliquées selon le type du segment courant, comme le montrent les équations servant à calculer l'erreur (équations 8.3 et 8.4) et le couple (équations 8.9 et 8.10). Le critère du passage d'une méthode de calcul à l'autre doit être déterminé. Deux cas sont possibles lorsque le trajet passe d'une droite à un autre segment et lorsqu'il passe d'un arc de cercle à un autre segment.

Dans le cas où le trajet passe d'une droite à un autre segment, le critère de passage est le point J illustré à la figure 8.2. Les paramètres de calcul sont ajustés en accord avec le type du prochain segment lorsque ce point est atteint ou dépassé ($s \geq s_J$).

Dans le cas où le trajet passe d'un arc de cercle à un autre segment, le critère de passage est lié à l'angle θ_1 couvert par l'arc de cercle du segment courant, comme illustré à la figure 8.2. Les paramètres de calcul sont ajustés en accord avec le type du prochain segment lorsque la position courante du véhicule atteint ou dépasse l'angle couvert par l'arc de cercle $(\theta_1 - \theta_2 \leq 0)$.

8.3 Agent traceur

Cet agent est réalisé à partir de l'agent procédural de l'environnement GEMAS. Cet agent reçoit de l'agent navigateur le trajet sélectionné et la vitesse maximale. Étant donné que les librairies de manoeuvres comme le dépassement ne sont pas réalisées, les recommandations reçues de l'agent proximité ne sont pas prises en compte. Les fonctionnalités de l'agent traceur du modèle simplifié de pilote se résument à recevoir des trajets de l'agent navigateur et de communiquer à l'agent contrôleur les tracés calculés et les vitesses désirées qui tiennent compte des limites imposées par l'agent superviseur. L'implémentation en langage JAVA de cet agent peut être consultée dans le code de la réalisation de l'application.

8.4 Agent proximité

Cet agent est basé sur l'agent flou de l'environnement GEMAS. L'agent flou non-esclave est utilisé afin de gérer les messages échangés. L'agent flou non-esclave a été choisi afin de démontrer que des messages de différents types peuvent être traités par un agent flou. L'agent flou non-esclave, en ayant le contrôle sur le traitement de ses messages, peut convertir des messages d'un format quelconque à un format acceptable pour son usage interne. Ainsi, un agent flou peut s'adapter à toutes situations demandant des traitements particuliers de messages.

Le niveau connaissance de l'agent proximité est donné dans les sections qui suivent. Le détail des manipulations des messages est détaillé dans le code JAVA de l'agent.

8.4.1 Configuration du niveau connaissance

Voici le fichier de connaissances de l'agent proximité :

```
@fuzzySets {
  file="proximity.sets";
}

@fuzzyRules{
  file="proximity.rules";
}

@configuration {
  name = "Proximity agent";
  waitForInputs = "true";
  andOperator="min";
  orOperator="max";
  inferenceOperator="min";
  aggregationOperator="max";
  defuzzificationOperator="mom";
}
```

8.4.2 Règles floues

Les règles floues sont en partie issues du bon sens et des lois québécoises sur la conduite automobile. Voici les règles floues de l'agent proximité :

```
IF ObjectCloseness IS Near
AND ObjectOrientation IS Left
AND EnvironmentCondition IS Normal
THEN PassObject IS No

IF ObjectCloseness IS Near
AND ObjectOrientation IS Left
AND EnvironmentCondition IS SoSo
THEN PassObject IS No

IF ObjectCloseness IS Near
AND ObjectOrientation IS Left
AND EnvironmentCondition IS Bad
THEN PassObject IS No

IF ObjectCloseness IS Near
AND ObjectOrientation IS Ahead
AND EnvironmentCondition IS Normal
THEN PassObject IS No

IF ObjectCloseness IS Near
AND ObjectOrientation IS Ahead
AND EnvironmentCondition IS SoSo
THEN PassObject IS No

IF ObjectCloseness IS Near
AND ObjectOrientation IS Ahead
AND EnvironmentCondition IS Bad
THEN PassObject IS No

IF ObjectCloseness IS Near
AND ObjectOrientation IS Right
```

```
AND EnvironmentCondition IS Normal
THEN PassObject IS Yes

IF ObjectCloseness IS Near
AND ObjectOrientation IS Right
AND EnvironmentCondition IS SoSo
THEN PassObject IS Maybe

IF ObjectCloseness IS Near
AND ObjectOrientation IS Right
AND EnvironmentCondition IS Bad
THEN PassObject IS No

IF ObjectCloseness IS Normal
AND ObjectOrientation IS Left
AND EnvironmentCondition IS Normal
THEN PassObject IS No

IF ObjectCloseness IS Normal
AND ObjectOrientation IS Left
AND EnvironmentCondition IS SoSo
THEN PassObject IS No

IF ObjectCloseness IS Normal
AND ObjectOrientation IS Left
AND EnvironmentCondition IS Bad
THEN PassObject IS No

IF ObjectCloseness IS Normal
AND ObjectOrientation IS Ahead
AND EnvironmentCondition IS Normal
THEN PassObject IS Yes

IF ObjectCloseness IS Normal
AND ObjectOrientation IS Ahead
AND EnvironmentCondition IS SoSo
THEN PassObject IS Maybe

IF ObjectCloseness IS Normal
AND ObjectOrientation IS Ahead
AND EnvironmentCondition IS Bad
THEN PassObject IS No

IF ObjectCloseness IS Normal
AND ObjectOrientation IS Right
```

```
AND EnvironmentCondition IS Normal
THEN PassObject IS Yes

IF ObjectCloseness IS Normal
AND ObjectOrientation IS Right
AND EnvironmentCondition IS SoSo
THEN PassObject IS Maybe

IF ObjectCloseness IS Normal
AND ObjectOrientation IS Right
AND EnvironmentCondition IS Bad
THEN PassObject IS No

IF ObjectCloseness IS Far
AND ObjectOrientation IS Left
AND EnvironmentCondition IS Normal
THEN PassObject IS No

IF ObjectCloseness IS Far
AND ObjectOrientation IS Left
AND EnvironmentCondition IS SoSo
THEN PassObject IS No

IF ObjectCloseness IS Far
AND ObjectOrientation IS Left
AND EnvironmentCondition IS Bad
THEN PassObject IS No

IF ObjectCloseness IS Far
AND ObjectOrientation IS Ahead
AND EnvironmentCondition IS Normal
THEN PassObject IS Yes

IF ObjectCloseness IS Far
AND ObjectOrientation IS Ahead
AND EnvironmentCondition IS SoSo
THEN PassObject IS Maybe

IF ObjectCloseness IS Far
AND ObjectOrientation IS Ahead
AND EnvironmentCondition IS Bad
THEN PassObject IS No

IF ObjectCloseness IS Far
AND ObjectOrientation IS Right
```

```
AND EnvironmentCondition IS Normal
THEN PassObject IS Yes

IF ObjectCloseness IS Far
AND ObjectOrientation IS Right
AND EnvironmentCondition IS SoSo
THEN PassObject IS Maybe

IF ObjectCloseness IS Far
AND ObjectOrientation IS Right
AND EnvironmentCondition IS Bad
THEN PassObject IS No
```

8.4.3 Ensembles flous

Les ensembles flous, comme certaines règles floues, sont issus du bon sens et des pratiques de conduite normales. Voici les ensembles flous de l'agent proximité :

```
@fuzzySet {
  name="ObjectCloseness";
  type="input";
  units="m";
  membershipFunction = "Near    0  0  4 8";
  membershipFunction = "Normal  4  8 12 16";
  membershipFunction = "Far    12 16 20 20";
}

@fuzzySet {
  name="ObjectOrientation";
  type="input";
  units="deg";
  membershipFunction = "Left  -90 -90 -45  0";
  membershipFunction = "Ahead -45   0   0 45";
  membershipFunction = "Right  0   45  90 90";
}

@fuzzySet {
  name="EnvironmentCondition";
  type="input";
  units="%";
  membershipFunction = "Normal 0  0  25  50";
  membershipFunction = "SoSo  25 50  50  75";
  membershipFunction = "Bad   50 75 100 100";
}

@fuzzySet {
  name="PassObject";
  type="output";
```

```
  units="";
  membershipFunction = "No    25 25 25 25";
  membershipFunction = "Maybe 50 50 50 50";
  membershipFunction = "Yes   75 75 75 75";
}
```

8.5 Agent navigateur

Cet agent est basé sur l'agent logique de l'environnement GEMAS. L'agent logique non-esclave est utilisé afin d'adapter les communications aux besoins de l'application.

Le paradigme logique implémenté dans cette première version de l'environnement GEMAS ne supporte pas l'ajout et le retrait dynamique de faits et de règles. Ceci est un inconvénient qui est compensé par un traitement procédural palliatif lors de la réception des messages reçus et envoyés par l'agent navigateur. Ce manque a pour effet de réduire significativement le code Prolog du niveau connaissance de l'agent, car la partie dynamique du problème ne peut pas être réalisée. Par contre, ce manque permet aussi de montrer qu'un agent logique peut être adapté selon les besoins à toutes situations en ajoutant des traitements procéduraux particuliers.

Le niveau connaissance de l'agent navigateur est donné dans les sections qui suivent. Le détail des manipulations des messages et des traitements palliatifs est détaillé dans le code JAVA de l'agent.

8.5.1 Configuration du niveau connaissance

Voici le fichier de connaissances de l'agent navigateur :

```
// Prolog source code files
@codeFiles {
  file="destinations.pl";
}
```

8.5.2 Code prolog

Le code prolog permet de trouver un trajet d'une destination à une autre. Un trajet est représenté par une suite de segments. Chaque segment est une droite ou un arc de cercle et est de la forme suivante :

$l, longueur, vitesse$
$cr, rayon, \theta_1, vitesse$
$cl, rayon, \theta_1, vitesse$

Une droite est identifiée par un l (*line*) et par ses paramètres : la longueur en mètres et la vitesse maximale sur ce segment en mètres par seconde. Les arcs de cercle sont identifiés par un cr (*circle right*) ou un cl (*circle left*) et par leurs paramètres : le rayon en mètres, l'angle couvert par l'arc de cercle en degrés et la vitesse maximale sur ce segment en mètres par seconde. La gauche et la droite sont définies par rapport à la direction du véhicule. Une représentation graphique de chacun de ces trajets est montrée lors des tests effectués à la section 8.8. Voici donc le code prolog de l'agent :

```
route(a, b, [l,50,28, l,25,14, cl,65,90,14, cr,65,90,14, l,50,28]).
route(a, c, [cr,65,60,14, l,50,28, l,25,14, cl,65,90,14, l,100,28]).
route(c, d, [l, 30, 21, l, 40, 14, cl, 65, 90, 14, cl, 65, 90, 14,
      cl,65,45,14, l,80,14, cr,65,45,14, cr,65,90,14,
      cr,65,30,14, cr,150,90,21, cr,150,45,21, l,100,28]).
route(b, c, [l,60,28, cr,10,90,28, cl,10,90,28, l,30,28]).

path(X,Y,Path) :- route(X,Y,Path).
```

8.6 Agent superviseur

Cet agent est basé sur l'agent logique de l'environnement GEMAS. L'agent logique non-esclave est utilisé afin d'adapter les communications aux besoins de l'application.

Le traitement des messages est minimal dans le cas de l'agent superviseur. Une adaptation des messages aux règles internes est effectuée afin d'assurer une syntaxe correcte des requêtes.

Le niveau connaissance de l'agent superviseur est donné dans les sections qui suivent. Le détail des manipulations des messages est détaillé dans le code JAVA de l'agent.

8.6.1 Configuration du niveau connaissance

Voici le fichier de connaissances de l'agent superviseur :

```
// Prolog source code files
@codeFiles {
  file="speed.pl";
}
```

8.6.2 Code prolog

Le code prolog permet d'associer l'état de routes avec un facteur de pondération de la vitesse maximale. Ce facteur représente un pourcentage de la vitesse maximale. Ainsi, un facteur de 75 indique que la vitesse maximale suggérée est de 75% de la vitesse maximale en condition normale. Voici donc le code prolog de l'agent :

```
// speelControl(condition, in % of speed limit)
speedControl(normal, 100).
speedControl(wet, 85).
speedControl(snow, 75).
speedControl(ice, 50).
```

8.7 Agent simulateur

Cet agent de type procédural sert d'interface utilisateurs pour l'application. Il utilise une interface graphique afin de permettre à l'usager de contrôler la simulation. L'interface de l'agent simulateur permet de choisir un trajet, de fixer les conditions environnementales et la proximité des véhicules. Elle affiche aussi le trajet désiré sélectionné par l'agent navigateur et le trajet effectif suivi par l'agent contrôleur. L'agent simulateur est un agent qui est basé sur l'agent procédural de l'environnement GEMAS.

Selon les choix de l'usager pour une simulation, l'agent simulateur envoie des messages aux agents concernés. Les choix de trajets de l'usager sont envoyés à l'agent navigateur, les conditions environnementales à l'agent superviseur et le statut de proximité à l'agent proximité. En retour, l'agent simulateur reçoit de l'agent navigateur le trajet sélectionné et de l'agent contrôleur le trajet effectif réellement suivi. Les figures 8.4 à 8.7 illustrent l'interface utilisateurs présentée à l'usager par l'agent simulateur.

8.8 Simulations et résultats

L'application permet de parcourir quatre trajets différents afin de valider l'environnement GEMAS et l'architecture du modèle complet de pilote ; trois sont des parcours réguliers et un quatrième est un parcours à trop haute vitesse afin de dépasser les limites du système de commande et ainsi causer une sortie de route. Ces quatre trajets permettent de valider que chaque agent du modèle de pilote exécute bien le travail qui lui est assigné. Ils permettent aussi de valider les agents spécialisés de l'environnement GEMAS dans un SMA hétérogène réel.

8.8.1 Conditions normales

L'interface utilisateurs de l'agent simulateur permet d'exécuter trois parcours en situation normale. Les trajets sont constitués de lignes droites et d'arcs de cercles à grands rayons, comme sur les autoroutes. Ces tests permettent de valider le système sur différents parcours, à différentes vitesses et dans différentes conditions climatiques afin de démontrer les capacités du système à s'adapter et de démontrer la validité du modèle de pilote dans différentes situations. Les figures 8.4 à 8.6 illustrent des exemples de tests en situation normale.

La figure 8.4 montre une des simulations avec le trajet 1. La simulation est en cours, comme l'indique le texte de la bordure de l'affichage. Dans la section où est dessiné le trajet, la ligne pleine représente le tracé désiré et les points représentent le tracé actuel parcouru par le véhicule. La distance entre les points est un indicateur de la vitesse du véhicule, car le rafraîchissement est fait à intervalle de temps fixe ; plus les points sont éloignés les uns des autres, plus la vitesse est grande.

L'interface utilisateurs montre aussi que les conditions environnementales sont normales et qu'un autre véhicule est relativement près vers la droite. On peut aussi voir que la vitesse courante du véhicule est de $22,9\ m/s$ ($82,6\ km/h$). Le véhicule est en accélération, car le niveau connaissance de l'agent navigateur spécifie une vitesse maximale de $28\ m/s$ ($\approx 100\ km/h$) sur ce segment.

La figure 8.5 montre une image du véhicule une fois le trajet 2 terminé, comme l'indique le texte de la bordure de l'affichage. L'interface montre que les conditions environnementales représentent une chaussée mouillée et qu'un véhicule se trouve à une distance normale vers l'avant. Comme le montre l'interface, la vitesse finale du véhicule est de $23,8m/s$ $(85,6km/h)$, car étant donné que la route est mouillée, l'agent superviseur a suggéré à l'agent navigateur que la vitesse maximale soit de 85% de la vitesse maximale normale.

La figure 8.6 montre une image du véhicule une fois le trajet 3 terminé. Ce trajet montre que le véhicule peut suivre des trajets complexes et à des vitesses variant de 14 à 28 m/s ($\approx$ 50 à 100 km/h).

8.8.2 Conditions extrêmes

Ce test est constitué d'un trajet comportant deux courbes à très petits rayons et avec le véhicule ayant une grande vitesse. Ce test provoque une sortie de route et amène une situation où le système de commande n'est plus valide et la simulation doit alors être stoppée. Ce test permet de montrer deux choses. La première est de montrer que de vraies conditions de conduite automobile sont simulées, car le véhicule quitte la route. La deuxième est de montrer que des modèles traitant les situations extrêmes sont nécessaires lors de l'implantation réelle d'un modèle complet de pilote, car lorsque des conditions où le système de commande n'est plus valide sont rencontrées, la simulation doit être stoppée. Cette situation n'est pas acceptable dans une application réelle. La figure 8.7 illustre un exemple de test en situation extrême.

8.9 Discussion

La réalisation de l'application de ce chapitre a deux objectifs. Le premier est de démontrer la validité des concepts de l'environnement GEMAS annoncés au chapitre 5 et le deuxième est de valider l'architecture du modèle complet de pilote annoncé au chapitre 7.

Pour le premier objectif, une implémentation du modèle simplifié de pilote de la section 7.3 a été réalisée avec des agents hétérogènes de types procédural, logique floue et logique. La validité des concepts énoncés pour l'environnement GEMAS et pour le modèle complet de

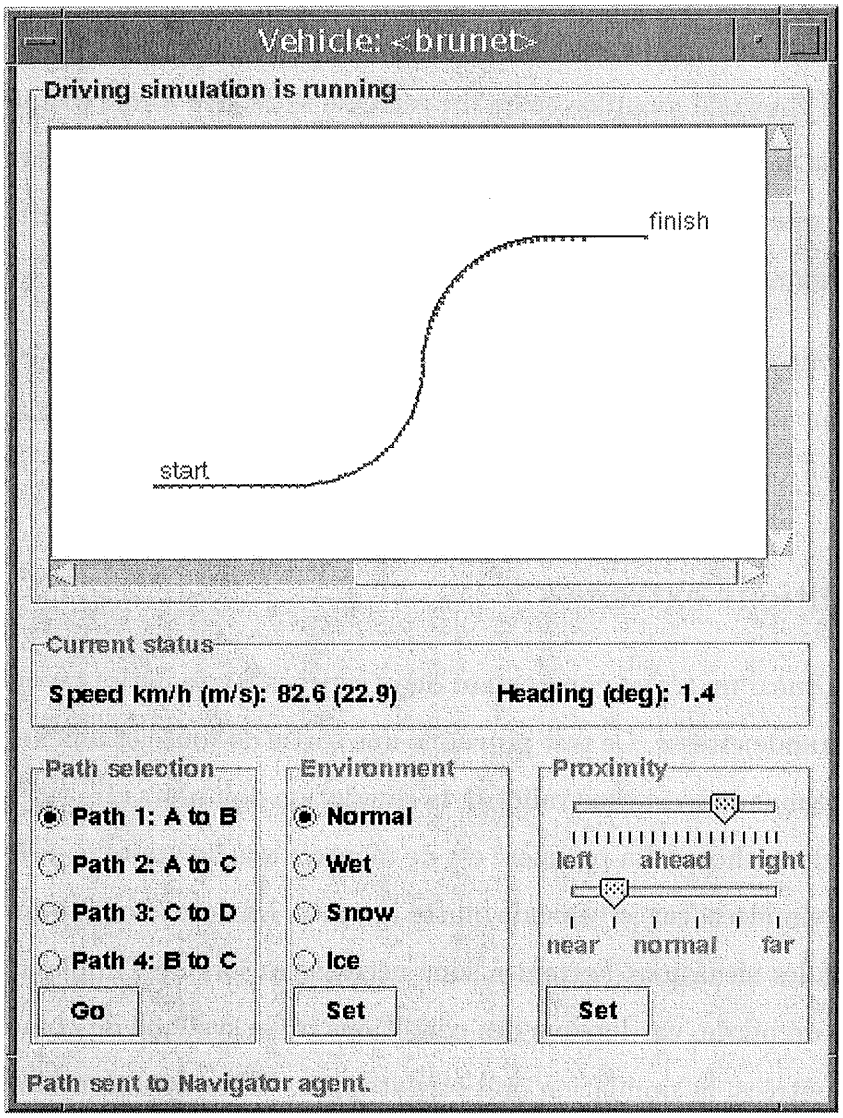


Figure 8.4 – Exemple de simulation en situation normale.

pilote est ainsi démontrée. En effet, les agents spécialisés de l'environnement GEMAS ont réalisé leur tâche respective dans l'application multi-agents réalisée. Le modèle GAM est un modèle générique d'agent qui peut effectivement supporter des agents hétérogènes réels et l'environnement GEMAS atteint donc les objectifs fixés.

Pour le deuxième objectif, le modèle simplifié de pilote accomplit le travail pour lequel il est conçu. Un trajet est sélectionné par l'usager et le véhicule est contrôlé par le système multi-agents afin de respecter le trajet sélectionné et amener le véhicule à la destination désirée. L'architecture du modèle de pilote est donc validée.

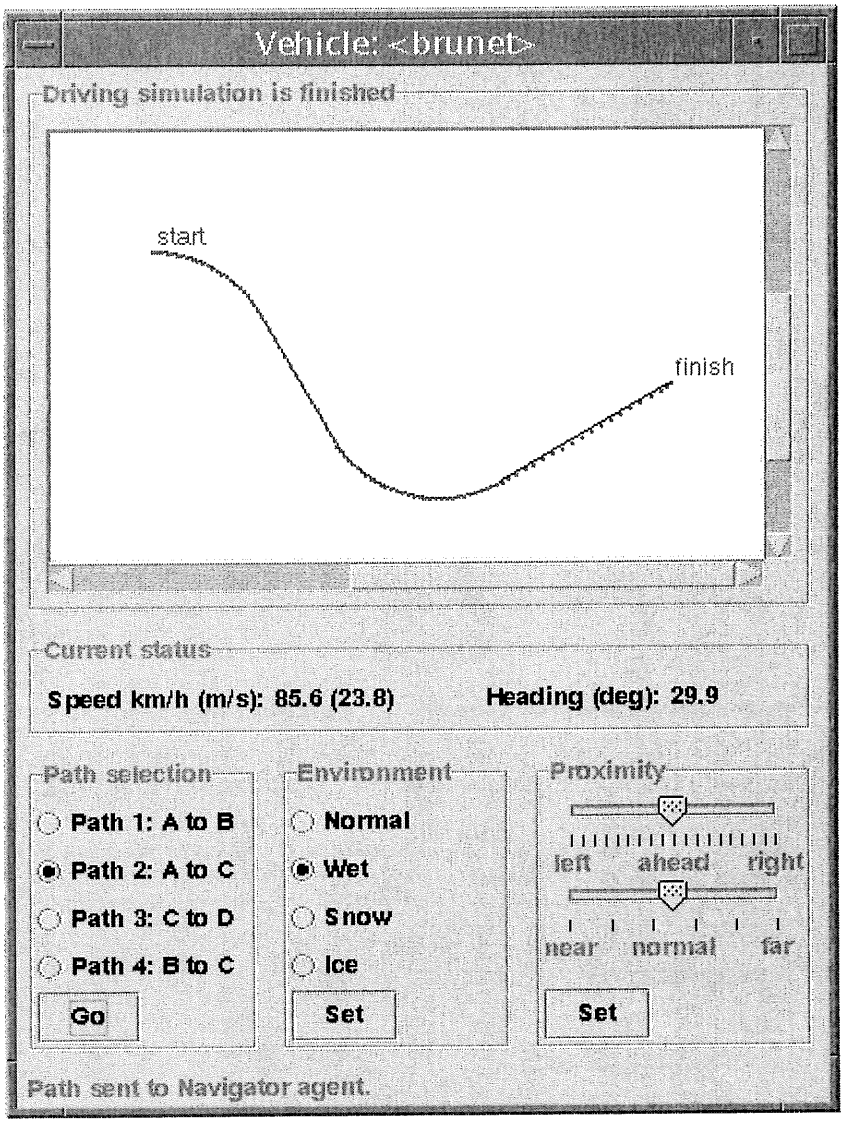


Figure 8.5 – Exemple de simulation avec chaussée mouillée.

Bien que les principaux objectifs soient atteints, la réalisation du modèle simplifié de pilote souffre d'un problème important qui est issu de l'implémentation du paradigme logique. L'ajout et le retrait dynamique de faits n'étant pas offert, la programmation en langage Prolog des agents navigateur et superviseur devient alors très simplifiée. La complexité est transférée dans une partie procédurale afin de contrebalancer ce manque. D'autres faiblesses sont apparentes, comme les suivantes :

- le manque de statut de proximité vers l'arrière qui est une information qui peut affecter significativement les situations de dépassement ;

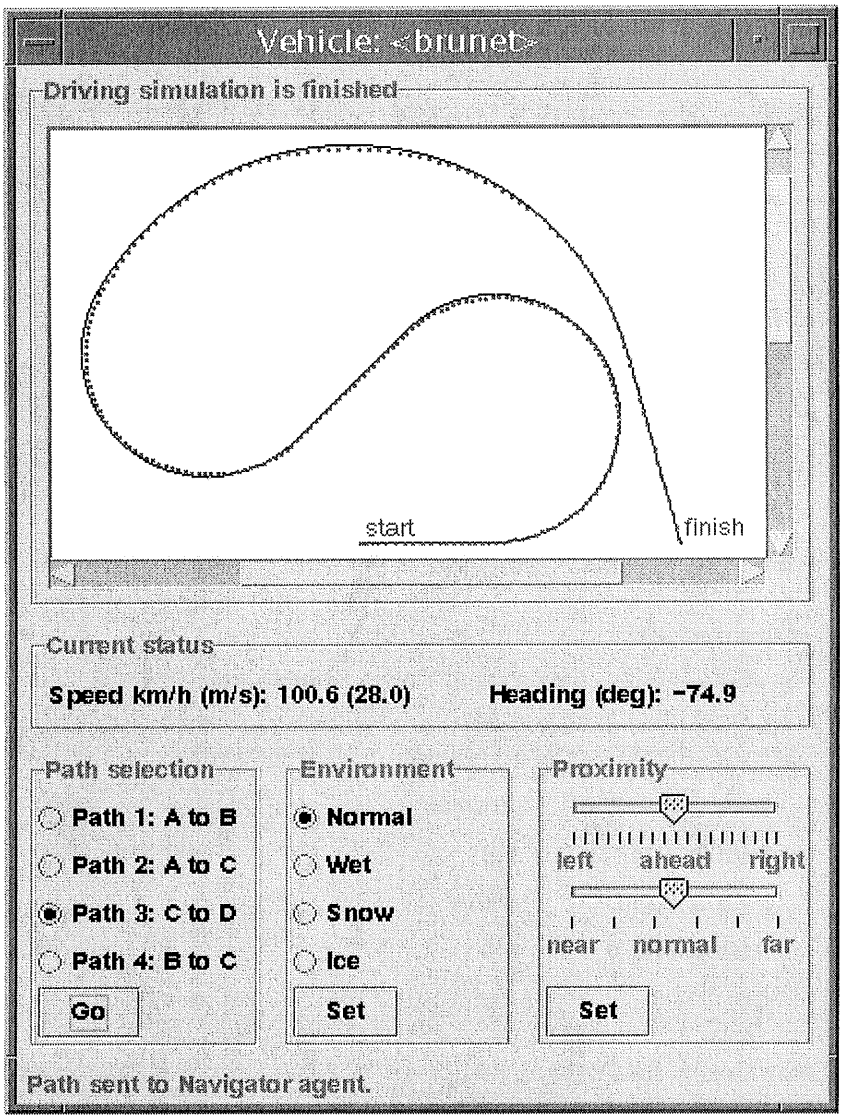


Figure 8.6 – Exemple de simulation avec tracé plus complexe.

– l'affichage des trajets par l'agent simulateur n'est pas assez sophistiqué ; un affichage plus précis du trajet désiré et du trajet réel permettrait de mieux voir les erreurs effectives de la modélisation.

L'application atteint les objectifs principaux pour laquelle elle a été réalisée. En effet, elle démontre que le modèle GAM engendre des agents avec différents paradigmes, que les agents spécialisés de l'environnement GEMAS peuvent être utilisés dans un SMA hétérogène, et que le modèle complet de pilote permet de contrôler un véhicule.

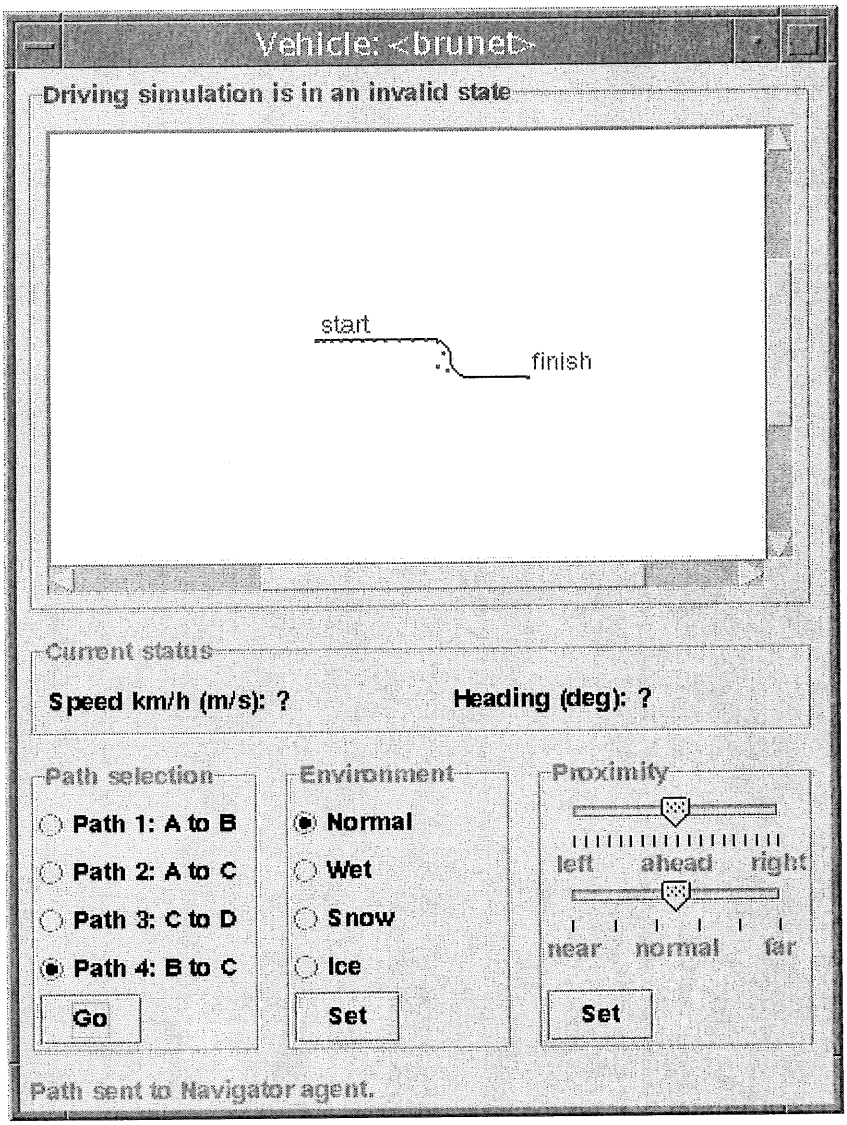


Figure 8.7 – Exemple de simulation en situation extrême.

QUATRIÈME PARTIE

BILAN

CHAPITRE 9

CONCLUSION

9.1 Synthèse

Cette thèse a présenté la définition et la réalisation d'un environnement générique, portable et réutilisable de développement de SMA hétérogènes. L'environnement de développement proposé est nommé GEMAS et il est basé sur un modèle générique d'agent nommé GAM. L'environnement est composé des agents issus du modèle GAM et des outils qui prennent avantage des fonctionnalités intégrées de débogage du modèle GAM.

Le modèle générique d'agent GAM permet d'engendrer des agents hétérogènes et offre des capacités intégrées de débogage. Le modèle GAM est composé de quatre niveaux : communication, interface, moteur et connaissance. Il engendre des agents hétérogènes en instanciant de différentes manières les niveaux supérieurs qui sont les niveaux moteur et connaissance. Un ensemble d'agents basés sur le modèle GAM a été réalisé en utilisant les paradigmes de la logique, de la logique floue, des réseaux de neurones, des algorithmes génétiques et des langages procéduraux.

L'environnement GEMAS comprend deux outils : un outil de collection et de visualisation de traces et un outil d'estampage temporel. Ces outils facilitent le développement d'applications basées sur les SMA et le modèle GAM en offrant des services d'affichage de traces, de configuration dynamique de débogage des agents et des mécanismes standardisés d'estampage temporel des traces.

Les spécifications de l'environnement GEMAS ont été définies au chapitre 5 et leur implémentation ainsi que les tests effectués ont été exposés au chapitre 6. Les tests ont démontré que

le modèle GAM permet d'engendrer des agents hétérogènes et que l'environnement GEMAS permet de développer des SMA hétérogènes.

Cette thèse a aussi présenté des travaux qui ont défini et réalisé une architecture complète, extensible et réutilisable de pilote basée sur les SMA. Cette architecture, exposée au chapitre 7, est composée de trois couches : organisation, coordination et exécution. Elle est inspirée de celle de Saridis [129]. Cette architecture a permis de spécifier un modèle complet de pilote dont une version simplifiée, exposée au chapitre 8, a été réalisée afin de le valider. Cette implémentation a aussi servi à valider le modèle GAM et l'environnement GEMAS avec une application réelle basée sur un SMA hétérogène.

9.2 Bilan

La section 1.3 du chapitre d'introduction a identifié trois défis qui sont mentionnés à plusieurs reprises dans la littérature sur les agents et les SMA :

1. Définir et réaliser un environnement générique de développement de SMA hétérogènes ;
2. Définir un modèle générique d'agent ;
3. Faire l'intégration systématique d'agents hétérogènes dans un système.

Deux buts ont alors été fixés afin de résoudre cette problématique :

1. Définir et réaliser un environnement générique de développement de SMA hétérogènes basé sur un modèle générique d'agent.
2. Définir et réaliser une architecture extensible et réutilisable de pilote basée sur les SMA.

Les spécifications et les concepts élaborés aux chapitres 5 et 7 ainsi que les résultats obtenus aux chapitres 6 et 8 montrent que les deux buts ont été atteints : un modèle générique d'agent nommé GAM a été défini, implémenté et validé ; un environnement générique et portable de développement de SMA hétérogènes, nommé GEMAS, avec des outils de débogage intégrés a été réalisé ; une architecture de pilote a été définie et une implémentation simplifiée a validé sa structure ainsi que l'environnement GEMAS.

Les deux buts ont été atteints et ainsi les trois défis identifiés ont été relevés. Les travaux de recherche effectués contribuent aux domaines des agents, des SMA et de la modélisation de pilotes pour véhicules autonomes et ils sont originaux de plusieurs façons :

- Ils définissent et réalisent un modèle générique d'agent, le modèle GAM, qui permet de générer des agents hétérogènes.
- Ils définissent un modèle générique d'agent (GAM) qui est orienté communication et non pas orienté paradigme, comme les modèles d'agents retrouvés dans la littérature. C'est cette caractéristique fondamentale qui permet d'engendrer des agents hétérogènes.
- Ils conçoivent et réalisent des agents basés sur le modèle GAM avec les paradigmes les plus courants : logique, logique floue, réseaux de neurones artificiels, algorithmes génétiques et langages procéduraux.
- Ils proposent et réalisent un environnement générique et portable de développement de SMA hétérogènes avec des mécanismes intégrés de débogage, l'environnement GEMAS.
- Ils conçoivent et réalisent des outils de débogage de SMA pour l'environnement GEMAS.
- Ils conçoivent et réalisent un modèle extensible et réutilisable de pilote pour véhicules autonomes basé sur les SMA hétérogènes..

9.3 Travaux futurs

L'état actuel du modèle GAM, de l'environnement GEMAS et du modèle de pilote démontre leur validité, et bien que les objectifs fixés soient atteints, des améliorations peuvent être apportées.

Les agents spécialisés de base sont fonctionnels, mais ne sont pas complets. Par exemple, l'agent logique a un interpréteur Prolog qui ne permet pas l'ajout et le retrait dynamique de faits et de règles ; l'agent flou ne permet pas tous les opérateurs dans les règles floues. Conceptuellement, ces agents sont complets, mais l'implémentation ne l'est pas.

Les agents spécialisés évolués ne sont pas implémentés ; ils n'étaient pas nécessaires à la démonstration. L'environnement GEMAS doit fournir ces agents afin qu'ils puissent être utilisés directement par les usagers et qu'il soit mieux apprécié par la communauté informatique.

Le modèle GAM prévoit la communication primitive, mais elle n'a pas été réalisée, car elle n'était pas nécessaire pour démontrer la validité du modèle GAM et de l'environnement GEMAS. De plus, les applications réalisées n'ont pas eu besoin de ce type de communication. Par contre, les interfaces pour ce type de communication devront être offertes et réalisées.

Les communications évoluées sont implémentées dans leur forme la plus simple. Des protocoles de communication plus évolués doivent être ajoutés afin de libérer les concepteurs de leur implémentation. Par exemple, un langage agent et des protocoles de négociation et de collaboration doivent faire partie de l'ensemble des outils disponibles dans l'environnement GEMAS.

Les agents à personnalités multiples (APM) ouvrent une nouvelle avenue dans la conception d'agents et dans la résolution de problèmes. Les APM peuvent mener à de nouveaux types d'agents intelligents qui changent dynamiquement de paradigme selon les besoins et la tâche à accomplir. Les APM méritent une investigation plus poussée.

Les outils de l'environnement GEMAS peuvent aussi être améliorés. Les outils de débogage offrent des fonctionnalités intéressantes, mais ils doivent être complétés. Par exemple, l'outil de visualisation affiche les traces dans l'ordre reçu ; des mécanismes de tri et de filtrage seraient très utiles. Ces fonctionnalités auraient par ailleurs été appréciées lors du développement de l'application de pilote. Aussi, un outil de récupération de traces locales doit être implémenté, car récupérer manuellement les traces sur différentes machines est une tâche fastidieuse.

De la même manière, le modèle de pilote implémenté doit être plus complet. La version implémentée montre que la structure du modèle complet est valable et que le modèle GAM et l'environnement GEMAS sont valides. Par contre, l'implémentation d'une version plus complète sur un vrai véhicule autonome permettrait de valider l'environnement et le modèle de manière plus poussée et de comparer leurs performances.

Toutes ces améliorations permettraient au modèle GAM, à l'environnement GEMAS et au modèle de pilote d'être encore plus intéressants et complets. Les versions réalisées du modèle GAM et de l'environnement GEMAS dans le contexte de cette recherche ne font qu'ouvrir la voie.

CINQUIÈME PARTIE

ANNEXES

ANNEXE A

COMPLÉMENT MATHÉMATIQUE DE L'AGENT CONTRÔLEUR

A.1 Constantes

Les constantes sont représentatives d'une voiture de type compact à traction avant. Elles sont très semblables à celles d'une Honda Accord 1980 [44].

$$\eta_1 = -16/9 \qquad \eta_2 = -8/3 \qquad \eta_3 = -4/3$$
$$\rho = 0.33m \qquad l_1 = 1.41m \qquad l_a = 2.82m$$
$$I_{C1} = 0.1kg * m^2 \qquad I_{C3} = 0.1455kg * m^2 \qquad I_{A3} = 2232kg * m^2$$
$$m_A = 1416kg \qquad m_C = 5kg \qquad L_p = 1m$$
$$K_1 = \rho / l_a \qquad K_2 = l_1^2 - 2l_a l_1 \qquad K_3 = K_1^2 K_2$$
$$K_4 = K_1 I_{C3} \qquad K_5 = K_3 m_A \qquad K_6 = K_1^2 (I_{C3} + I_{A3})$$
$$K_7 = I_{C1} + \rho^2 (m_A + m_C) \qquad K_8 = K_1^2 m_A l_1 (l_1 - 2l_a) + K_6 \qquad K_9 = K_5 + K_6$$
$$K_{10} = -K_4^2 + K_9 I_{C3} \qquad K_{11} = I_{C3} K_7 \qquad K_{12} = K_4 (K_8 - K_9)$$
$$K_{13} = -K_4 K_7 \qquad K_{14} = K_4^2 - K_8 I_{C3} \qquad K_{15} = l_1 - l_a$$
$$K_{16} = l_1 / l_a \qquad K_{17} = 1 / l_a$$

A.2 Formules

En tenant compte des équations 8.2 et 8.5 à 8.7 :

$$detM = K_{10}\sin^2 x_1 + K_{11} \tag{A.1}$$

$$f_1 = \frac{(K_{12}\sin^2 x_1 + K_{13})\cos x_1}{detM} \tag{A.2}$$

$$f_2 = \frac{K_{14}\sin x_1 \cos x_1}{detM} \tag{A.3}$$

$$g_1 = \frac{K_9 \sin^2 x_1 + K_7}{detM} \tag{A.4}$$

$$g_2 = -\frac{K_4 \sin x_1}{detM} \tag{A.5}$$

$$g_3 = \frac{I_{C3}}{detM} \tag{A.6}$$

$$h_1 = K_{17}(1+\tan^2 x_1)(2x_2^2 \tan x_1 + f_1 x_2 x_4) \tag{A.7}$$

$$h_2 = \rho x_2 x_4 (f_2 \cos x_1 - \sin x_1) \tag{A.8}$$

$$M_1 = -\rho \sin x_1 \sin x_7 + \rho \cos x_1 \cos x_7 - L_p K_1 \sin x_1 \sin(x_1 + x_7) \tag{A.9}$$

$$M_2 = -\rho \sin x_1 \cos x_7 - \rho \cos x_1 \sin x_7 - L_p K_1 \sin x_1 \cos(x_1 + x_7) \tag{A.10}$$

$$N_1 = -L_p \sin(x_1 + x_7) \tag{A.11}$$

$$N_2 = -L_p \cos(x_1 + x_7) \tag{A.12}$$

$$I_{11} = M_1 g_2 + N_1 g_1 \tag{A.13}$$

$$I_{12} = M_1 g_3 + N_1 g_2 \tag{A.14}$$

$$I_{21} = M_2 g_2 + N_2 g_1 \tag{A.15}$$

$$I_{22} = M_2 g_3 + N_2 g_2 \tag{A.16}$$

$$\Psi_1 = h_1 \sin\alpha + h_2 \cos\alpha \tag{A.17}$$

$$\Psi_2 = -\frac{h_1 \Delta x + h_2 \Delta y}{R_c} - \frac{[\dot{x}_P \Delta y + \dot{y}_P \Delta x]^2}{R_c^3} \tag{A.18}$$

$$\Psi_3 = -\rho x_2 x_4 (\sin x_1 - f_2 \cos x_1) \tag{A.19}$$

$$\zeta_1 = I_{11} \sin\alpha + I_{21} \cos\alpha \tag{A.20}$$

$$\zeta_2 = I_{12} \sin\alpha + I_{22} \cos\alpha \tag{A.21}$$

$$\zeta_3 = -\frac{I_{11}\Delta x + I_{21}\Delta y}{R_c} \tag{A.22}$$

$$\zeta_4 = -\frac{I_{12}\Delta x + I_{22}\Delta y}{R_c} \tag{A.23}$$

$$\zeta_5 = \rho g_2 \cos x_1 \tag{A.24}$$

$$\zeta_6 = \rho g_3 \cos x_1 \tag{A.25}$$

BIBLIOGRAPHIE

[1] AGRE, P.E. CHAPMAN, D. Pengi : an implementation of a theory of activity. *Proceedings of the 6th National Conference on Artificial Intelligence*, pages 268–72. Morgan Kaufmann, 1987.

[2] ALBUS, J.S. Outline for a theory of intelligence. *IEEE Transactions on Systems, Man and Cybernetics*, volume 21, numéro 3, pages 473–509, 1991.

[3] ALBUS, J.S. RCS : a reference model architecture for intelligent systems. *Working notes : AAAI 1995 Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents*, 1995.

[4] ALBUS, J.S. The NIST Real-time Control System (RCS) : an approach to intelligent systems research. *Journal of Experimental and Theoretical Artificial Intelligence*, volume 9, numéro 2-3, pages 157–74, 1997.

[5] ANDERSON, J. EVANS, M. A generic simulation system for intelligent agent designs. *Applied Artificial Intelligence*, volume 9, numéro 5, pages 525–60, 1995.

[6] ARNOLD, K., GOSLING, J., HOLMES, D. *The Java Programming Language.* Addison-Wesley, troisième édition, 624 pages, 2000.

[7] ASTOLFI, A. Exponential stabilization of a car-like vehicle. *IEEE International Conference on Robotics and Automation*, pages 1391–1396, 1995.

[8] AUBE, M. SENTENI, A. Emotions as commitments operators : a foundation for control structure in multi-agents systems. Van de VELDE, W. PERRAM, J.W., éditeurs, *MAAMAW'96 : Agents Breaking Away : Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 13–25. Springer-Verlag, 1996.

[9] BAUMGARTNER, E.T. SKAAR, S.B. An autonomous vision-based modile robot. *IEEE Transactions on Automatic Control*, volume 39, numéro 3, pages 493–502, mars 1994.

[10] BELLIFEMINE, F. POGGI, A. JADE - a FIPA-compliant agent framework. *PAAM'99 : Proceedings of the Fourth International Conference on the Practical Applications of Intelligent Agents and Multi-Agents*, pages 97–108. Practical Application Company, 1999.

[11] BENDER, E.A. *Mathematical methods in artificial intelligence.* IEEE Computer Society Press, 664 pages, 1996.

[12] BOLLELLA, G., GOSLING, J., BROSGOL, B.M., DIBBLE, P., FURR, S., HARDIN, D., TURNBULL, M. *The Real-Time Specification for Java.* Addison-Wesley, 224 pages, 2000.

[13] BONASSO, R.P., KORTENKAMP, D., MILLER, D.P., SLACK, M. Experiences with an architecture for intelligent, reactive agents. WOOLDRIDGE, M., MÜLLER, J.-P.,

TAMBE, M., éditeurs, *Intelligent Agents II – Agent Theories, Architectures, and Languages – Proceedings of IJCAI'95-ATAL Workshop*, volume 1037, *LNAI*, pages 187–202. Springer-Verlag, 1996.

[14] BOOCH, G., RUMBAUGH, J., JACOBSON, I. *The Unified Modeling Language User Guide.* Addison-Wesley, 512 pages, 1999.

[15] BOURON, T., FERBER, J., SAMUEL, F. MAGES : a multi-agent testbed for heterogeneous agents. DEMAZEAU, Y. MÜLLER, J.-P., éditeurs, *MAAMAW'90 : Proceedings of the Second European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 195–214. North-Holland, 1991.

[16] BRADY, M. HU, H. The mind of a robot. *Philosophical Transactions of the Royal Society - Series A*, volume 349, numéro 1689, pages 15–27, 1994.

[17] BROOKS, R.A. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, volume 2, numéro 1, pages 14–23, mars 1986.

[18] BROOKS, R.A. Elephants don't play chess. MAES, P., éditeur, *Designing Autonomous Agents*, pages 3–15. MIT Press, 1990.

[19] BROOKS, R.A. Intelligence without reason. *IJCAI'91 : Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 569–95, 1991.

[20] BROOKS, R.A. Intelligence without representation. *Artificial Intelligence*, volume 47, pages 139–159, 1991.

[21] BRUNET, C.-A. *Contribution à l'intégration de la programmation procédurale et de la programmation logique : le langage COP.* Mémoire de maîtrise, Université de Sherbrooke, Sherbrooke, Québec, Canada, 104 pages, 1994.

[22] BRUNET, C.-A. GONZALEZ-RUBIO, R. COP : a simple way to integrate imperative programming and declarative programming. *CCECE'95 : Proceedings of the Canadian Conference on Electrical and Computer Engineering*, pages 1034–1037, septembre 1995.

[23] BRUNET, C.-A., GONZALEZ-RUBIO, R., TÉTREAULT, M. A multi-agent architecture for a driver model for autonomous road vehicle. *CCECE'95 : Proceedings of the Canadian Conference on Electrical and Computer Engineering*, pages 772–775, septembre 1995.

[24] BRUNET, C.-A., GONZALEZ-RUBIO, R., TÉTREAULT, M. Vers un modèle complet de pilote pour véhicules autonomes : une approche basée sur les systèmes multi-agents hétérogènes. *ICIA'99 : Proceedings of the Third International Conference on Industrial Automation*, pages 19.1–4, juin 1999.

[25] BÜRCKERT, H.-J. MÜLLER, J. RATMAN : Rational agents testbed for multi agent networks. DEMAZEAU, Y. MÜLLER, J.-P., éditeurs, *MAAMAW'90 : Proceedings of the Second European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 217–230. North-Holland, 1991.

[26] BUSETTA, P., RÖNNQUIST, R., HODGSON, A., LUCAS, A. JACK intelligent agents - components for intelligent agents in Java. Rapport technique 1, Agent Oriented Software, Melbourne, Australia, 8 pages, 1999.

[27] CASTELFRANCHI, C. To be or not to be an agent. MÜLLER, J.-P., WOOLDRIDGE, M.J., JENNINGS, N.R., éditeurs, *Intelligent Agents III – Agent Theories, Architectures, and*

Languages – Proceedings of ECAI'96 Workshop (ATAL), volume 1193, *LNAI*, pages 37–9. Springer-Verlag, 1997.

[28] CHAIB-DRAA, B. Industrial applications of distributed AI. *Communications of the ACM*, volume 38, numéro 11, pages 49–53, 1995.

[29] CHAIB-DRAA, B. LEVESQUE, P. Hierarchical model and communication by signs, signals, and symbols in multi-agent environments. PERRAM, J.W. MÜLLER, J.-P., éditeurs, *MAAMAW'94 : Distributed Software Agents and Applications : Proceedings of the 6th European Workshop on Modelling Autonomous Agents in Multi-Agent World*, pages 150–65. Springer-Verlag, 1996.

[30] CLARK, K.L. GREGORY, S. PARLOG : parallel programming in logic. *ACM Transactions on Programming Languages*, volume 8, pages 1–49, 1986.

[31] CLOCKSIN, W.F. MELLISH, C.S. *Programming in Prolog.* Springer-Verlag, 4^{e} édition, 281 pages, 1994.

[32] COLLIS, J., NDUMU, D., NWANA, H., LEE, L. The ZEUS agent building tool-kit. *BT Technology Journal*, volume 16, numéro 3, pages 60–68, juillet 1998.

[33] D'ANDRÉA-NOVEL, B., CAMPION, G., BASTIN, G. Control of nonholonomic wheeled mobile robots by state feedback linearisation. *The International Journal of Robotic Research*, volume 14, numéro 6, pages 543–559, 1995.

[34] DAVIS, R. SMITH, R.G. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, volume 20, numéro 1, pages 63–109, 1983.

[35] DECASTRO, A., SPIELMAN, J., DUMA, J. FUDGE : FUzzy Development and Generation Environment. http ://www.mot.com/pub/SPS/MCU/fuzzy, 1994.

[36] DENG, Z. BRADY, M. Dynamic tracking of a wheeled mobile robot. *International Conference on Intelligent Robots and Systems*, volume 2, pages 1295–1298, 1993.

[37] DENNETT, D.C. *The Intentional Stance.* MIT Press, 400 pages, 1989.

[38] DESANTIS, R.M. Path-tracking for car-like robots with single and double steering. *IEEE Transactions on Vehicular Technology*, volume 44, numéro 2, pages 366–377, 1995.

[39] DONGES, E. A two-level model of driver steering behavior. *Human Factors*, volume 20, numéro 6, pages 691–707, décembre 1978.

[40] DURFEE, E.H., LESSER, V.R., CORKILL, D.D. Coherent cooperation among communicating problem solvers. BOND, A.H. GASSER, L., éditeurs, *Readings in Distributed Artificial Intelligence*, pages 268–284. Morgan-Kaufmann, 1988.

[41] DURFEE, E.H., LESSER, V.R., CORKILL, D.D. Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering*, volume 1, numéro 1, pages 63–83, mars 1989.

[42] DURFEE, E.H. MONTGOMERY, T.A. MICE : a flexible testbed for intelligent coordination experiments. *Proceedings of the Ninth Workshop on Distributed Artificial Intelligence*, pages 63–83, 1989.

[43] EHSANI, S.M. TÉTREAULT, M. Supervision task in a driver model. *Proceedings of the AIAA Guidance, Navigation and Control Conference*, pages 1835–1843, août 1995.

[44] EHSANI, S.M., TÉTREAULT, M., GOU, M., LAFONTAINE, J. De. Geometric lateral-offset tracking and speed control of a car-like mobile robot. À être publié.

[45] FARHOODI, F. GRAHAM, I. A practical approach to designing and building intelligent software agents. *PAAM'96 : Proceedings of the First International Conference on the Practical Applications of Intelligent Agents and Multi-Agents*, pages 181–204. Practical Application Company, 1996.

[46] FERBER, J. JACOPIN, E. The framework of ECO-problem solving. DEMAZEAU, Y. MÜLLER, J.-P., éditeurs, *MAAMAW'90 : Proceedings of the Second European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 181–193. North-Holland, 1991.

[47] FERGUSON, I.A. On the role of BDI modeling for integrated control and coordinated behavior in autonomous agents. *Applied Artificial Intelligence*, volume 9, numéro 4, pages 421–47, 1995.

[48] FININ, T., LABROU, Y., MAYFIELD, J. KQML as an agent communication language. BRADSHAW, J.M., éditeur, *Software Agents*, chapitre 14, pages 291–316. MIT Press, 1997.

[49] FIPA. FIPA ACL message structure specification. Specification document XC00061, Foundation for Intelligent Physical Agents, 8 pages, 2000. http ://www.fipa.org/.

[50] FIPA. FIPA agent management specification. Specification document XC00023, Foundation for Intelligent Physical Agents, 2000. http ://www.fipa.org/.

[51] FISCHER, K., MÜLLER, J.-P., PISCHEL, M. Unifying control in a layered agent architecture. *ICMAS'95 : Proceedings of the First International Conference on Multi-Agent Systems*, page 446. AAAI Press, 1995.

[52] FISCHER, K., MÜLLER, J.-P., PISCHEL, M. A pragmatic BDI architecture. WOOLDRIDGE, M., MÜLLER, J.-P., TAMBE, M., éditeurs, *Intelligent Agents II – Agent Theories, Architectures, and Languages – Proceedings of IJCAI'95-ATAL Workshop*, volume 1037, *LNAI*, pages 203–18. Springer-Verlag, 1996.

[53] FRANKLIN, S. GRAESSER, A. Is it an agent, or just a program ? : a taxonomy for autonomous agents. MÜLLER, J.-P., WOOLDRIDGE, M.J., JENNINGS, N.R., éditeurs, *Intelligent Agents III – Agent Theories, Architectures, and Languages – Proceedings of ECAI'96 Workshop (ATAL)*, volume 1193, *LNAI*, pages 21–35. Springer-Verlag, 1997.

[54] GARROTT, W.R., WILSON, D.L., SCOTT, R.A. Closed loop automobile manoeuvres using preview-predictor models. *SAE paper 820305*, 1982.

[55] GASSER, L., BRAGANZA, C., HERMAN, N. MACE : a flexible testbed for distributed AI research. HUHNS, M.N., éditeur, *Distributed Artificial Intelligence (Research Notes in Artificial Intelligence)*, chapitre 5, pages 119–152. Morgan-Kaufmann, 1987.

[56] GENESERETH, M.R. KETCHPEL, S.P. Software agents. *Communications of the ACM*, volume 37, numéro 7, pages 48–53, 1994.

[57] GEORGEFF, M.P. dMARS technical overview. Australian Artificial Intelligence Institute, 1996. http ://www.aaii.oz.au/proj/dmars_tech_overview/dMARS-1.html.

[58] GEORGEFF, M.P. LANSKY, A.L. Reactive reasoning and planning. *AAAI'87 : Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 677–682, 1987.

[59] GERALD, C.F. WHEATLEY, P.O. *Applied Numerical Analysis.* Addison-Wesley, 6e édition, 768 pages, 1999.

[60] GUESSOUM, Z. DOJAT, M. A real-time agent model in an asynchronous-object environment. Van de VELDE, W. PERRAM, J.W., éditeurs, *MAAMAW'96 : Agents Breaking Away : Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 190–203. Springer-Verlag, 1996.

[61] GUHA, R.V. LENAT, D.B. Enabling agents to work together. *Communications of the ACM*, volume 37, numéro 7, pages 127–142, 1994.

[62] HAYES-ROTH, B., PFLEGER, K., LALANDA, P., MORIGNOT, P., BALABANOVIC, M. A domain-specific software architecture for adaptive intelligent systems. *IEEE Transactions on Software Engineering*, volume 21, numéro 4, pages 288–301, 1995.

[63] HERRIN, G.D. An empirical model for automobile driver horizontal curve negotiation. *Human Factors*, volume 16, numéro 2, pages 129–133, 1974.

[64] HESSBURG, T., PENG, H., TOMIZUKA, M. An experimental study on lateral control of a vehicle. *Proceedings of the 1991 American Control Conference*, volume 3, pages 3084–3089, juin 1991.

[65] HEWITT, C. Viewing control structures as patterns of passing messages. *Artificial Intelligence*, volume 8, numéro 3, pages 323–364, 1977.

[66] HEXMOOR, H., KORTENKAMP, D., HORSWILL, I. Software architectures for hardware agents. *Journal of Experimental and Theoretical Artificial Intelligence*, volume 9, numéro 2-3, pages 147–56, 1997.

[67] HOLLAND, J.H. *Adaptation in Natural and Artificial Systems.* MIT Press, deuxième édition, 211 pages, 1992.

[68] HOLLAND, J.H., HOLYOAK, K.J., NISBETT, R.E., THAGARD, P.R. *Induction : Processes of Inference, Learning and Discovery.* MIT Press, 385 pages, 1986.

[69] HU, H., BRADY, M., DU, F., PROBERT, P. A transputer network for real-time control of an intelligent mobile robot. *Proceedings of the IEEE Conference on Control Application*, volume 3, pages 1049–1054, 1994.

[70] HUBER, M.J. JAM : A BDI-theoric mobile agent architecture. *Agents'99 : Proceedings of the Third International Conference on Autonomous Agents*, pages 236–243, may 1999.

[71] HÜGLI, H., TIÈCHE, F., FACCHINETTI, C. Integration of vision in autonomous mobile robotics. *Proceedings of IEEE International Symposium on Systems, Man and Cybernetics*, volume 1, pages 1047–1052, 1994.

[72] HUHNS, M.H. SINGH, M.P. Agents and multiagents systems : Themes, approaches, and challenges. HUHNS, M.H. SINGH, M.P., éditeurs, *Readings in agents*, chapitre 1, pages 1–23. Morgan-Kaufmann, 1998.

[73] IGLESIAS, C.A., GONZALEZ, J.C., VELASCO, J.R. MIX : a general purpose multiagent architecture. WOOLDRIDGE, M., MÜLLER, J.-P., TAMBE, M., éditeurs, *Intelligent Agents II – Agent Theories, Architectures, and Languages – Proceedings of IJCAI'95-ATAL Workshop*, volume 1037, *LNAI*, pages 251–66. Springer-Verlag, 1996.

[74] JAGANNATHAN, S., ZHU, S.Q., LEWIS, F.L. Path planning and control of a mobile base with nonholonomic constraints. *Robotica*, volume 12, pages 529–539, 1994.

[75] JENNINGS, N. *Cooperation in Industrial Multi-Agent Systems*, volume 43, *Series in Computer Science*. World Scientific, 188 pages, 1994.

[76] JENNINGS, N.R. WOOLDRIDGE, M. Applying agent technology. *Applied Artificial Intelligence*, volume 9, numéro 4, pages 357–69, 1995.

[77] JENSEN, K., WIRTH, N., MICKEL, A.B., MINER, J.F. *Pascal user manual and report.* Springer-Verlag, 4^{e} édition, 266 pages, 1991.

[78] JOHANSEN, D., van RENESSE, R., SCHNEIDER, F.B. Operating system support for mobile agents. HUHNS, M.H. SINGH, M.P., éditeurs, *Readings in agents*, pages 263–266. Morgan-Kaufmann, 1998.

[79] KAGEYAMA, I. PACEJKA, H.B. On a new driver model with fuzzy control. *Symposium of the 12th IAVSD*, pages 314–324, 1991.

[80] KAY, A. Computer software. *Scientific American*, volume 251, numéro 3, pages 53–59, septembre 1984.

[81] KEHTARNAVAZ, N. SOHN, W. Steering control of autonomous vehicles by neural networks. *Proceedings of the 1991 American Control Conference*, pages 3096–3101, 1991.

[82] KERNIGHAN, B.W. RITCHIE, D.M. *The C Programming Language.* Prentice-Hall, deuxième édition, 274 pages, 1988.

[83] KORNHAUSER, A.L. Neural network approaches for lateral control of autonomous highway vehicles. *Proceedings of the Vehicle Navigation and Information Systems Conference*, pages 1143–1151, octobre 1991.

[84] KRAMER, U. A model of driver behaviour. *Ergonomics*, volume 25, numéro 10, pages 891–907, 1982.

[85] KRAMER, U. On the application of fuzzy sets to the analysis of the system driver-vehicle-environment. *Automatica*, volume 21, numéro 1, pages 101–107, 1985.

[86] KRAMER, U. ROHR, G. A fuzzy model of driver behaviour : Computer simulation and experimental results. *Proceedings of the IFAC Conference on Man-Machine Systems*, pages 31–35, 1982.

[87] KROLL, C.V. Preview-predictor model of driver behavior in emergency situations. *Highway Research Record*, volume 364, pages 16–26, 1971.

[88] LEFEBVRE, D.R. SARIDIS, G.N. A computer architecture for intelligent machines. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2745–2750, mai 1992.

[89] LESSER, V.R. CORKILL, D.D. The distributed vehicle monitoring testbed : a tool for investigating distributed problem solving networks. *Artificial Intelligence*, volume 4, numéro 3, pages 15–33, 1983.

[90] LUBIN, J.M., HUBER, E.C., GILBERT, S.A., KORNHAUSER, A.L. Analysis of a neural network lateral controler for an autonomous road vehicle. *Proceedings of the Future Transportation Technology Conference and Exposition*, pages 23–44, août 1992.

[91] LUCK, M., GRIFFITHS, N., D'INVERNO, M. From agent theory to agent construction : a case study. MÜLLER, J.-P., WOOLDRIDGE, M.J., JENNINGS, N.R., éditeurs, *Intelligent Agents III – Agent Theories, Architectures, and Languages – Proceedings of ECAI'96 Workshop (ATAL)*, volume 1193, *LNAI*, pages 49–63. Springer-Verlag, 1997.

[92] LUGER, G.F. STUBBLEFIELD, W.A. *Artificial Intelligence : structures and strategies for complex problem solving.* Addison-Wesley, troisième édition, 868 pages, 1998.

[93] MAC ADAM, C.C. An optimal preview control for linear systems. *Journal of Dynamic Systems, Measurement, and Control*, volume 102, pages 188–190, septembre 1980.

[94] MAES, P. The dynamics of action selection. MAES, P., éditeur, *IJCAI'89 : Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 991–7, 1989.

[95] MAES, P. Situated agents can have goals. MAES, P., éditeur, *Designing Autonomous Agents*, pages 49–70. MIT Press, 1990.

[96] MAES, P. The Agent Network Architecture (ANA). *SIGART Bulletin*, volume 2, numéro 4, pages 115–120, 1991.

[97] MAES, P. Agents that reduce work and information overload. *Communications of the ACM*, volume 37, numéro 7, pages 31–40, 1994.

[98] MAÎTRE, B. LAASRI, H. Cooperating expert problem-solving in blackboard systems : ATOME case study. DEMAZEAU, Y. MÜLLER, J.-P., éditeurs, *MAAMAW'89 : Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 251–263. North-Holland, 1990.

[99] MARTIN, D.L., CHEYER, A., LEE, G.-L. Development tools for the Open Agent Architecture. *PAAM'96 : Proceedings of the First International Conference on the Practical Applications of Intelligent Agents and Multi-Agents*, pages 387–404. Practical Application Company, 1996.

[100] MARTINENGO, A., CAMPANI, M., TORRE, V. Artificial systems and complex behaviours. *Proceedings of IEEE International Conference on Intelligent Robots and Systems*, volume 1, pages 194–201, 1994.

[101] MARUICHI, T., ICHIKAWA, M., TOKORO, M. Modeling autonomous agents and their groups. DEMAZEAU, Y. MÜLLER, J.-P., éditeurs, *MAAMAW'89 : Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 215–234. North-Holland, 1990.

[102] MATARIC, M.J. Issues and approaches in the design of collective autonomous agents. *Robotics and Autonomous Systems*, volume 16, numéro 2-4, pages 321–31, 1995.

[103] MCCARTHY, J. Ascribing mental qualities to machines. RINGLE, M., éditeur, *Philosophical Perspectives in Artificial Intelligence*, chapitre 8, pages 161–195. Humanities Press, 1979.

[104] MCCULLOCH, W.S. PITTS, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, volume 5, pages 115–137, 1943.

[105] McRuer, D.T., Allen, R. Wade, Weir, D.H., Klein, R.H. New results in driver steering control models. *Human Factors*, volume 19, numéro 4, pages 381–397, août 1977.

[106] Minsky, M.L. Papert, S. *Perceptrons : An Introduction to Computational Geometry.* MIT Press, 275 pages, 1987.

[107] Moulin, B. Chaib-draa, B. An overview of distributed artificial intelligence. O'Hare, G.M.P. Jennings, N.R., éditeurs, *Foundations of Distributed Artificial Intelligence*, chapitre 1, pages 3–55. Wiley and Sons, 1996.

[108] Müller, H.J. Towards agent systems engineering. *Data and Knowledge Engineering*, volume 23, numéro 3, pages 217–45, 1997.

[109] Müller, J.-P. *The Design of Intelligent Agents : A Layered Approach*, volume 1177, *Lecture Notes in Artificial Intelligence.* Springer-Verlag, 227 pages, 1996.

[110] Müller, J.-P., Wooldridge, M.J., Jennings, N.R., éditeurs. *Intelligent Agents III – Agent Theories, Architectures, and Languages – Proceedings of ECAI'96 Workshop (ATAL)*, volume 1193, *LNAI.* Springer-Verlag, XV-401 pages, 1997.

[111] Musliner, D.J., Durfee, E., Shin, K. World modelling for the dynamic construction of real-time control plans. *Artificial Intelligence*, volume 74, numéro 1, pages 83–127, mars 1995.

[112] Musliner, D.J., Hendler, J.A., Agrawala, A.K., Durfee, E.H., Strosnider, J.K., Paul, C.J. The challenges of real-time AI. *IEEE Computer*, volume 28, numéro 1, pages 58–66, janvier 1995.

[113] Ndumu, D.T., Nwana, H.S., Lee, L.C., Collins, J.C. Visualizing and debugging distributed multi-agent systems. *Proceedings of the Third International Conference on Autonomous Agents*, pages 326–333, mai 1999.

[114] Negroponte, N., éditeur. *The Architecture Machine; Towards a more Human Environment.* MIT Press, 153 pages, 1970.

[115] Nii, H.P. Blackboard systems : the blackboard model of problem-solving and the evolution of blackboard architectures. *Artificial Intelligence*, volume 7, numéro 3, pages 39–53, 1986.

[116] Nwana, H., Ndumu, D., Lee, L., Collis, J. ZEUS : A tool-kit for building distributed multi-agent systems. *Applied Artificial Intelligence*, volume 13, numéro 1, pages 129–186, 1999.

[117] Nwana, H.S. Jennings, N.R. Co-ordination in multi-agent systems. *Software agents and soft computing - Towards enhancing machine intelligence - Concepts and applications*, pages 42–58. Springer-Verlag, 1997.

[118] Nwana, H.S. Ndumu, D.T. An introduction to agent technology. *BT Technology Journal*, volume 14, numéro 4, pages 55–67, 1996.

[119] Nwana, H.S. Ndumu, D.T. A perspective on software agent research. *The Knowledge Engineering Review*, volume 14, numéro 2, pages 125–142, 1999.

[120] Nwana, H.S. Wooldridge, M. Software agent technologies. *BT Technology Journal*, volume 14, numéro 4, pages 68–78, 1996.

[121] PENG, H. TOMIZUKA, M. Preview control for vehicle lateral guidance in highway automation. *Journal of Dynamic Systems, Measurement, and Control*, volume 115, pages 679–686, décembre 1993.

[122] PETRIE, C. What is an agent. MÜLLER, J.-P., WOOLDRIDGE, M.J., JENNINGS, N.R., éditeurs, *Intelligent Agents III – Agent Theories, Architectures, and Languages – Proceedings of ECAI'96 Workshop (ATAL)*, volume 1193, *LNAI*, pages 41–4. Springer-Verlag, 1997.

[123] QUINTUS. *Quintus Flex 1.21 Reference Manual.* Quintus, 216 pages, juin 1991.

[124] RAO, A.S. GEORGEFF, M.P. Modelling rational agents within a BDI-architecture. *KR'91 : Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 473–484. Morgan Kaufmann, 1991.

[125] RAO, A.S. GEORGEFF, M.P. An abstract architecture for rational agents. *KR'92 : Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, pages 439–449. Morgan Kaufmann, octobre 1992.

[126] ROCHWERGER, B., FENNEMA, C.L., DRAPER, B., HANSON, A.R. Executing reactive behavior for autonomous navigation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 838–841, 1994.

[127] RUMBAUGH, J., JACOBSON, I., BOOCH, G. *The Unified Modeling Language Reference Manual.* Addison-Wesley, 576 pages, 1999.

[128] RUSSELL, S. NORVIG, P. *Artificial Intelligence : A Modern Approach.* Prentice Hall Series in Artificial Intelligence. Prentice Hall, 912 pages, 1995.

[129] SARIDIS, G.N. Toward the realization of intelligent controls. *Proceedings of the IEEE*, volume 67, numéro 8, pages 1115–1133, août 1979.

[130] SARIDIS, G.N. Foundations of the theory of intelligent controls. *Proceedings of the IEEE Workshop on Intelligent Control*, pages 23–28, 1985.

[131] SARIDIS, G.N. Intelligent machines : Distributed vs. hierarchical intelligence. *Proceedings of IFAC/IMAC International Symposium on Distributed Intelligence Systems*, pages 34–39, 1988.

[132] SARKAR, N., YUN, X., KUMAR, V. Control of mechanical systems with rolling constraints : Application to dynamic control of mobile robots. *International Journal of Robotics Research*, volume 13, numéro 1, pages 55–69, 1994.

[133] SELFRIDGE, O.G. Pandemonium : A paradigm for learning. BLAKE, D.V. UTTLEY, A.M., éditeurs, *Proceedings of the Symposium on Mechanization of Thought Processes*, pages 511–529, 1959.

[134] SHAPIRO, E.Y. Concurent prolog : A progress report. *Computer*, volume 19, numéro 8, pages 44–58, août 1986.

[135] SHOHAM, Y. Agent-Oriented Programming. *Artificial Intelligence*, volume 60, pages 51–92, 1993.

[136] SINGH, M.P. *Multiagent Systems : A Theoritical Framework for Intentions, Know-How, and Communications*, volume 799, *Lecture Notes in Artificial Intelligence.* Springer-Verlag, 168 pages, 1994.

[137] SLOMAN, A. POLI, R. SIM-AGENT : a toolkit for exploring agent designs. WOOLDRIDGE, M., MÜLLER, J.-P., TAMBE, M., éditeurs, *Intelligent Agents II – Agent Theories, Architectures, and Languages – Proceedings of IJCAI'95-ATAL Workshop*, volume 1037, *LNAI*, pages 392–407. Springer-Verlag, 1996.

[138] SMITH, D.E. STARKEY, J.M. Overview of vehicle models, dynamics, and control applied to automated vehicles. *Advanced Automotive Technologies*, volume 40, pages 69–87, 1991.

[139] STERLING, L. SHAPIRO, E. *The art of Prolog : Advanced Programming Techniques.* MIT Press, deuxième édition, 560 pages, 1994.

[140] STROUSTRUP, B. *The C++ Programming Language.* Addison-Wesley, troisième édition, 928 pages, 1997.

[141] SUGENO, M. MURAKAMI, K. Fuzzy parking control of a model car. *Proceedings of the 23rd Conference on Decision and Control*, pages 902–903, décembre 1984.

[142] SUGENO, M. NISHIDA, M. Fuzzy control of a model car. *Fuzzy Sets and Systems*, volume 16, numéro 2, pages 103–113, juillet 1985.

[143] TÉTREAULT, M. EHSANI, S.M. Analyse cinématique d'un robot mobile de type automobile et stabilisation de la dynamique interne. *Third International Conference on Industrial Automation*, pages 9.11–14, juin 1999.

[144] THOMAS, S.R. The PLACA agent programming language. WOOLDRIDGE, M.J. JENNINGS, N.R., éditeurs, *Intelligent Agents – Proceedings of ECAI-94 Workshop on Agent Theories, Architectures, and Languages*, volume 890, *LNAI*, pages 355–370. Springer-Verlag, 1995.

[145] TOKORO, M. Agents : towards a society in which humans and computers cohabitate. PERRAM, J.W. MÜLLER, J.-P., éditeurs, *MAAMAW'94 : Distributed Software Agents and Applications : Proceedings of the 6th European Workshop on Modelling Autonomous Agents in Multi-Agent World*, pages 1–10. Springer-Verlag, 1996.

[146] TYRELL, T. The use of hierarchies for action selection. *From Animals to Animats 2 : Proceedings of the Second International Conference in Simulation of Adaptive Behavior*, pages 138–147, 1992.

[147] VAN CANEGHEM, M. *L'anatomie de Prolog.* InterEditions, 192 pages, 1986.

[148] VAN LIEDEKERKE, M.H. AVOURIS, N.M. Debugging multi-agent systems. *Information and Software Technology*, volume 37, numéro 2, pages 103–12, 1995.

[149] VERE, S. BICKMORE, T. A basic agent. *Computational intelligence*, volume 6, pages 41–60, 1990.

[150] VON MARTIAL, F., éditeur. *Coordinating Plans of Autonomous Agents*, volume 610, *Lecture Notes in Artificial Intelligence.* Springer-Verlag, 246 pages, 1991.

[151] WERNER, E. What ants cannot do. PERRAM, J.W. MÜLLER, J.-P., éditeurs, *MAAMAW'94 : Distributed Software Agents and Applications : Proceedings of the 6th European Workshop on Modelling Autonomous Agents in Multi-Agent World*, pages 19–39. Springer-Verlag, 1996.

[152] WINSTON, P.H. *Artificial Intelligence.* Addison-Wesley, troisième édition, 750 pages, 1992.

[153] WITTIG, T., éditeur. *ARCHON : An Architecture for Multi-agent Systems.* Ellis Horwood, 135 pages, 1992.

[154] WOOLDRIDGE, M. Agent-based software engineering. *IEE Proceedings - Software Engineering*, volume 144, numéro 1, pages 26–37, 1997.

[155] WOOLDRIDGE, M. Intelligent agents. WEISS, G., éditeur, *Multiagent Systems : a Modern Approach to Distributed Artificial Intelligence*, chapitre 1, pages 27–77. The MIT Press, 1999.

[156] WOOLDRIDGE, M. JENNINGS, N.R. Intelligent agents : theory and practice. *Knowledge Engineering Review*, volume 10, numéro 2, pages 115–52, 1995.

[157] YEN, J. LANGARI, R. *Fuzzy logic : intelligence, control and information.* Prentice Hall, 548 pages, 1999.

[158] ZADEH, L.A. Fuzzy sets. *Information and Control*, volume 8, pages 338–353, 1965.

[159] ZADEH, L.A. Commonsense knowledge representation based on fuzzy logic. *Computer*, volume 16, pages 61–65, 1983.

[160] ZAPATA, R., LÉPINAY, P., NOVALES, C., DEPLANQUES, P. Reactive behaviors of fast mobile robots in unstructured environments : Sensor-based control and neural networks. *Proceedings of the Second Intenational Conference on Simulation of Adaptive Behavior*, pages 108–115. The MIT Press, 1992.