UNIVERSITÉ DU QUÉBEC À MONTRÉAL

DÉVELOPPEMENT D'UNE ARCHITECTURE D'AGENT CONSCIENT POUR UN SYSTÈME TUTORIEL INTELLIGENT

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE (ING. CONNAISSANCE)

PAR

PATRICK HOHMEYER

DÉCEMBRE 2006

REMERCIEMENTS

En préambule à ce mémoire, je souhaite adresser ici tous mes remerciements aux personnes qui m'ont apporté leur aide et qui ont ainsi contribué à l'élaboration de ce mémoire.

Tout d'abord le professeur Roger Nkambou, directeur de ce mémoire et directeur du laboratoire GDAC, pour l'aide et le temps qu'il a bien voulu me consacrer et sans qui ce mémoire n'aurait jamais vu le jour.

Je voudrais aussi remercier toute l'équipe du GDAC pour leur soutien et leur travaux préalables qui ont créé le contexte qui rend ce mémoire possible. Je remercie particulièrement Daniel Dubois pour nos discussions profondes au sujet de l'architecture.

J'adresse également mes remerciements au Pr. Froduald Kabanza de l'université de Sherbrooke, pour avoir contribué à une partie du financement de ce travail de recherche.

J'exprime ma gratitude envers le Pr. Stan Franklin de l'université de Memphis, pour avoir accepté de coopérer avec notre laboratoire et de nous avoir donné accès à son implémentation de l'architecture IDA.

Enfin, je tiens à remercier madame Suzanne Lapointe de m'avoir mis en contact avec Roger Nkambou.

Le travail de recherche présenté dans ce mémoire a été financé en grande partie par le Conseil de Recherches en Sciences Naturelles et en Génie du Canada (CRSNG).

TABLES DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

RÉSUMÉ

Depuis au moins une trentaine d'années, des ordinateurs ont été utilisés dans le domaine de l'enseignement. Les premiers systèmes ont été raffinés par l'intégration des techniques de l'intelligence artificielle donnant ainsi lieu aux systèmes tutoriels intelligents (STI). Les STI sont des agents autonomes et intelligents qui doivent considérer une quantité importante d'information afin de mieux suivre le raisonnement d'un apprenant et l'aider dans son processus d'apprentissage.

Chez les humains, la conscience joue un rôle de premier plan dans le traitement de l'information. En effet, elle permet entre autre de filtrer l'accès aux informations fournies par l'environnement. Récemment, des chercheurs dans le domaine de la psychologie et de l'informatique ont fondé un nouvel axe de recherche lié à la conscience artificielle ; le but est de tenter de reproduire les mécanismes de la conscience dans des agents logiciels afin d'augmenter leur capacité à raisonner.

Ce mémoire traite de l'architecture d'un agent tutoriel intelligent « conscient ». Cette architecture est une extension du système IDA, développé par l'équipe du Pr. Stan Franklin de l'université de Memphis. Le système IDA offre un ensemble d'outils et de modèles permettant l'intégration de la conscience dans un agent logiciel. Il confère à un agent des capacités à filtrer les évènements de l'environnement pour centrer le raisonnement sur les informations les plus importantes. Cette capacité de filtrer l'information est réalisée grâce à la théorie de la conscience humaine de Baars.

L'architecture qui résulte de cette adaptation de IDA est basée sur l'interaction d'agents plus simples (appelés *micro-processus*) qui collaborent sous la direction d'un réseau des actes (inspiré des travaux de Maes). Elle a été intégrée avec succès dans un système tutoriel intelligent pour l'entraînement des astronautes (CanadarmTutor).

En plus de comporter plusieurs avantages par rapport aux architectures existantes, l'architecture proposée est générique et peut être réutilisée pour d'autres projets.

CHAPITRE I

INTRODUCTION

Depuis au moins une trentaine d'années, des ordinateurs ont été utilisés dans le domaine de l'enseignement. Les premiers systèmes offraient déjà du contenu non-linéaire, c'est-à-dire que l'ordinateur se basait sur la dernière réponse de l'étudiant pour choisir la prochaine leçon ou le prochain exercice. Cette pratique a été raffinée dans les systèmes tutoriels intelligents (STI), qui sont alors capables de maintenir un profil cognitif individuel de chaque étudiant afin d'adapter la session d'enseignement à ses besoins et préférences. De plus, l'intégration des techniques de l'intelligence artificielle tant dans l'explication des connaissances du domaine d'apprentissage que dans les connaissances stratégiques liées à la conduite du dialogue a permis d'obtenir des STI encore plus « intelligents ».

Alors que les chercheurs du domaine de l'AIED (« Artificial Intelligence in Education ») tentent de simuler le comportement d'un *tuteur* humain, certaines recherches en psychologie cognitive essaient de simuler le fonctionnement de la *conscience* humaine (conscience artificielle). Ces recherches multidisciplinaires sur la conscience artificielle ont donné lieu à des architectures dites « conscientes » ; d'une part, il s'agit, pour les psychologues, de se doter de plateformes pouvant permettre de valider des théories psychologiques, et d'autre part, pour les informaticiens, de trouver de nouvelles architectures cognitives pouvant servir de fondement aux agents intelligents.

Le projet STC (Système Tutoriel Conscient), dans lequel s'intègre ce travail, veut unir ces deux domaines, en implémentant un système tutoriel intelligent à l'aide d'une architecture fidèle à une théorie de la conscience.

1.1 Contexte de l'étude

Cette première section présente le contexte global du projet STC (Systèmes Tutoriels Conscients) dont le but est de créer un STI complet basé sur les mécanismes de la conscience. La portée du projet spécifique à ce mémoire est plus restreinte et sera établie dans les sections subséquentes.

Le 23 avril 2001, pendant la mission STS-100, un bras robotique a été installé sur la station spatiale internationale. Ce bras robotique, appelé Canadarm2, provient de l'agence spatiale canadienne et est un élément crucial pour l'assemblage de la station. Il sert non seulement à transporter des grosses charges (comme les nouveaux modules), mais aussi comme une plateforme de travail mobile lors des sorties extravéhiculaires des astronautes.

La manipulation de ce bras robotique est une tâche difficile qui nécessite un entraînement important des astronautes. Les sept degrés de libertés du bras sont une première difficulté, car ils augmentent considérablement le nombre de manœuvres possibles. La deuxième difficulté est la limitation de la vue. En effet, il est impossible d'avoir une vue globale de la station, car l'astronaute ne peut voir le bras qu'à travers trois moniteurs dont chacun montre la vue d'une de la quinzaine de caméras montées sur la station et sur le bras. Il doit donc sélectionner parmi ces caméras celles qui lui offrent la vue idéale pour la manipulation en cours. La figure 1.1 montre le module de manipulation sur la station spatiale avec ses trois moniteurs.

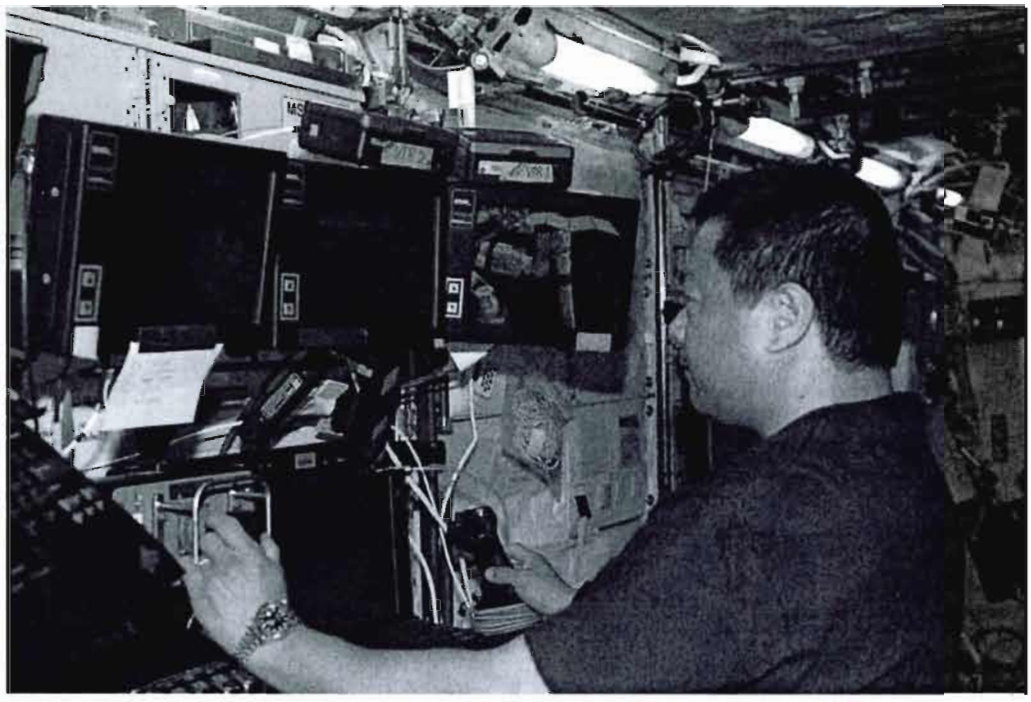

Figure 1.1 Leroy Chiao en train de manipuler le bras canadien (Photo provenant de la NASA)

Lors de la manipulation du bras, l'astronaute doit éviter que le bras entre en collision avec la station spatiale. Il est aussi possible d'amener le bras dans une configuration bloquante. Cette situation est appelée une *singularité* et constitue un autre danger que l'astronaute doit considérer lors de la manipulation. Pour éviter tout risque de collision ou de singularité, il est crucial que l'astronaute ait une bonne vue en tout temps. Ainsi, les choix de caméras doivent être réajustés tout au long de la manipulation afin de permettre, à chaque étape, les sélections offrant toujours les meilleures vues.

Le défi est donc tel qu'un entraînement préalable est nécessaire. Un tel entraînement prépare les astronautes entre autre à une bonne connaissance de l'environnement (« *spatial awareness* »), condition nécessaire pour une meilleure sélection de caméras et de vues.

Pour l'instant, cet entraînement est assuré par des entraîneurs humains de l'agence spatiale canadienne (ASC) en utilisant un simulateur à grande échelle du bras robotique Canadarm2.

En collaboration avec l'ASC et le CRSNG, le laboratoire GDAC a développé un système tutoriel intelligent offrant une interface du simulateur du bras robotique Canadarm2 qui permettra aux astronautes de s'entraîner sans supervision humaine.

Pour y arriver, les étudiants du GDAC ont développé un tuteur non-cognitif qui est doté d'un simulateur de la station spatiale et d'un planificateur de chemins pour la manipulation du bras (Kabanza et al, 2005 ; Nkambou, Belghith et Kabanza, 2006). Le simulateur permet à l'utilisateur d'expérimenter avec le bras canadien dans un micro-monde et de réaliser les exercices fournis par le tuteur. Le rôle du planificateur est de trouver, à partir de n'importe quelle situation, un chemin qui amène le bras canadien à la destination de l'exercice en cours. Ainsi, il assiste l'étudiant en trouvant et en illustrant les possibilités de déplacement du bras. Il peut générer au besoin des démonstrations de tâches pour illustrer un trajet idéal à la demande de l'apprenant.

Comme le GDAC ne dispose pas du budget nécessaire pour répliquer la vraie station de manipulation (vue sur la figure 1.1), le simulateur a été réalisé sur un ordinateur personnel standard. La figure 1.2 montre l'interface du simulateur qui réplique les trois écrans disponibles aux astronautes. Pour chaque écran, l'étudiant peut choisir la caméra dont la vue y sera affichée. De plus, chaque écran permet de manipuler les angles de vue de la caméra. Évidemment, l'interface permet aussi de manipuler le bras en sélectionnant le joint du bras que l'étudiant veut faire bouger, puis en faisant varier l'angle de rotation du joint à l'aide du clavier. Une telle manipulation change l'état du micro-monde et ses effets sont reflétés sur les trois écrans.

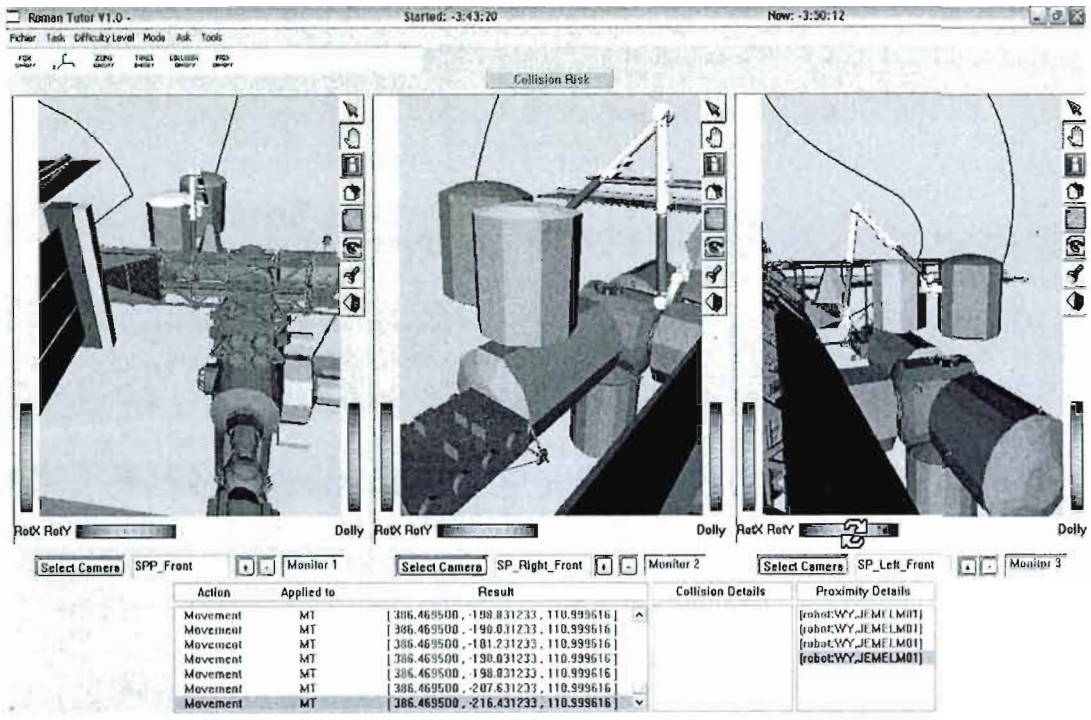

Figure 1.2 Capture d'écran du simulateur (fournie par Khaled Belghith)

En-dessous des caméras, des informations supplémentaires sont affichées, permettant à l'étudiant de mieux comprendre ce qu'il vient de faire et dans quelle situation il se trouve. Par exemple, dans la scène qu'on voit dans la figure 1.2, il y a un risque de collision (un rapprochement entre un joint du bras et un module de la station), et dans la liste en bas à droite, le nom du joint (WY) et du module (JEMELM01) qui se sont rapprochés sont affichés.

Le tuteur fournit donc une rétroaction pédagogique lors de la manipulation du Canadarm2. Toutefois, il n'est pas un système tutoriel intelligent cognitif au sens de Aleven et al (2006). En effet, les rétroactions du tuteur non cognitif sont essentiellement générées à partir des informations reçues du planificateur de chemins qui est capable de valider les manipulations effectuées. Il peut ainsi indiquer le fait que l'apprenant, suite à ses manipulations actuelles, peut encore atteindre le but (en invoquant le planificateur pour vérifier s'il existe un chemin menant au but à partir de l'état actuel). Il peut aussi lui indiquer clairement la procédure à suivre en faisant encore une fois appel aux services du planificateur. Cependant, même s'il

est capable d'observer les déviations de l'apprenant (par rapport au but de la manipulation), il ne peut pas expliquer les raisons de cette déviation en termes d'un manque de (ou d'une mauvaise) connaissance d'une procédure ou de l'environnement. Ainsi, même si ce tuteur, grâce au planificateur permet une rétroaction pédagogique avancée, il est loin de comprendre le modèle cognitif de l'apprenant (ce qui se passe dans la tête de l'étudiant, les motivations profondes de sa démarche, etc.). Cette capacité à comprendre la démarche de l'apprenant constitue pourtant le fondement premier d'un tuteur cognitif.

Le projet STC (Système Tutoriel Conscient), dont fait partie mon projet de maîtrise, tente d'étendre le simulateur avec un vrai système tutoriel intelligent et cognitif qui s'adapte à chaque étudiant individuellement et qui est équipé de mécanismes permettant de tracer le raisonnement de l'apprenant et de l'aider efficacement.

Plus particulièrement, *mon* projet de maîtrise est un sous-projet de ce projet plus large. Il consiste à mettre en place une architecture d'agent conscient qui supportera le système tutoriel conscient. Il ne s'agit donc pas de développer un système tutoriel conscient pour un certain domaine d'apprentissage, mais plutôt du choix et de l'implémentation d'une architecture d'agent conscient capable de supporter l'implémentation d'un tel système.

1.2 Problématique

Nous nous intéressons à utiliser une architecture d'agent pour un système tutoriel intelligent, parce qu'un tel système **est** un agent intelligent. Il présente les cinq caractéristiques attendues d'un agent intelligent telles que définies par Wooldridge (1999).

En effet, un STI est un système :

- *autonome* : il détermine lui-même ses actions ;
- ayant un environnement est *complexe* et *imprévisible* (constitué entre autre du modèle cognitif de l'apprenant (le modèle du monde que l'apprenant crée dans sa tête) qui évolue au fil de l'apprentissage) ;
- *réactif* : il réagit aux actions de l'étudiant et à tout changement dans son modèle cognitif (et même affectif) ou dans l'environnement d'apprentissage ;

- *proactif* : il amène l'étudiant vers un but d'enseignement et peut changer de stratégie lorsqu'il le juge nécessaire.
- qui possède des *habiletés sociales*, il interagit avec l'étudiant, une entité extérieure qui ne partage pas nécessairement les mêmes buts que lui.

De plus, notre système tutoriel doit considérer une multitude d'informations : l'état du micro-monde, les dangers potentiels dans ce monde (par exemple une collision imminente), l'évaluation des actions de l'étudiant, les préférences de l'étudiant, les buts pédagogiques, les rappels provenant de la mémoire, etc. Chez un être humain, la conscience (d'accès) permet de gérer cette complexité et c'est pourquoi nous avons envisagé une architecture d'agent « conscient » pour notre système tutoriel. Nous prétendons aussi qu'un tuteur basé sur une telle architecture d'agent « conscient » montrera un comportement plus proche d'un comportement humain, ce qui facilitera potentiellement l'acceptation par l'étudiant et augmentera sa motivation à apprendre.

Ainsi, nous émettons l'hypothèse qu'une architecture d'agent « conscient » est une bonne base pour un système tutoriel plus intelligent. Si cette conjecture s'avère fondée, notre architecture pourra servir de base au développement d'autres systèmes tutoriels intelligents. Dans le domaine de l'AIED, il y a un besoin d'un fondement efficace pour les tuteurs intelligents orientés agents. Les modèles actuels (architecture BDI, système de production, etc.) souffrent de plusieurs limites lorsqu'il s'agit de prendre en compte plusieurs sources d'informations. Nous reviendrons sur les limites un peu plus loin dans ce mémoire.

Il est aussi bien de noter la complexité inhérente au développement des STI qui constituent des applications complexes. En adaptant une architecture existante pour de tels systèmes, nous validons en même temps l'intérêt et le potentiel de l'architecture en question.

1.3 Objectif

L'objectif du projet STC est double : premièrement, nous visons à évaluer la faisabilité et l'utilité d'un système tutoriel intelligent basé sur une architecture d'agent « conscient ». En deuxième lieu, nous visons à valider l'idée que l'architecture d'agent « conscient » choisie est assez puissante pour soutenir un système aussi complexe qu'un système tutoriel intelligent.

Le travail présenté dans ce mémoire vise précisément à :

1. implémenter l'architecture du tuteur conscient et
2. l'instancier et valider cette architecture à travers un système tutoriel intelligent complexe (CanadarmTutor).

Faisant partie d'un projet plus large, ce projet de mémoire a le mandat précis de fournir une architecture pour le développement du système tutoriel conscient.

Pour illustrer les capacités de l'architecture, ce projet inclut l'implémentation d'un exemple simple de diagnostic du raisonnement de l'apprenant qui permet de traverser tout le cycle cognitif de l'architecture proposée. Le but de cette implémentation est uniquement de tester l'architecture et de vérifier qu'elle fonctionne. À moyen terme, le projet global ira plus loin et offrira des fonctionnalités pédagogiques complètes.

En effet, le projet de ce mémoire ne permet pas à lui seul de valider les conjectures du projet global, à cause de l'absence des fonctionnalités pédagogiques. Un système tutoriel intelligent doit choisir le *bon* acte parmi un grand nombre d'actes possibles. Même si nous avons implémenté un enchaînement d'actes (le diagnostique), ceci n'est qu'une seule séquence d'actes, pour démontrer que l'architecture choisit un acte. Pour évaluer l'utilité de l'architecture d'agent « conscient », il faut évaluer la qualité de ce choix, et ceci nécessite qu'il y ait plusieurs choix possibles. Ces autres choix seront donc à implémenter dans la poursuite du projet global ; ils ne font pas partie du projet décrit dans ce mémoire.

1.4 Méthodologie

Afin d'atteindre ces objectifs, nous avons d'abord choisi une architecture existante qui simule la cognition humaine et qui intègre des mécanismes de la conscience. Après avoir évalué plusieurs de ces types d'architectures (IDA, ACT-R (Anderson, 2004) et CARAMEL 2 (Sabah, 1997)), nous avons décidé de nous baser sur l'architecture IDA, développée par Stan Franklin, à l'université de Memphis. Ce choix se justifie par le fait que IDA est une architecture d'agent universelle, qu'elle est complète, et qu'elle a déjà été réalisée et déployée. L'architecture CARAMEL 2 sera très brièvement présentée à la section 2.3 et une étude comparative entre IDA et ACT-R sera traitée au chapitre IV.

Suite à une entente entre le GDAC et l'université de Memphis, nous avons eu accès au code de IDA. Malheureusement, après une évaluation de ce code, nous avons choisi de ré-

implémenter l'architecture plutôt que de l'adapter à nos besoins. Nous avons donc implémenté une nouvelle architecture à partir des publications de Franklin et en nous inspirant du code fourni.

Lors de cette adaptation de l'architecture aux exigences d'un système tutoriel, nous avons respecté le plus possible la théorie de Baars (la théorie de la conscience sur laquelle Franklin a basé son architecture) pour rester fidèles à l'objectif d'avoir une architecture d'agent *conscient*.

Aussi, pour favoriser la réutilisation de cette architecture, nous avons veillé à produire une architecture la plus générique possible.

1.5 Organisation de ce mémoire

Dans un premier temps, nous présenterons un survol de l'état de l'art sur les agents, les architectures d'agent conscient et les systèmes tutoriels intelligents.

Les deux chapitres suivants forment le cœur de ce mémoire : le chapitre 3 expose une vue globale de l'architecture, permettant au lecteur de comprendre son fonctionnement. Le chapitre 4 positionne cette architecture par rapport à des architectures similaires. Il la compare avec celle de Franklin (IDA), le modèle du tableau noir et ACT-R. Ce dernier est l'architecture la plus utilisée dans les systèmes tutoriels dits cognitifs.

Les trois derniers chapitres exposent les détails plus techniques de notre implémentation. Ils s'adressent principalement à un programmeur qui voudrait réutiliser et étendre notre architecture d'agent conscient pour son propre agent. Ces chapitres permettent aussi de mieux comprendre comment l'architecture est implémentée. Nous terminons ce mémoire par une courte conclusion qui résume nos contributions.

CHAPITRE II

ÉTAT DE L'ART

2.1 Introduction

Ce projet s'inspire d'une multitude de travaux reliés. Dans ce chapitre, nous allons en présenter les plus importants.

Pour permettre au lecteur de mieux comprendre les mérites de l'architecture présentée dans ce mémoire, la première section de ce chapitre aborde le domaine de recherche des agents, des agents intelligents et de leurs architectures.

Le chapitre se poursuit avec une section sur la théorie de la conscience de Baars et une implémentation de celle-ci dans l'architecture IDA. C'est sur cette architecture que se base la nôtre, qui sera présentée dans les chapitres subséquents. Ce document emprunte aussi une partie importante du vocabulaire utilisé par l'équipe de Memphis.

Une section sur les systèmes tutoriels intelligents achèvera ce chapitre.

2.2 Agents et architectures d'agent

Pour mieux situer les architectures présentées, nous introduisons les concepts d'agent et de système multi-agents dans cette section.

Comme l'a si bien affirmé Wooldridge (1999), les chercheurs ne s'entendent pas sur une définition universelle du concept d'agent. Le seul consensus existant affirme que la notion *d'autonomie* est essentielle pour définir un agent. Wooldridge lui-même propose la définition suivante : « An *agent* is a computer system that is *situated* in some *environment*, and that is capable of *autonomous action* in this environment in order to meet its design objectives. » Le concept clé est celui d'une *action autonome*, qui implique qu'un agent doit être capable de

déterminer lui-même ses actions et peut refuser d'exécuter une demande ou une requête. Ceci distingue un agent d'un objet[1], car un objet va toujours exécuter une méthode appelée, tandis qu'un agent peut s'abstenir de ceci.

Une autre caractéristique d'un agent est sa capacité de s'adapter à un évènement inattendu. Un programme traditionnel va se terminer et attendre qu'il soit relancé par une entité externe (un humain ou un programme de surveillance) ou au moins annuler l'action en cours pour revenir à un état stable. Un agent est autonome (ne doit donc pas dépendre d'une entité externe) et n'a pas un contrôle suffisant de son environnement pour revenir en arrière. Par exemple l'environnement d'un traitement de texte est essentiellement le document en cours d'édition. Lorsqu'un problème survient pendant son utilisation, l'application se termine et, lorsque l'utilisateur la relance, elle rétablit le dernier état stable. L'environnement d'un tuteur est constitué essentiellement de l'apprenant. Lorsque l'apprenant est frustré, le tuteur ne peut pas le remettre dans l'état précédent. Le tuteur est obligé de réagir à la situation, même si elle n'a pas été prévue.

Ce sont justement ces environnements complexes, comportant des évènements inattendus, qui créent le besoin d'avoir des agents capables d'actions autonomes. Selon Wooldridge, un environnement complexe possède plusieurs caractéristiques qui le rendent imprévisible. Quoiqu'il soit possible de formuler des prédictions, elles ne sont que des hypothèses, car il est :

- impossible de connaître complètement l'état actuel de l'environnement.
- même si c'était le cas, les prédictions ne seront pas parfaites
- le passé a un effet sur le présent (le passé est garant de l'avenir) et il faut donc le considérer pour connaître l'environnement et
- l'environnement change au fil du temps (indépendamment de l'agent), les croyances de l'agent par rapport à celui-ci peuvent devenir invalides.

Ainsi, un agent doit être préparé à la possibilité que ses prédictions ne s'avèrent pas et que ses actions échouent.

[1] Pour cette phrase, « objet » est utilisé dans le sens d'un objet de la programmation orienté-objet.

Au delà de ces considérations générales sur les agents, les caractéristiques essentielles recherchées pour un agent *intelligent* sont les suivantes :

- *réactivité* : il doit réagir rapidement aux changements de son environnement et ajuster ses plans en conséquence ;
- *proactivité* : il doit planifier ses actes et *prendre des initiatives* pour atteindre ses buts (ce qui implique qu'il ait des buts) ;
- *habilité sociale* : il doit interagir avec d'autres agents ou des humains pour atteindre ses buts (que les autres agents ne partagent pas nécessairement).

Cette catégorie d'agents nous intéresse particulièrement car le comportement attendu d'un système tutoriel intelligent répond à ces caractéristiques. Dans ce qui suit, nous présentons trois architectures d'agent, dont deux pour des agents intelligents.

2.2.1 Agents purement réactifs

Un agent purement réactif est un agent dont l'action dépend uniquement de ce qu'il perçoit à un instant donné. Un tel agent ne conserve aucune information interne, il ne considère pas l'historique pour décider de son action.

L'architecture d'agent réactif la plus connue est la « subsumption architecture » de Rodney Brooks (Brooks, 1986). Observant que l'approche préexistante (de *déduire* la prochaine action en utilisant la logique formelle) est impossible pour tout environnement réel, Brooks proposa une architecture ne contenant aucune représentation symbolique. Cette architecture est basée sur l'idée qu'un comportement rationnel est un produit de *l'interaction* avec l'environnement et *émerge* de l'interaction de comportements plus simples.

C'est pourquoi l'agent contient un ensemble de règles simples, chacune réagissant à des situations de l'environnement. Ces règles ont une forme très simple : *situation* → *action.*

Chaque agent possède un ensemble de ces règles simples. De plus, les règles sont classées dans une hiérarchie de subsomption (d'où le nom de l'architecture), où les règles sont organisées en couches. Une règle d'une couche plus basse a une priorité plus forte et peut inhiber une règle d'une couche supérieure. Pour un robot par exemple, la couche la plus basse (avec la plus forte priorité) contient les règles qui garantissent la survie du robot, qui

lui permettent d'éviter les collisions. Les couches supérieures contiennent les règles qui dirigent le robot vers son but. Ainsi, le robot se dirige vers son but, sauf si cela le met en risque de collision, auquel cas la couche inférieure inhibe l'avancement et évite l'obstacle. Une fois l'obstacle évité, le robot reprend le chemin vers son but.

Cette approche possède l'avantage de réagir rapidement. Aucun raisonnement complexe ne doit être entrepris, puisqu'il suffit de vérifier les préconditions des règles.

Cette architecture comporte plusieurs limites dont Wooldridge en soulève cinq. Le premier point interdit à lui seul son utilisation pour un système tutoriel intelligent : comme les agents réactifs ne modélisent pas leur environnement, ils doivent disposer d'informations suffisantes dans leur environnement local, leur permettant de déterminer l'action acceptable. (Wooldridge, 1999). Or un système tutoriel intelligent base ses décisions sur l'historique de l'interaction avec l'étudiant – ce qui est une information non-locale.

2.2.2 Agents BDI

L'architecture BDI (*Belief-Desire-Intention*) prend ses racines dans une théorie philosophique du raisonnement humain. Selon cette théorie, un humain se base sur ses croyances et désirs pour formuler les intentions qui mènent à ses actes.

L'état interne d'un agent qui utilise cette architecture consiste en ses croyances, ses désirs et ses intentions. Les *croyances* forment le modèle que l'agent se fait de son environnement ; c'est l'ensemble des faits qu'il présume vrais. Les *désirs* sont l'ensemble des options (désirables) qui s'offrent à l'agent. Les *intentions* sont l'ensemble des options que l'agent a décidé de poursuivre. Ainsi, la seule différence entre une option et une intention est que l'agent s'est engagé à poursuivre la dernière, tandis qu'il n'a que considéré la première.

L'architecture BDI utilise un cycle cognitif qui est présenté dans la figure 2.1. Au début, l'entrée des capteurs est analysée par une *fonction de révision des croyances* (*belief revision function* – brf) qui met à jour les croyances de l'agent à partir de nouvelles perceptions de l'environnement. Par exemple, supposons que les manipulations effectuées sur le bras robotique amènent ce dernier dangereusement proche de la station. Suite à cette captation, la croyance qu'il y a un danger de collision est ajoutée à l'ensemble des croyances.

Ensuite, la fonction « *generate options* » *génère des options* à partir des croyances et des intentions de l'agent. Cette fonction accomplit deux buts : elle planifie, et elle découvre des opportunités. La planification permet de déterminer comment l'agent peut atteindre ses intentions dans l'environnement actuel – c'est-à-dire en respectant les contraintes données par ses croyances. La découverte lui permet de trouver de nouvelles opportunités qui s'offrent suite aux nouvelles croyances. Dans l'exemple précédent, la fonction pourrait générer trois nouvelles options : « laisser l'étudiant continuer », « avertir l'étudiant du danger » et « interrompre l'étudiant et remédier à la cause de la manipulation dangereuse »

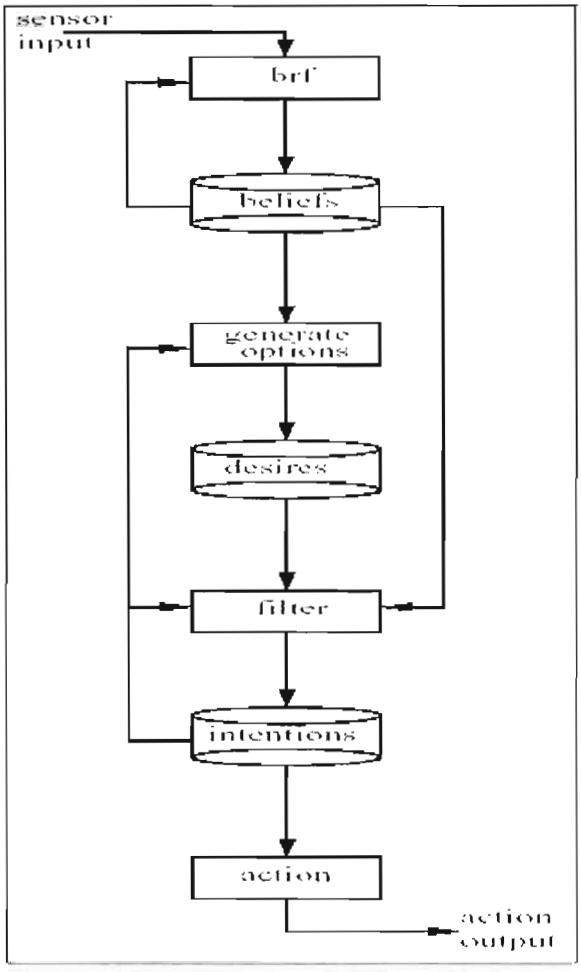

Figure 2.1 Cycle cognitif de l'architecture BDI (Bratman et al., 1988)

Les options ainsi générées (qui constituent en fait les désirs de l'agent) sont ensuite *filtrées* par une troisième fonction (*filter*). C'est cette fonction qui choisit parmi les désirs de l'agent ceux auxquels il s'engage. Pour faire ce choix, la fonction se base sur les intentions actuelles de l'agent ainsi que sur ses croyances. Dans notre exemple, supposons que l'agent croit que l'étudiant est un novice et qu'il croit qu'un comportement dangereux chez un novice doit être corrigé immédiatement. Dans ce cas, il choisira la dernière des trois nouvelles options. Ce choix exclut la première et rend inutile la deuxième (lorsque l'exercice est interrompu, il n'est plus utile de donner des avertissements).

La fonction de filtrage réexamine aussi les intentions existantes. Elle enlève celles qui ont été réalisées. Elle peut aussi désengager l'agent d'une intention qui n'est plus réalisable ou profitable, étant données les croyances actuelles. Décider du moment pour se désengager est une question difficile : un agent qui se désengage trop rapidement n'accomplira aucune de ses intentions ; un agent qui ne se désengage pas mettra son énergie à poursuivre une intention qui est vouée à l'échec. En général, plus l'environnement est volatil, plus l'agent devrait réexaminer ses intentions souvent.

Finalement, une fonction « *action* » choisit une des intentions exécutables pour procéder à son exécution.

Le modèle BDI est un modèle accrédité dans le domaine des agents. Il est intuitif, car il reprend les notions du raisonnement humain. De plus, un grand effort a été mis à produire un modèle formel de cette architecture. L'architecture que nous proposons demeure mieux adaptée à un système tutoriel intelligent. Nous justifierons cette affirmation dans le chapitre 4, après avoir présenté cette architecture.

2.2.3 INTERAP

Sachant qu'un agent purement réactif réagit rapidement à son environnement et qu'un agent proactif est capable de planifier pour parvenir à ses buts, l'idée d'une architecture hybride en couches (où il y a une couche réactive et une couche proactive) se présente naturellement. INTERAP offre une telle possibilité.

L'architecture d'INTERAP est appelé une *architecture en couches verticales en deux passes*, parce que le contrôle commence dans la couche la plus basse (la couche réactive) qui reçoit

la perception, puis monte dans les couches supérieures, puis descend et repasse une deuxième fois dans la couche réactive pour qu'elle choisisse une action.

L'architecture INTERAP possède trois couches, illustrées par la figure 2.2. La couche la plus basse est réactive, celle du milieu proactive, et la couche supérieure gère l'aspect social. Chaque couche possède un répertoire qui lui est propre : la couche réactive conserve un modèle du monde, la couche proactive préserve les plans et les actions de l'agent lui-même et la couche supérieure maintient le savoir sur les plans et actions des *autres* agents.

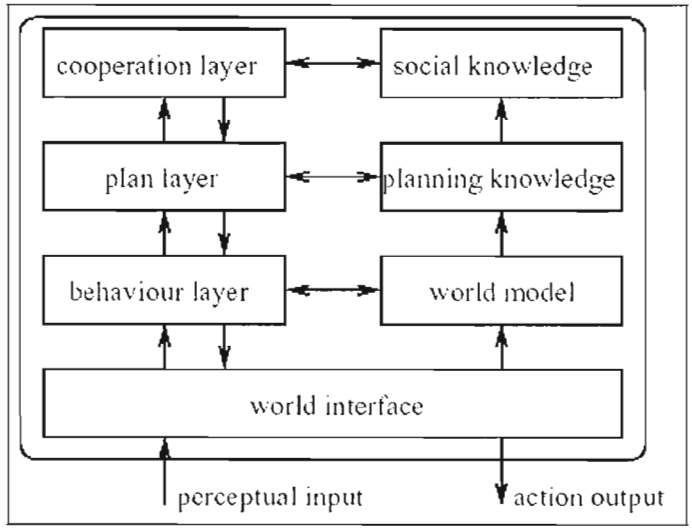

Figure 2.2 L'architecture de INTERAP (Wooldridge, 1999)

Chaque couche essaie de traiter la situation elle-même. Ainsi, quand une action purement réactive suffit dans la situation, la couche réactive exécute immédiatement une action et l'agent bénéficie de la rapidité d'un agent réactif.

Quand une couche n'est pas assez compétente pour traiter la situation, elle *active* la couche supérieure qui va essayer de la traiter à son tour pour ensuite exécuter sa décision à travers la couche inférieure. L'exécution descendante utilise les capacités de la couche inférieure. Ainsi, la couche sociale ne va pas agir sur l'environnement, mais créer des plans dans la couche de planification. Cette dernière va donner des ordres à la couche réactive qui elle va agir sur l'environnement.

Une architecture en couches est une approche très pragmatique au problème de la création d'agents à la fois réactif et proactif. Malheureusement, l'architecture en couche en soi n'offre pas de mécanisme qui permettrait de filtrer la multitude d'informations qui se présentent à un système tutoriel intelligent basé sur un micro-monde. Naturellement, il est possible d'ajouter un tel mécanisme à cette architecture. Cependant, ce qui nous a conduit vers le domaine de la conscience artificielle, c'est que les architectures d'agent « conscient » présentent un mécanisme de filtrage d'information qui est non seulement intégré, mais au cœur de ces architectures. En effet, c'est la nature même d'une conscience d'accès de filtrer les informations importantes d'une affluence d'informations parfois moins utiles.

2.3 Conscience artificielle

Le domaine de la conscience artificielle est un domaine jeune, dans lequel plusieurs chercheurs expérimentent des approches parfois très différentes.

Avant d'opter pour l'architecture IDA, nous avons aussi évalué CARAMEL 2 de Gérard Sabah (Sabah, 1997). En effet, CARAMEL 2 est une architecture dont l'élément central est la compréhension de la langue naturelle, et comme nous n'en avons présentement pas besoin, nous avons préféré travailler avec une architecture plus générale.

CARAMEL 2 apporte plusieurs idées, dont une « conscience », les « carnets d'esquisse » et des méta-agents de contrôle. La « conscience » est un processus contrôlé qui établit le lien entre le traitement automatisé (« inconscient ») de l'information et la réflexion de plus haut niveau. Un « carnet d'esquisse » est un tableau noir modifié qui permet un retour d'information d'un niveau supérieure vers les processus plus simples. Par exemple, lorsque le niveau le plus bas interprète une série de sons comme étant les lettres « a », « r », « p », « r » et « e », la couche supérieure va détecter que ceci n'est pas un mot valide et demander à la couche inférieur de lui faire une nouvelle proposition. Finalement CARAMEL est un système multi-agents et multi-experts qui présente des méta-agents qui contrôlent un ou plusieurs autres agents, offrant ainsi de l'introspection.

Notre architecture par contre est basée sur l'architecture d'agent conscient IDA de Stan Franklin de l'université de Memphis. Avant de la présenter, nous allons esquisser la théorie psychologique sur laquelle cette architecture se fonde.

2.3.1 Le modèle de Baars

Le modèle de Baars n'est pas une imitation de la conscience, mais une « théorie scientifique rigoureuse de la conscience » (traduit de Baars, 1997). Nous présenterons ici les éléments essentiels de cette théorie afin de permettre au lecteur de mieux comprendre l'architecture de Franklin qui en est une implémentation.

Pour décrire sa théorie, Baars utilise la métaphore d'un théâtre. L'élément central de cette métaphore est « *the stage of working memory* », c'est-à-dire la *scène* qui représente la mémoire de travail ; cette scène à une taille très limitée. La scène est en grande partie dans l'obscurité, et l'humain n'est pas conscient du contenu de cette partie obscure. La conscience est comme un *spot* lumineux d'attention qui se promène sur la scène de la mémoire de travail. Comme le dit Baars : « *the theatre has a powerful spotlight of attention, and only events in the bright spot on stage are strictly conscious* ».

Sur la scène se promènent les *acteurs* qui sont en *compétition* pour attirer le spot de l'attention. Ces acteurs varient considérablement dans leur complexité : ils peuvent être aussi simples qu'un seul neurone sensoriel ou être aussi complexes qu'un événement multimodal composé par des millions de neurones.

Au-delà de la compétition sur scène pour l'attention, Baars inclut dans sa métaphore le travail *dans les coulisses*, qui établit le contexte influençant les activités sur la scène. Baars évoque l'exemple de la perception de la profondeur : un humain suppose que la lumière vient d'en haut et se sert de cette supposition pour interpréter ce qui est capté par les yeux, à un point tel que nous ne reconnaissons plus un visage quand il est illuminé d'en bas. Ce contexte permet de comprendre des informations autrement ambiguës ; avec le risque de mal les interpréter quand le contexte choisi n'est pas approprié pour ces informations.

Le contexte peut aussi inclure des suppositions conceptuelles et guider nos pensées conscientes, même si la supposition elle-même ne devient jamais consciente. Un membre important de ce contexte inconscient est le *réalisateur*, l'ensemble des fonctions exécutives. C'est lui qui connaît les buts actuels du système et qui guide la mémoire de travail (et l'attention) en fonction de ces buts.

Finalement, il y a l'*audience*, l'ensemble des processus inconscients qui observent la scène et qui réagissent à ce qui s'y passe. La majorité du travail cognitif est faite par l'audience, de

façon inconsciente et décentralisée, et ce n'est que le résultat de ce travail qui monte sur la scène pour devenir, possiblement, conscient.

Les travaux de Baars contiennent plus que cette simple métaphore ; entre autres, ils évoquent des résultats scientifiques qui soutiennent sa théorie et ils dressent les implications de cette théorie. Nous n'allons pas élaborer là-dessus, car il suffit de comprendre la métaphore pour comprendre l'implémentation de Franklin, ainsi que la nôtre.

2.3.2 L'architecture IDA de Franklin

Inspirée par la théorie de Baars, l'équipe de Stan Franklin de l'université de Memphis a développé trois agents intelligents qui implémentent à un degré croissant cette théorie. Ce travail a débuté en 1996 avec l'agent *Virtual Mattie*, qui gère une liste de courriels d'annonce de conférences. Cet agent fut remplacé par *Conscious Mattie*, dont l'implémentation est plus fidèle à la théorie de Baars. Finalement, l'équipe de Franklin a développé l'agent IDA (*Intelligent Distribution Agent*) pour la marine américaine, qu'elle continue de perfectionner.

Comme nous nous sommes basés sur cette dernière architecture pour développer notre agent, elle sera présentée dans ce paragraphe. Le but de cette présentation est de donner au lecteur une idée de ce qui a été accompli par l'équipe de Franklin pour qu'il puisse mieux évaluer notre apport.[2]

IDA est utilisé pour l'assignation de postes aux marins de la marine américaine et joue le rôle complexe mais bien décrit du *répartiteur*. À l'achèvement de leur affection, les marins se font assigner un nouveau poste. Pour cette attribution, IDA doit prendre en compte plusieurs contraintes, incluant aussi bien les exigences de la marine que les préférences du marin concerné. IDA doit correspondre avec chaque marin par courriel et interroger des bases de données. Normalement, un marin contacte le *répartiteur* lui-même avant la fin de son affectation, mais s'il ne le fait pas, IDA doit prendre l'initiative de le contacter. (Franklin, 2005)

---

[2] Cette description est assez brève et ne permettra peut-être pas au lecteur de comprendre le fonctionnement exact de cette architecture. Nous référons le lecteur aux publications de Franklin et aux chapitres trois et quatre de ce mémoire qui comparent notre architecture avec celle de Franklin et présentent plus de détails.

L'architecture de Franklin décrit un système multi-agent. Franklin appelle le système complet un « conscious agent » et les agents simples qui le composent des « *codelets* », ce que nous avons traduit par *micro-processus*[3]. Le point central du système est la *conscience d'accès*[4] (la *publication*) qui amène l'agent à réagir uniquement sur les informations les plus pertinentes. L'architecture utilise aussi deux réseaux : un réseau sémantique pour la perception et un réseau des actes (qui relie des actes avec leurs préalables et leurs conséquences) pour le choix de l'action.

IDA fonctionne en suivant un *cycle cognitif*, répété perpétuellement et qui détermine ainsi le comportement de l'agent. Ce cycle contient neuf étapes, illustrées dans la figure 2.3. Les deux premières étapes constituent la perception, dans laquelle les entrées du monde extérieur sont transformées en un format interne et ajoutées à la mémoire de travail. Suite à cela, les mémoires à long terme réagissent au contenu de la mémoire de travail et y ajoutent les associations rappelées. Puis, des micro-processus d'attention forment des coalitions à partir du contenu de la mémoire de travail et la coalition la plus forte est publiée. Suite à cette publication, le réseau des actes active des parties qui sont pertinentes pour la situation et choisit un acte à poser, qui est ensuite exécuté.

[3] À ne pas confondre avec la notion de processus légers (« *threads* »).

[4] Comme le terme « conscience » désigne un agrégat de concepts flous, nous spécifions qu'il s'agit d'une *conscience d'accès*. Ce terme désigne la « conscience » dans son rôle de régler l'accès à l'information : une information « consciente » est plus accessible qu'une information inconsciente.

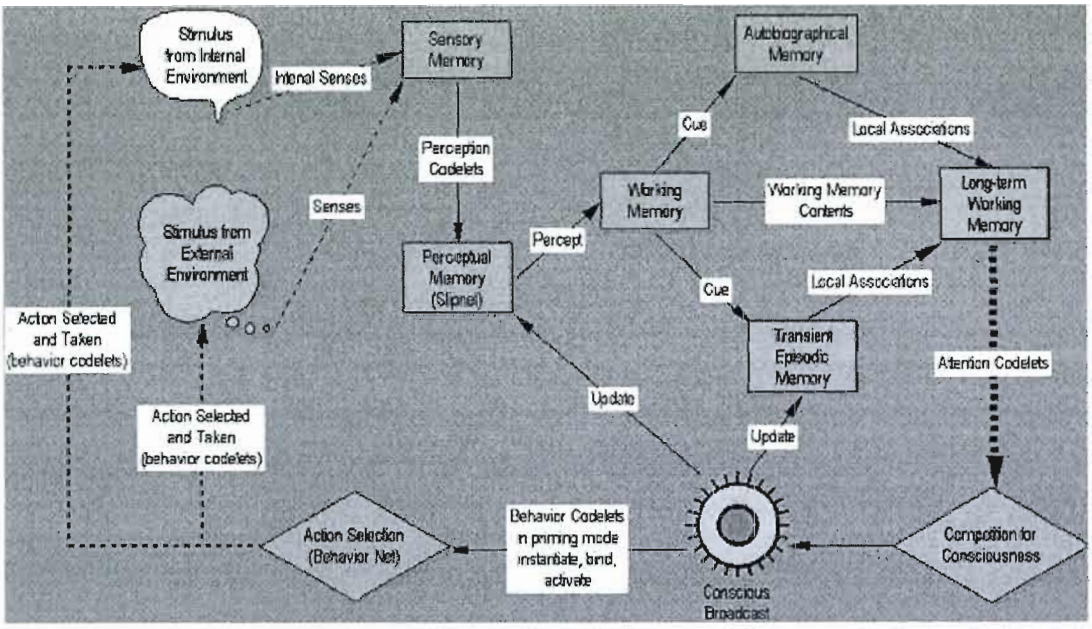

Figure 2.3 Cycle cognitif d'IDA (Franklin, 2005)

Le cycle cognitif d'IDA[5]

Comme IDA perçoit son monde par des chaînes de caractères (qui peuvent être soit un courriel, soit le résultat d'une requête à une base de données), le but de la perception est d'interpréter ces chaînes de caractères.

Pour cela, Franklin utilise un réseau sémantique dont les nœuds sont activés selon leur pertinence par rapport au message reçu. Il nomme cette structure un « *slipnet* », selon la structure utilisée dans l'architecture Copycat (Hofstadter 1984), même si les deux structures ne sont pas complètement identiques.

Copycat prend en entrée deux chaînes de trois caractères qui sont formées selon des règles analogues et où un caractère est manquant. Par exemple « abc » et « ut? », où le « ? » représente le caractère manquant. Le « slipnet » s'appelle ainsi parce que l'état du réseau peut glisser (en anglais : « slip ») d'une interprétation vers une interprétation analogue (par exemple trouver que « abc » est une séquence de lettres croissante, ce qui est analogue à « uts », une séquence de lettres décroissante).

[5] Encore une fois, ceci n'est qu'un survol afin d'éviter d'anticiper sur la description de l'architecture STC. Les concepts introduits lors de ce survol seront expliqués au prochain chapitre. (Sauf le « slipnet » qui est propre à l'architecture de Franklin).

Dans le réseau de Franklin, il n'y a pas cette recherche d'analogie, les nœuds étant surtout organisés selon leur sémantique : par exemple, « nor » est une abréviation de « Norfolk » un « emplacement ». Ainsi, quand le courriel contient le mot « nor », le nœud « nor » est stimulé, ce qui va stimuler le nœud « Norfolk », qui à son tour stimulera le nœud « emplacement ». Puis le nœud « emplacement » va stimuler les nœuds « préférence », « acceptation » et « requête d'information », ce qui augmente la chance que ces concepts soient détectés dans le courriel. En effet, la présence d'un emplacement dans le courriel indique qu'il s'agit d'une de ces trois propositions. Ainsi, le réseau trouve une représentation symbolique et sémantique de la chaîne de caractères.

Les règles du réseau font que les niveaux d'activation des nœuds se stabilisent lorsqu'une représentation est trouvée. Ceci (la stimulation des nœuds qui représentent un mot et la stimulation à l'intérieur du réseau jusqu'à-ce que le réseau se stabilise), constitue la première étape du cycle cognitif d'IDA : « *perception* ». La deuxième étape, « p*ercept to preconscious buffer* », est simplement l'acte de prendre le percept – décrit par les nœuds du slipnet qui ont une activation au-dessus d'un seuil – et de l'inscrire dans un tampon préconscient.

Franklin utilise aussi le terme « mémoire de travail » pour désigner ce tampon préconscient. Il n'est pas complètement identique à la scène, à cause de ce qui se passe à l'étape trois du cycle cognitif : « *Local associations* ». Dans cette étape, le contenu du tampon préconscient est utilisé pour stimuler les mémoires à long terme : la mémoire épisodique (« transient episodic memory », qui contient des informations sur l'épisode en cours, par exemple l'endroit où on a garé sa voiture aujourd'hui) et la mémoire déclarative (« long-term declarative memory » qui contient des concepts et associations généralisés).

Les mémoires à long terme sont implémentées à l'aide d'une mémoire creuse distribuée (« sparse distributed memory », (Kanerva 1988)). Cette structure utilise un espace binaire multi-dimensionnel et encode toute information dans un vecteur de bits. Elle permet de retrouver, à partir d'un vecteur d'indice, le vecteur de bit le plus semblable qui a été écrit dans la mémoire. Ainsi, n'importe quelle partie du vecteur peut être utilisée pour retrouver le vecteur complet. Le fonctionnement de cette structure reproduit le fonctionnement de la mémoire humaine en plusieurs points. Pour en savoir davantage, nous référons le lecteur au livre de Kanerva de 1988 et à l'article de Anwar et Franklin de 2003. Le contenu du tampon préconscient, complété avec les associations retournées par ces deux mémoires, donne la

mémoire de travail à long terme (« long-term working memory »), correspondant à la scène de Baars.

Suite à cela, il y a la compétition pour la conscience (« c*ompetition for consciousness* ») pendant laquelle les micro-processus d'attention forment des coalitions avec les micro-processus d'information. Aussi bien les micro-processus d'information que les micro-processus d'attention ont une *activation* qui représente leur pertinence pour la situation actuelle. Les micro-processus d'attention choisissent leur activation en fonction de la correspondance entre la situation donnée et leurs préférences internes. Malheureusement, Franklin ne donne pas de règles pour déterminer l'activation des micro-processus d'information.

Ce sont les micro-processus d'information qui portent les informations qui sont contenues dans la mémoire de travail à long terme et les micro-processus d'attention qui les regroupent dans des coalitions. Les articles de Franklin ne se prononcent pas clairement sur la question à savoir si les micro-processus d'information se créent automatiquement à partir du contenu de la mémoire de travail à long-terme, s'ils **constituent** cette mémoire(c'est-à-dire que la mémoire de travail à long terme ne sera pas un tampon, mais une collection de micro-processus) ou encore s'ils sont créés par les micro-processus d'attention au besoin.

Les micro-processus d'attention par contre, existent en dehors de la mémoire de travail à long-terme et examinent son contenu à chaque cycle cognitif pendant cette étape pour former les coalitions. Puis, le « *spotlight controller* » choisit la coalition avec la plus grande activation moyenne de ses membres.

À la prochaine étape (« c*onscious broadcast* »), le contenu de la coalition gagnante est publié à tout le système. Il est aussi enregistré dans la mémoire épisodique. Le système réagit à cette publication dans la prochaine étape, (« r*ecruitement of resources* »). Lors de cette étape, les micro-processus d'action (« *behavior codelets* »), qui sont concernés par la publication, s'activent.

Pour comprendre les dernières étapes, nous allons brièvement introduire la structure du réseau des actes (« *behavior net* »). Ce réseau a été développé à partir du réseau de Petty Maes (Maes 1989). C'est un réseau qui est composé d'*actes* (« *behaviors* ») liés entre-eux par

des états, décrivant les préalables et les résultats prévus pour chaque acte. Chaque acte possède une activation qui est une forme de planification.

De plus, le réseau contient des *désirs* (« *drives* »), qui sont les buts ultimes de l'agent, et des *buts* (« goals ») qui sont des buts intermédiaires qui permettent de hiérarchiser les actes. Finalement, les *enchaînements* (« *behavior streams* ») sont des regroupements de buts et d'actes qui donnent une structure au réseau. Un enchaînement est instancié quand il est pertinent à la situation et le mécanisme de sélection d'action décrit par Petty Maes ne s'exécute que sur les enchaînements instanciés.

Au début de la vie de l'agent, aucun enchaînement n'est instancié et c'est lors de la septième étape (« *setting goal context hierarchy* ») que les micro-processus d'action, qui se sont activés à l'étape précédente, instancient les enchaînements d'actes qui sont appropriés à la situation. Concrètement, comme chaque micro-processus d'action est associé à un acte, un tel micro-processus activé instancie l'enchaînement auquel appartient son acte. Cette instanciation permet aussi de fixer les variables qui sont utilisées par les actes de l'enchaînement. Finalement, les micro-processus d'action envoient de l'activation aux actes. Cette activation est une des sources d'activation qui circule dans le réseau.

Lors de la prochaine étape (« *action chosen* »), les autres sources d'activation (l'environnement à travers des états, les motivations à travers des désirs et la planification précédente à travers l'activation présente dans les actes) distribuent l'activation dans le réseau et ajustent l'activation des actes. Puis l'acte exécutable (dont toutes les préconditions sont satisfaites) avec la plus forte activation est choisi pour être exécuté.

L'exécution elle-même est la dernière étape du cycle cognitif (« *action taken* »). Elle consiste en l'exécution des micro-processus d'action qui composent l'acte choisi, et cette exécution peut avoir des conséquences aussi bien internes qu'externes. En général, cette action influence le cycle suivant, par exemple, une requête vers une base de données externe dont la réponse va alimenter la perception au prochain cycle.

Finalement, le cycle cognitif reprend à la perception et continue perpétuellement.

Un peu plus loin dans ce mémoire, nous présenterons une adaptation de ce processus dans une architecture des systèmes tutoriels intelligents. En attendant, nous rappellerons, dans la prochaine section, certains concepts fondamentaux des STI.

2.4 Systèmes tutoriels intelligents

Puisque notre architecture soutiendra un système tutoriel intelligent, il semble pertinent de présenter les concepts fondamentaux de ce domaine. Pour cela, nous nous basons sur l'article de Beck, Stern et Haugsjaa de 1996.

Contrairement aux systèmes classiques d'apprentissage par ordinateur, les STI ne se basent pas simplement sur la dernière réponse de l'apprenant, mais conservent un modèle de l'apprenant qui se construit à partir des toutes les interactions avec ce dernier. Cela permet au système de prendre des décisions pédagogiques sur la suite de la session d'apprentissage. La communauté AIED s'accorde sur le fait qu'un STI est constitué au moins de trois composantes de base : le module de l'apprenant, le module du domaine et le tuteur ou module pédagogique (figure 2.4 tirée de l'article de Beck).

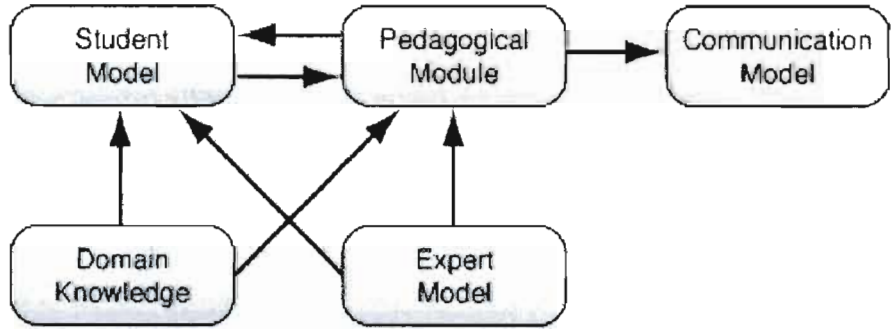

Figure 2.4 Interaction des composantes d'un STI (Beck, 1996)

2.4.1 Modules d'un STI

Le modèle de l'apprenant conserve les croyances que le système a sur l'apprenant, et ce sont ces croyances qui guideront les décisions du module pédagogique. Ce dernier encode des connaissances stratégiques permettant au système de guider le processus d'apprentissage. Ces connaissances sont généralement inspirées des théories d'apprentissage ou d'instruction. Des travaux récents cherchent à formaliser ces théories grâce à l'ingénierie ontologique, ce qui représenterait une source importante de connaissances pour ce dernier module.

Le module du domaine comporte généralement deux sous composantes : les connaissances du domaine d'apprentissage constituées essentiellement de concepts, de faits ou de

procédures ; et un système expert capable de résoudre les problèmes du domaine (modèle expert). Ce dernier exploite les connaissances du domaine ou encore des mécanismes de résolution de problème qui lui permettent de trouver des solutions aux problèmes du domaine.

Tel qu'indiqué en introduction, on s'attend à ce qu'un système tutoriel intelligent soit en mesure de suivre le raisonnement de l'apprenant, de diagnostiquer les erreurs de l'apprenant, d'expliquer le domaine d'apprentissage et de fournir l'aide adéquate dont l'apprenant a besoin.

2.4.2 Agents pédagogiques

Une des tendances actuelles en AIED est de déployer des STI sous forme d'agents pédagogiques intégrés dans des environnements virtuels.

Plusieurs formes d'agents pédagogiques ont été proposés : des agents aidants, des agents compagnons ou des agents perturbateurs. STEEVE (Johnson, 2001) est un exemple d'agent aidant intégré dans un environnement virtuel, où il peut démontrer des tâches à l'étudiant, comme la manipulation des machines (voir figure 2.5).

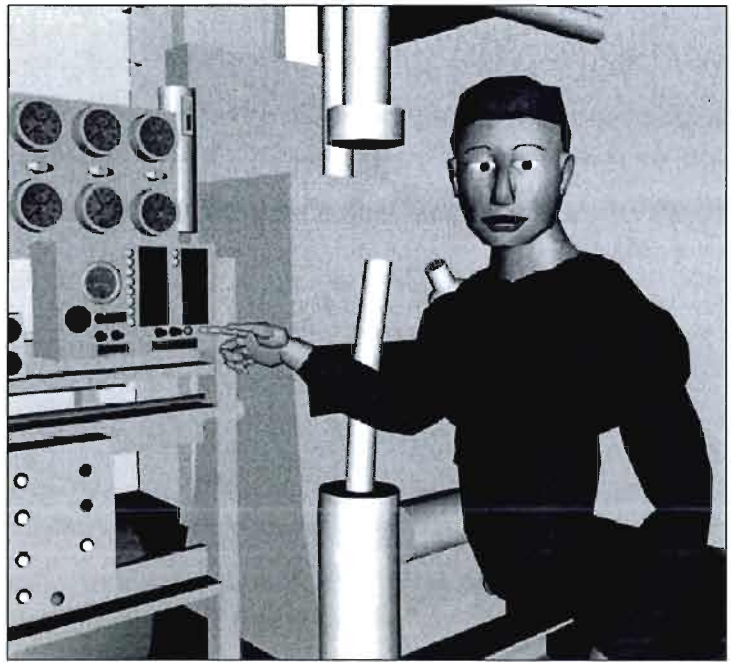

Figure 2.5 STEEVE dans un environnement virtuel (Johnson, 2001)

Dans le cas d'agents compagnon, un agent peut jouer le rôle d'un *co-apprenant* (ou *apprenant artificiel*), simulant un autre étudiant qui apprend en même temps que l'apprenant humain. L'interface du système tutoriel doit laisser bien distinguer entre le tuteur de ce co-apprenant. Car contrairement au tuteur qui possède (grâce au module expert) une connaissance complète du domaine de l'apprentissage, ce co-apprenant a une connaissance limitée. Il peut travailler avec l'apprenant humain pour combiner leur compréhension du domaine.

Lorsque ce co-apprenant fournit des mauvaises informations à l'apprenant humain (entre autre pour valider la confiance de l'humain dans ses connaissances), on dit qu'il joue le rôle d'un agent perturbateur (Frasson et al., 1997).

2.5 Conclusion

De nombreuses publications dans le domaine des systèmes tutoriels intelligents (STI) traitent les tâches qu'un STI doit accomplir et les rôles qu'il peut jouer dans l'interaction avec l'apprenant. Les architectures proposées sont modulaires et divisent le système selon les fonctionnalités. Certains travaux introduisent aussi des architectures multi-agents où les trois composantes de base sont des agents plutôt que des modules.

Nous avons aussi montré qu'une architecture d'agent conscient est une architecture propice pour filtrer la quantité importante d'informations reçue par un STI. Néanmoins, aucun travail dans le domaine des STI se base présentement sur une architecture d'agent conscient.

Le prochain chapitre présente donc une nouvelle architecture d'agent conscient : une adaptation de l'architecture IDA pour un système tutoriel « conscient ».

CHAPITRE III

UNE NOUVELLE ARCHITECTURE D'AGENT CONSCIENT

3.1 Introduction

Dans les chapitres précédents, nous avons présenté au lecteur le contexte de notre travail. D'abord le cadre théorique des travaux de recherche desquels nous nous sommes inspirés, puis l'environnement dans lequel notre implémentation exploratoire est intégrée.

Avant de donner les détails d'implémentation de notre solution dans les chapitres 5, 6 et 7, ce chapitre décrit l'architecture de notre système pour en donner une vue d'ensemble et permettre au lecteur de comprendre comment les différentes parties interagissent.

L'architecture étant basée sur l'architecture IDA, elle-même fondée sur la théorie de Baars, nous allons utiliser des termes de la théorie de Baars dans ce chapitre en supposant que le lecteur a une idée de cette théorie que nous avons déjà exposée dans la section 2.3.1.

Notre système comporte un cycle cognitif qui est répété perpétuellement. Pour commencer la présentation de cette architecture, nous aborderons un survol de ce cycle, suivi d'une description des *micro-processus*, éléments fondamentaux de cette architecture. Les micro-processus agissent à l'intérieur des trois modules principaux (la perception, l'atelier global et le réseau des actes) que nous allons esquisser ci-après (ils seront détaillés dans les chapitres subséquents). Par la suite, nous détaillerons le comportement dynamique du système à un niveau d'abstraction élevé : notre version du cycle cognitif.

3.2 Survol du cycle cognitif

Tout comme l'architecture de Franklin, l'architecture se base sur un cycle cognitif qui est répété perpétuellement. Quoique ce cycle est inspiré de celui de Franklin, il en diffère sur

plusieurs points. Dans cette section, nous allons faire un survol de *notre* cycle cognitif. Une description plus détaillée de chaque étape suivra dans la section 3.5, pour le comparer dans le chapitre suivant avec celui de Franklin.

1. **Perception.** Lors de cette étape, les informations captées par les capteurs de l'agent sont analysées et converties en une représentation interne : des micro-processus d'information. L'implémentation concrète de cette étape varie fondamentalement avec la nature de l'environnement observé. Par exemple, dans notre système, nous observons les mouvements d'un bras robotique dans un environnement virtuel à travers des messages formels, ce qui est très différent des applications de Franklin, qui observent des courriels écrits en langue naturelle.

2. **Raisonnement par les micro-processus de raisonnement.** Les micro-processus de raisonnement s'exécutent. Chacun ajuste son activation, joint la scène s'il le veut et prend d'autres actions au besoin.

3. **Compétition pour la « conscience ».** Les micro-processus qui se retrouvent sur la scène (soit parce qu'ils viennent de la joindre, soit parce qu'ils y étaient au cycle précédent et ne l'ont pas quittée) forment des coalitions[1], et la coalition la plus forte gagne la compétition.

4. **Publication de la coalition gagnante.** Tout le système reçoit une publication qui contient les micro-processus de la coalition gagnante.

5. **Recrutement de ressources.** Chaque micro-processus peut réagir individuellement à cette publication. Par exemple un micro-processus d'arbitrage écoute la publication pour se tenir au courant des propositions de solution à la délibération qu'il modère. Les états du réseau des actes reçoivent aussi la publication et se mettent à vrai quand ils reconnaissent une information pertinente à eux. Par exemple, l'état « Collision détectée » se met à vrai quand la publication contient un micro-processus de type « Collision ». Ainsi, les états font le lien entre le module cognitif et le réseau des actes.

6. **Choix de l'acte à prendre.** La publication marque la fin de l'étape de raisonnement et c'est maintenant le temps de choisir un acte. Se basant sur l'état de l'environnement (représenté par les états), les désirs actuels de l'agent et la planification précédente, le

[1] En distinction de l'utilisation habituelle, dans le vocabulaire de Franklin, « coalition » signifie un regroupement d'*un* ou plusieurs micro-processus.

réseau des actes choisit un acte pour l'exécution. Il se peut que le réseau n'arrive pas à effectuer un choix. Dans ce cas, le choix est reporté au prochain cycle.

7. **Exécution de l'acte choisi.** Finalement, les micro-processus qui constituent l'acte choisi sont exécutés et modifient l'environnement interne et/ou externe de l'agent.

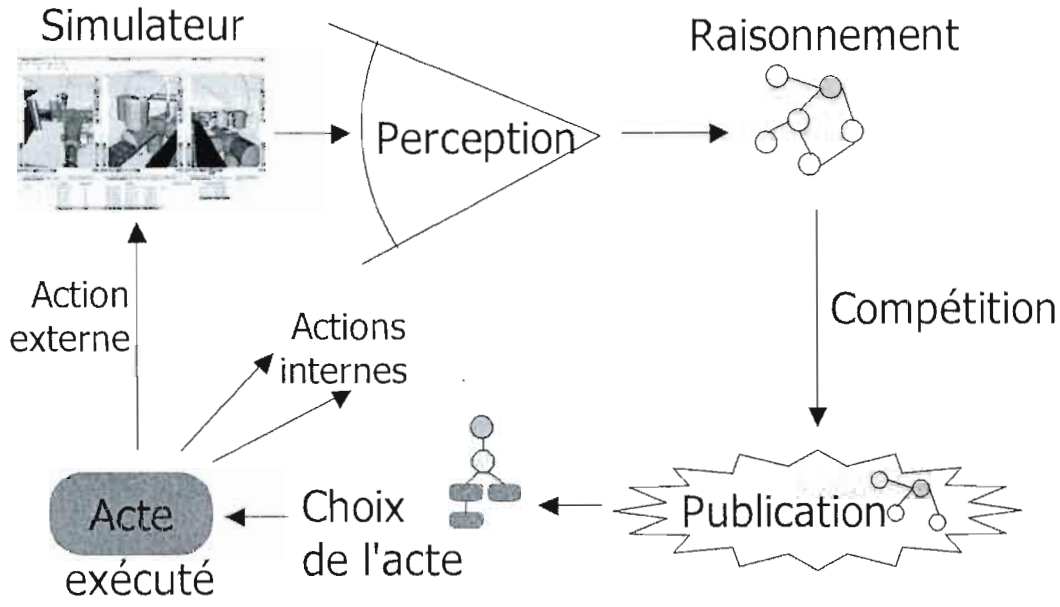

Figure 3.1 Le cycle cognitif de notre architecture

3.3 Micro-processus

Cette architecture et celle de Franklin utilisent des agents très simples et un répertoire central de l'information (la mémoire de travail). Dans ces architectures, chacun de ces agents simples est aussi une information dans le répertoire central, ce qui permet à l'agent d'avoir une introspection complète. Dans ses publications, Franklin réserve le terme *agent* pour décrire le système complet (un agent autonome, intelligent et « conscient »). Il utilise le terme « *codelets* » pour référer aux agents simples. Nous avons conservé ce terme et choisi la traduction *micro-processus* pour les désigner. Ce terme fait référence aux processus simples de l'esprit de la théorie de Baars. Il n'est aucunement lié au concept des processus légers de la programmation. Cette traduction s'accorde aussi avec la définition de Franklin, selon laquelle un micro-processus est un processus actif et spécifique dont la fonction est représentée par quelques lignes de code (Franklin et Patterson, 2006).

Chaque micro-processus a donc deux facettes possibles : il agit et/ou il représente une information.

La facette « *information* » de chaque micro-processus est constituée de plusieurs éléments :

1. une étiquette qui identifie le type de l'information. Par exemple « Collision ».
2. une *valeur* qui précise l'information. Par exemple « possible ». L'étiquette et la valeur forment *l'identificateur* du micro-processus d'information (« Collision=possible »). Celui-ci permet de référer à un micro-processus de façon uniforme et générique.
3. des *liens* – plus ou moins forts – que le micro-processus peut avoir avec d'autres micro-processus. Ils permettent d'indiquer les micro-processus qui ont un rapport avec le premier, créant ainsi un pseudo réseau sémantique. Le micro-processus « Collision=possible » est lié aux deux objets qui sont en risque de collision[2]. Ces liens peuvent être unidirectionnels, de sorte qu'un micro-processus A peut avoir un lien vers B, sans que B n'ait un lien vers A.
4. le *niveau d'activation* du micro-processus. L'activation représente l'importance de l'information dans la situation actuelle. Le niveau d'activation varie entre 0 et 1. C'est le micro-processus lui-même qui détermine son niveau d'activation pour indiquer sa pertinence. Ceci implique que nous faisons confiance aux micro-processus pour choisir un bon niveau d'activation[3]. Ainsi, les règles qui déterminent le niveau d'activation de chaque micro-processus sont une partie importante des paramètres à ajuster pour un bon fonctionnement du système.

La figure 3.2 illustre ces quatre éléments : elle présente le micro-processus « Collision=possible » avec son activation et ses trois liens vers d'autres micro-processus. (La figure montre uniquement l'identificateur de ces autres micro-processus.)

---

[2] Un micro-processus n'est jamais lié à un objet. Quand nous écrivons qu'un micro-processus est lié à un objet, ceci signifie que le micro-processus est lié au micro-processus qui représente l'objet en question. Nous avons choisi cette formulation pour alléger le texte.

[3] Ce qui implique que le modèle ne peut pas être utilisé dans un système ouvert, où les micro-processus peuvent provenir d'une source externe – et comme ceci n'est pas le but du système, nous acceptons cette limitation.

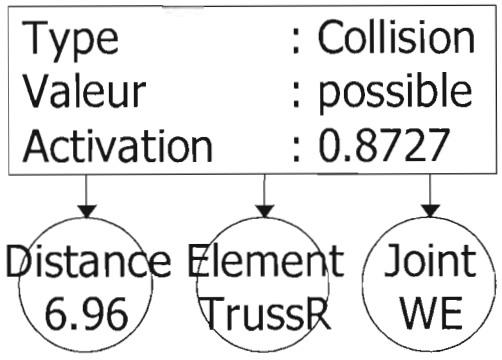

Figure 3.2 Les quatre éléments de la facette « information »

Un micro-processus n'est pas limité à ces quatre informations universelles, il peut aussi contenir des informations supplémentaires qui lui sont spécifiques.

La facette « *agent* » de chaque micro-processus (ce qu'il fait quand il agit) varie selon le type du micro-processus. Nous distinguons six types principaux[4] qui sont énumérés dans le tableau 3.1. De plus, l'architecture se base sur un cycle cognitif qui est répété perpétuellement et nous regroupons les micro-processus selon l'étape du cycle où ils agissent. Présentement, nous distinguons trois groupes de micro-processus : les micro-processus de perception, de raisonnement et d'action[5].

Les ***micro-processus de perception*** agissent juste après avoir reçu un stimulus. Ils prennent l'information qui provient de l'extérieur et la rendent utilisable à l'interne ; concrètement, ils la convertissent en micro-processus d'information. L'action de ces micro-processus consiste donc à créer et à mettre à jour des micro-processus d'information.

[4] Ce sont les six types qui sont présentement utilisés dans notre implémentation. L'architecture ne limite pas les types de micro-processus et selon l'application, il est possible d'en rajouter.

[5] L'élaboration des mémoires à long terme, qui sont prévues dans l'architecture, implique un quatrième groupe, les micro-processus de mémorisation.

Tableau 3.1 Types de micro-processus

Groupe	***Nom***	***Rôle***
Perception	Micro-processus de perception	Interpréter l'environnement de l'agent
Raisonnement	Micro-processus d'information	Représenter l'information
Raisonnement	Micro-processus de confirmation	Vérifier que les résultats attendus se réalisent
Raisonnement	Micro-processus d'arbitrage	Mener la délibération
Action	Micro-processus moteur	Agir sur l'environnement externe de l'agent
Action	Micro-processus de lancement	Lancer des micro-processus de raisonnement

Ensuite agissent les ***micro-processus de raisonnement***. Ils ont deux fonctions différentes : la première consiste à former un groupe d'informations pertinentes (une *coalition*), ce qui est important pour l'aspect « conscience » de notre architecture. La deuxième est le raisonnement : faire un traitement sur l'information présente pour l'enrichir avec le résultat.

Plusieurs types de micro-processus font partie de ce groupe. Les *micro-processus d'information* en sont les plus simples ; ils représentent l'information et forment des coalitions, mais ils ne raisonnent pas eux-mêmes. Par contre, ils sont utilisés dans le raisonnement des autres types de micro-processus de raisonnement.

D'autres membres de ce groupe sont les *micro-processus de confirmation*. Quand l'agent exécute un acte, il peut lancer un tel micro-processus pour vérifier que les résultats prévus de cet acte se réalisent. Et quand un de ces résultats attendus n'est pas atteint, le micro-processus de confirmation rapporte ce problème.

Finalement, les *micro-processus d'arbitrage* mènent la délibération. La délibération permet à plusieurs micro-processus de coopérer à la résolution d'un problème. Le micro-processus d'arbitrage modère cette coopération et annonce la décision prise. La délibération permet d'utiliser la conscience comme un tableau noir : les micro-processus participants apportent des bouts de solution qui sont publiés quand ils sont pertinents, et l'arbitre décide quand la solution est suffisante.

Finalement, les ***micro-processus d'action*** sont ceux qui agissent à la fin du cycle cognitif. Contrairement aux micro-processus des deux autres groupes, leurs possibilités d'action ne sont pas limitées. Ils peuvent aussi bien affecter l'état interne de l'agent, qu'effectuer une action sur l'environnement externe de l'agent. Et contrairement aux micro-processus des deux autres groupes (qui agissent à chaque cycle), seul un petit nombre de micro-processus d'action agissent lors d'un cycle donné. Pendant chaque cycle cognitif, l'agent choisit un seul acte à exécuter et seuls les micro-processus d'action faisant partie de cet acte agissent.

Deux types de micro-processus d'action existent : les *micro-processus d'action moteurs* et les *micro-processus d'action de lancement*.

Les *micro-processus d'action moteurs* affectent l'environnement externe de l'agent. Par exemple un micro-processus qui affiche un message à l'étudiant, ou un qui envoie une demande à l'environnement 3D de remontrer les derniers cinq mouvements du Canadarm que l'étudiant a effectués.

Les *micro-processus d'action de lancement* affectent l'environnement interne de l'agent en lançant des micro-processus de raisonnement qui continuent d'agir dans les cycles subséquents. Ceci permet, par exemple, d'amorcer un raisonnement prolongé (comme la délibération) ou d'attendre un résultat. Les micro-processus de confirmation sont créés ainsi.

Dans notre implémentation, nous n'utilisons pas d'autre micro-processus d'action. D'autres applications (ou des versions ultérieures de la nôtre) pourront en inclure d'autres. Par exemple des micro-processus d'action qui créeront des nouveaux micro-processus de perception.

3.4 Modules principaux

Notre architecture comporte trois modules principaux, chacun correspondant à un groupe de micro-processus, c'est-à-dire à la perception, l'atelier global et le réseau des actes. Avant d'expliquer leur fonctionnement dynamique dans la section suivante, nous allons présenter leur structure. Comme ce chapitre vise à donner une vue d'ensemble au lecteur, nous avons choisi de présenter des descriptions de ces modules à un haut niveau d'abstraction, pour couvrir les détails dans les chapitres suivants.

Le module de la ***perception*** est la partie qui traite l'information provenant des capteurs. Le résultat de la perception est que des micro-processus d'information rejoignent la « scène » de l'atelier global (voir le paragraphe suivant). Le fonctionnement de la perception varie selon la nature des informations captées et du contexte applicatif. Dans notre cas, l'agent reçoit des messages textuels, envoyés par le simulateur, qui décrivent l'état actuel du micro-monde. Ces messages ont une forme fixe spécifiée par une grammaire formelle. Ceci permet d'utiliser un analyseur syntaxique qui transforme les messages en un arbre syntaxique. Les deux structures utilisées par la perception sont donc une file d'entrée pour les messages et l'arbre syntaxique qui est inspecté par les micro-processus de perception.

L'***atelier global***[6] est la partie centrale de l'architecture. Il contient la mémoire centrale de travail et les mécanismes qui implémentent la « conscience ». Du point de vue statique, l'atelier global est composé de trois collections de micro-processus : le théâtre, la scène et les spectateurs ; trois concepts de la métaphore de Baars. Le *théâtre* est « l'endroit » où tous les micro-processus de raisonnement résident. Concrètement, c'est l'ensemble de tous les micro-processus de raisonnement qui existent à un moment donné. La *scène* est l'ensemble des micro-processus présentement pertinents. Vu que les micro-processus sont aussi des informations, la scène est l'équivalent de la mémoire de travail centrale. Finalement, les *spectateurs* sont l'ensemble des micro-processus qui ne sont pas sur la scène. Un micro-processus spectateur n'est présentement pas pertinent, mais il peut le devenir. (Les mécanismes seront détaillés lors de la description du cycle cognitif dans la section suivante.)

C'est le ***réseau des actes*** qui, dans cette architecture, est responsable de la prise d'action. Il est le module le plus complexe des trois, basé sur le travail de Patty Maes (Maes 89) et son adaptation par l'équipe de Franklin (Negatu et Franklin 2002). Il est composé principalement d'états et d'actes.

Les *états* reflètent des états de l'environnement interne et externe de l'agent (par exemple, un état « Collision détectée » reflète le fait qu'une collision s'est produite dans le simulateur).

Les *actes* sont l'ensemble de toutes les actions disponibles à l'agent. Chacun est décrit par ses préconditions, ses effets prévus et les actions effectuées par l'acte :

---

[6] Ceci correspond au « Global Workspace » de la théorie de Baars. Quoique « zone de travail globale » et « espace de travail général » sont des traductions plus fidèles, nous avons choisi le terme « atelier global » parce qu'il est plus facile à mémoriser.

- Les *préconditions* indiquent quels états doivent être vrais pour que l'acte puisse être effectué.
- Les *effets* marquent les états qui sont supposés devenir vrais (ou faux) à la suite de l'exécution de l'acte.
- Les *actions effectuées* par l'acte sont l'ensemble de micro-processus d'action qui sont exécutés pour accomplir l'acte.

Ces deux structures (états et actes) correspondent au modèle de Maes, sauf que Maes n'a pas utilisé des micro-processus pour décrire le moyen d'exécuter un acte.

De plus, le réseau contient des ajouts faits par Franklin, qui sont les *buts*[7] (ce sont essentiellement des actes sans actions) et les *enchaînements* (un regroupement d'actes et de buts). L'objectif de ces deux structures est double. D'une part, ils donnent une structure au réseau pour aider son constructeur. D'autre part, ils permettent de limiter la planification sur la partie du réseau présentement pertinente. (Nous allons détailler ces ajouts et leurs effets dans le chapitre 7 sur le réseau des actes.)

Finalement, les *désirs*[8] du réseau indiquent des états qui sont désirables à atteindre et dirigent ainsi le choix de l'action.

L'architecture prévoit un quatrième module qui sera l'ensemble des mémoires à long terme. Mais ce module n'est pas encore conçu pour des raisons évoquées dans la section 4.2 du chapitre suivant, lors de la comparaison avec l'architecture de Franklin et son choix de mémoire.

3.5 Détails du cycle cognitif

Dans la section 3.2, nous avons donné un survol du cycle cognitif. Maintenant, nous détaillerons chaque étape et illustrerons les rôles que les micro-processus y jouent.

---

[7] Franklin et Maes utilisent chacun le terme « *goal* » différemment dans leurs articles. Nous avons choisi d'utiliser le mot « but » avec la signification de Franklin. Le « *goal* » de Maes est un ensemble d'états que l'agent cherche à réaliser et ceci dirige la planification. Ce qui est équivalent à nos désirs.

[8] Nous avons choisi de préférer le terme « *désir* » sur le terme « *drive* » (pulsion) que Franklin utilise, parce que ce dernier est trop lié à la psychanalyse freudienne.

3.5.1 Perception

Comme le fonctionnement de la perception varie selon la nature des informations captées et le contexte applicatif, l'étape de la perception n'est pas subdivisée dans la description du cycle cognitif, même si notre implémentation exploratoire comporte des sous-étapes.

Dans cette implémentation exploratoire, la perception se divise en trois sous-étapes. D'abord, l'agent reçoit un message du simulateur, puis ce message est analysé syntaxiquement et finalement les micro-processus de perception utilisent le résultat de l'analyse pour créer, modifier et détruire[9] des micro-processus d'information.

Comme le simulateur peut envoyer des messages à tout instant, la première étape se fait de façon asynchrone et n'est donc pas strictement une sous-étape de l'étape de perception. Quand le simulateur envoie un message, ce message est immédiatement mis dans une file d'attente pour qu'il puisse être traité par la perception lors du prochain cycle cognitif.

Lors de l'étape de perception de ce prochain cycle, le (ou les) message(s) est retiré de la file et analysé par un analyseur syntaxique qui crée un arbre syntaxique du message (cf. chapitre VI). Ensuite, les micro-processus de perception inspectent cet arbre pour trouver des informations qui les concernent. Chaque micro-processus s'intéresse à une information précise. Par exemple, il y a un micro-processus pour chaque joint du bras robotique. Quand un micro-processus trouve l'information qui l'intéresse, il trouve le micro-processus d'information qui contient l'information de la dernière perception. Le micro-processus compare ensuite la nouvelle information avec l'ancienne et décide de la pertinence de la nouvelle information – il choisit l'activation qu'il veut attribuer au micro-processus d'information. L'implémentation comporte trois heuristiques pour déterminer le niveau d'activation :

1. Chaque information a une certaine importance (et donc activation) de base, selon son type. La détection d'une collision est plus importante que de savoir que le joint SY existe encore.
2. Une information qui a changé (par exemple un joint du robot qui a bougé) ou qui est nouvelle, est plus intéressante qu'une information qui ne change pas.

[9] Normalement, un micro-processus devient inactif quand il n'est plus pertinent. Néanmoins, la perception peut aussi détruire un micro-processus, si son existence représente un fait qui n'est plus vrai dans le micro-monde observé (le simulateur)

3. Une situation qui s'améliore est moins importante qu'une situation qui s'aggrave. Dans l'implémentation présentée, ceci est pris en compte pour le risque de collision quand un joint se rapproche de la station. Quand le joint continue de se rapprocher, c'est un danger plus grave, quand il s'éloigne, la situation s'améliore.

Après avoir déterminé la pertinence, le micro-processus de perception met à jour l'information du micro-processus d'information ainsi que son activation. Souvent, cette mise à jour va modifier les liens que les micro-processus d'information ont entre eux. Par exemple, quand une caméra envoie son image sur un écran, le micro-processus qui représente cet écran a un lien avec le micro-processus de la caméra. Quand la caméra change, le lien avec l'ancienne caméra est détruit et un nouveau lien est établi. La nouvelle information écrase l'ancienne, contenue dans le micro-processus d'information (au lieu de créer un deuxième micro-processus d'information). Ceci assure que la représentation de la réalité extérieure est à jour et qu'il n'y a pas un mélange d'informations à jour et d'informations obsolètes.

Ainsi, ce qui est perçu arrive dans le théâtre.

3.5.2 Raisonnement par les micro-processus de raisonnement

Il y a plusieurs activités pendant l'étape du raisonnement. Premièrement, l'activation de tous les micro-processus décroît. Comme cela se passe à chaque cycle, l'activation d'un micro-processus décroît progressivement dans le temps. Ceci implémente l'idée qu'une information perd de l'importance avec son âge si elle ne reçoit pas de nouveaux stimuli. De plus, la force des liens entre les micro-processus décroît très lentement pour refléter un vieillissement similaire.

Comme la compétition ne se déroule qu'entre les micro-processus qui se trouvent sur la scène, il est important pour les micro-processus de joindre et quitter la scène au bon moment. Ces deux actions se passent pendant cette étape de raisonnement. Un micro-processus joint la scène quand il décide qu'il est pertinent à la situation. C'est chaque micro-processus qui prend cette décision selon ses propres règles. Par exemple, un micro-processus de confirmation joindra la scène quand son temps d'attente est écoulé. Une fois sur la scène, un micro-processus y reste jusqu'à-ce qu'il soit publié ou que son activation – qui décroît – tombe en dessous d'un seuil. C'est aussi pendant cette étape de raisonnement que les micro-

processus ayant été publiés au cycle précédent quittent la scène. Comme un micro-processus ne peut pas quitter et joindre la scène au même cycle, ceci permet à l'agent de publier une autre coalition, même s'il y a une situation très urgente qui autrement aurait monopolisé les publications.

Finalement, l'étape de raisonnement donne à chaque micro-processus de raisonnement la possibilité d'agir. Ce que le micro-processus fait avec cette possibilité dépend de sa nature : les micro-processus d'information ne prennent pas d'action, les micro-processus de confirmation surveillent la scène pour vérifier si le résultat attendu s'est réalisé et les micro-processus d'arbitrage vérifient si la proposition actuelle mérite d'être choisie et annoncée.

3.5.3 Compétition pour la conscience

Après l'étape de raisonnement, où les micro-processus se préparent pour la compétition, se déroule la compétition elle-même. Chaque micro-processus qui se trouve sur la scène forme une coalition avec tous les micro-processus avec lesquels il a des liens, y inclus ceux qui ne sont pas sur la scène. Ceci implique qu'un micro-processus peut faire partie de deux coalitions différentes. Le micro-processus qui forme la coalition est nommé le *micro-processus principal* de cette coalition.

Pour illustrer cela, voici un exemple lorsque le joint SY du bras canadien s'approche de l'élément « US Lab » de la station. Il y a trois micro-processus sur la scène (« Collision=possible » (C), « Élément=US Lab » (E) et « Joint=SY » (J)) auxquels s'ajoutent deux en dehors de la scène (« Distance= 8,23 » (D) et « Rotation=0.55 » (R)). Tel qu'indiqué sur la figure 3.3, C a un lien vers E, J et D. E n'a pas de liens (les liens peuvent être dirigés) et J a un lien vers R. Ceci donne les coalitions (C, E, J, D), (E) et (J, R). J et E se trouvent donc chacun dans deux coalitions. De plus, la formation des coalitions ne se fait qu'avec les micro-processus qui sont liés directement. C a un lien vers J qui a un lien vers R, mais R n'est pas dans la coalition formée par C. La formation des coalitions ne tient donc pas compte des liens transitifs.

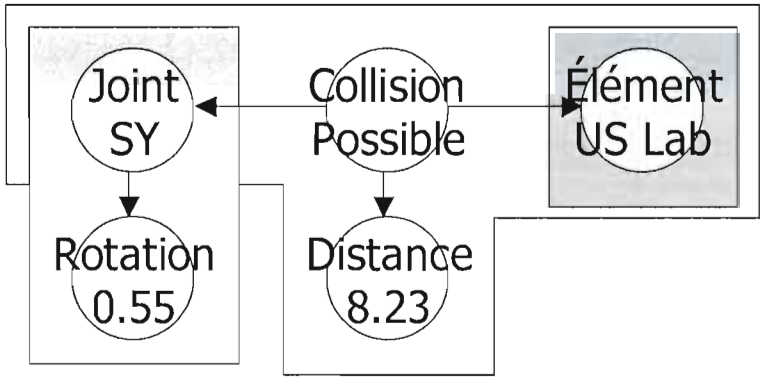

Figure 3.3 Exemple de formation de coalitions

Puis, chaque coalition calcule sa force et la coalition la plus forte gagne la compétition. La force d'une coalition est la somme de l'activation de son micro-processus principal et de la moyenne des autres micro-processus. Pour continuer avec le même exemple, si C a une activation de .8, E de .5, J de .5 et D de .8, la coalition (C, E, J, D) a une activation de .8 + (.5 + .5 + .8)/3 = 1.4. Par contre, la coalition (E) n'a qu'une activation de .5 + 0 = .5. Ceci crée deux biais qui sont intentionnels :

1. L'activation du micro-processus principal est aussi importante que l'ensemble des autres activations.
2. Les coalitions d'un seul micro-processus sont défavorisées.

Nous allons les justifier pendant la comparaison avec le choix de Franklin.

3.5.4 Publication de la coalition gagnante

Une fois la coalition la plus forte déterminée, cette coalition est publiée pour que tout le système « en prenne conscience ». Normalement, une composante du système ignore ce qui se passe sur la scène et ne réagit que sur les informations qui sont publiées. Ainsi la compétition et la publication ensemble filtrent les informations qui arrivent sur la scène et amènent ainsi l'agent à dédier ses ressources aux informations les plus pertinentes. Ceci distingue cette architecture d'agent « conscient » (et celle de Franklin) du modèle du tableau noir : dans ce modèle, chaque composante prend en considération toute information qui arrive sur le tableau noir, et c'est à la composante de déterminer la pertinence de

l'information pour elle. Dans l'architecture présentée dans ce chapitre, la compétition détermine la pertinence d'une information de façon centralisée, ce qui donne une direction globale à l'agent. Ceci évite qu'une composante travaille sur une information qui est globalement insignifiante. De plus, la publication n'implique pas que toutes les composantes travaillent sur la même information. Si, par exemple, la publication dit que l'étudiant s'ennuie, le modèle du domaine ne va pas y réagir, parce qu'il n'est pas concerné par l'état émotionnel de l'étudiant.

3.5.5 Recrutement de ressources

Comme tout le système entend la publication, celle-ci sert à recruter des ressources spécialisées de l'agent. Ainsi, la publication crée le lien entre l'atelier global et le réseau des actes. Dans le réseau des actes, ce sont les états qui écoutent la publication. Quand l'état entend une publication qui contient une information qui le concerne, il se met à vrai. Ce micro-processus (contenu dans la publication) *déclenche* l'état. Par exemple, quand une publication contient le micro-processus d'information « Collision=effective », l'état « Collision détectée » se met à vrai et influence ainsi le choix du prochain acte.

Les états (et les actes et enchaînements d'actes qui y sont reliés) ne sont pas les seules ressources qui sont recrutées par la publication. Elle peut aussi recruter des micro-processus inactifs au moment de la publication, ainsi que des modules additionnels qui ne sont pas nécessairement implémentés par des micro-processus.

3.5.6 Choix de l'acte à prendre

Le choix de l'acte à prendre se fait dans un réseau d'acte, selon les règles de Maes. C'est-à-dire que chaque acte du réseau contient un *niveau d'activation*[10] qui représente une sorte de planification et le choix de l'acte se base sur ce niveau d'activation. L'exécution d'un acte, dont le niveau d'activation est élevé, est « prévue » prochainement. « Prévue » a été mis entre guillemets pour souligner qu'il n'y a pas de représentation explicite des plans et qu'il n'y

[10] L'activation des actes est différente de l'activation des micro-processus. Quoiqu'ils indiquent tous les deux la pertinence de l'objet et servent à choisir l'objet le plus pertinent, ils ne sont pas comparables. Le niveau d'activation des actes est un nombre positif, calculé selon des règles strictes, tandis que celui des micro-processus est un nombre entre 0 et 1, déterminé individuellement.

a donc pas de prévision d'exécution dans le sens d'une liste des actes dont l'exécution est prévue.

Nous allons ici présenter un survol des règles de Patty Maes ; pour une description plus détaillée, nous référons le lecteur à son article de 1989 (Maes 1989). Pour cela, la définition suivante est nécessaire : un acte est exécutable quand toutes ses préconditions sont satisfaites, c'est-à-dire quand tous les états nommés dans les préconditions sont vrais.

L'activation fluctue dans le réseau selon quatre mécanismes :

- Les désirs injectent[11] de l'activation aux actes qui annoncent dans leurs effets prévus qu'ils peuvent satisfaire le désir[12]. Ainsi, un acte pouvant satisfaire un désir, créant un effet désiré, reçoit de l'activation.
- Les états *qui sont vrais* envoient de l'activation (mais moins qu'un désir) aux actes qui ont besoin d'eux comme précondition. Ainsi, un acte qui est présentement exécutable, ou proche de l'être (c'est-à-dire que la plupart de ses préconditions sont satisfaites), reçoit de l'activation.
- Un acte qui n'est pas exécutable envoie de l'activation vers des actes qui annoncent dans leurs effets prévus qu'ils vont satisfaire les préconditions de l'acte. Ainsi, ces actes précurseurs reçoivent de l'activation et la probabilité qu'ils s'exécutent et permettent ainsi au premier acte de devenir exécutable, augmente.
- Un acte qui est exécutable envoie de l'activation vers les actes qui ont besoin des effets prévus de son exécution. Ainsi, la probabilité que l'acte prochain (dans la planification implicite) s'exécute, augmente.

L'ensemble de ces règles favorise le fait que l'activation provenant des désirs se propage dans le réseau jusqu'à ce qu'elle arrive à un acte exécutable, qui est ensuite sélectionné et le « plan » – la chaîne des actes entre le désir et cet acte – est exécuté, un acte après l'autre, pour satisfaire le désir.

La plus grande différence entre ce réseau et le modèle de Maes est que l'activation apportée par les désirs et les états n'est pas fixe, mais peut varier selon les besoins actuels de l'agent.

[11] *Injecter* et *envoyer* de l'activation revient à incrémenter le nombre qui représente l'activation de la cible. Les détails seront expliqués dans le chapitre VII.

[12] Comme nous avons déjà remarqué, nos *désirs* correspondent aux « *goals* » (*buts*) de Maes.

L'activation apportée par un état varie selon l'activation du micro-processus qui l'a déclenché. Pour un désir, des règles individuelles peuvent être spécifiées.

Après un tour de circulation d'activation, l'acte exécutable avec le plus haut niveau d'activation est choisi pour être exécuté. S'il n'y a pas d'acte exécutable, ou si le niveau d'activation de tout acte exécutable est en dessous d'un seuil, aucun acte n'est choisi pendant ce cycle cognitif.

3.5.7 Exécution de l'acte choisi

S'il y a un acte choisi, il s'exécute à cette étape-ci. Comme nous l'avons décrit dans la section sur les micro-processus, l'exécution de l'acte correspond essentiellement à l'exécution de ses micro-processus. Les effets de cette exécution peuvent être très variés, mais en général, ils préparent le prochain acte. Ceci peut être le lancement de micro-processus d'information dont la publication activera d'autres ressources, ou de demander à l'étudiant une information qui est nécessaire pour la suite. Par exemple, pour entreprendre un diagnostic, le premier acte lance un micro-processus d'information « Request=Diagnosis » pour que celui-ci se fasse publier et recrute les ressources nécessaires au diagnostic.

3.5.8 Exemple complet d'un cycle

Dans cette section, nous présentons un exemple de haut niveau d'un cycle cognitif. Nous allons utiliser ce même exemple dans les chapitres subséquents pour illustrer le fonctionnement détaillé.

Juste avant le début de ce cycle, les capteurs ont capté le message d'une collision. Lors de la perception, ce message est transformé en un arbre syntaxique. Des micro-processus de perception traversent cet arbre et mettent à jour des micro-processus d'information. Entre autre, un micro-processus d'information de proximité (« Collision=possible ») est créé avec une activation de 0,8728. Ce micro-processus est lié à trois autres micro-processus d'information : « Joint=WE », « Station-Element=S1P1TrussRight01 » et « Distance=6,95931 ».

Lors du raisonnement, l'activation du micro-processus de proximité décroît légèrement pour atteindre 0,8727 et il rejoint la scène. Puis, il forme une coalition avec les trois micro-

processus auxquels il est lié. Cette coalition a une force de 1,2398, ce qui lui permet de gagner la compétition et d'être publiée. Cette publication est entendue par l'état « Proximity » qui reconnaît l'information « Collision=possible » et se met à vrai.

Actuellement, dans le réseau des actes, l'activation circule et l'état « Proximity » en envoie à l'acte « Start Diagnosis ». De plus, cet acte est devenu exécutable, car l'état « Proximity » est son unique précondition. Comme il est le seul acte exécutable et que son niveau d'activation est supérieur au seuil, il est choisi pour exécution. Finalement, il est exécuté et il lance plusieurs micro-processus de raisonnement (dont un micro-processus d'arbitrage) qui vont délibérer, dans les cycles subséquents, sur la cause de ce rapprochement du joint WE à l'élément S1P1TrussRight01.

CHAPITRE IV

COMPARAISONS AVEC D'AUTRES TRAVAUX

4.1 Introduction

Dans ce chapitre, nous allons comparer notre architecture avec d'autres travaux.

Évidemment, ceci inclut une comparaison avec l'architecture de Franklin, où nous allons soulever et justifier les modifications que nous y avons apportées. Cette partie amorce le chapitre.

Elle est suivie de trois comparaisons avec des architectures connexes :

- le modèle du tableau noir, dont l'architecture de Franklin est partiellement inspirée,
- les agents BDI, parce qu'il s'agit d'une architecture générale d'agent,
- et finalement ACT-R, parce qu'il s'agit d'une modélisation de la cognition humaine,

Ce chapitre termine aussi la partie plus théorique de ce mémoire, pour laisser la place à la partie plus technique composée des trois derniers chapitres.

4.2 Comparaison avec IDA

Cette section est organisée autour des deux cycles cognitifs. Ainsi, le cycle cognitif de notre architecture (STC) sera comparé à celui de l'architecture de Stan Franklin (IDA). Nous allons aussi justifier les modifications que nous avons apportées à ce dernier.

Pour cette comparaison, cette section va référer au tableau 4.1, qui compare le cycle cognitif d'IDA à neuf étapes à notre cycle qui en compte sept. Les titres des étapes d'IDA sont tous repris tels quels de (Franklin, 2005) ; les guillemets ont été omis pour alléger le tableau.

Tableau 4.1 Comparaison des cycles cognitifs d'IDA et du STC

IDA (Franklin, 2005)	***STC***
1. Perception 2. Percept to preconscious buffer	1. Perception
3. Local associations	Prévu, mais ne pas encore conçu
4. Competition for consciousness	2. Raisonnement par les micro-processus de raisonnement 3. Compétition pour la « conscience »
5. Conscious Broadcast	4. Publication de la coalition gagnante
6. Recruitement of ressources	5. Recrutement de ressources
7. Setting goal context hierarchy 8. Action choosen	6. Choix de l'acte à prendre
9. Action taken	7. Exécution de l'acte choisi

La première différence qui ressort de ce tableau est que l'architecture STC ne présente qu'une seule étape pour la perception, tandis que IDA en présente deux. De plus, les implémentations des deux perceptions sont très différentes. L'architecture STC comporte uniquement une seule étape pour la perception, parce que sa perception fonctionne très différemment de celle de Franklin. Comme noté dans la section 1.3.2, IDA perçoit des courriels écrit en anglais. Pour comprendre le langage naturel, il utilise une structure qu'il appelle « *slipnet* » et cette structure crée le besoin d'une étape pour extraire les résultats. De plus, le « preconscious buffer » est nécessaire pour les mémoires que Franklin utilise. Comme la perception du système tutoriel conscient fonctionne très différemment (des micro-processus qui se promènent dans un arbre syntaxique), il n'y a simplement pas besoin d'une étape pour copier le résultat dans un tampon pré-conscient.

La deuxième différence est l'absence de la troisième étape de Franklin, qui fait intervenir les mémoires. Cette étape est prévue dans architecture STC et les mécanismes pour intégrer facilement des mémoires sont en place, mais nous n'avons pas encore conçu celles-ci.

La raison pour laquelle l'architecture STC n'utilise pas la « *sparse distributed memory* » d'IDA est qu'elle fonctionne très bien dans le domaine d'IDA, où toutes les informations enregistrées ont le même format : le numéro social du marin, son ancien poste, son nouveau poste et quelques dates pertinentes. Comme chaque entrée dans la mémoire est composée des mêmes champs, il est facile de l'encoder avec une chaîne de bits (le moyen de stockage de la sparse distibuted memory).

Dans le cas d'un tuteur par contre, les entrées sont beaucoup plus complexes et variées. Pour illustrer cela, voici deux situations : premièrement, un étudiant a demandé de l'aide après avoir eu une collision pendant un exercice de manipulation ; deuxièmement, un étudiant a donné trois mauvaises réponses pour un exercice de reconnaissance spatiale. À part l'identité de l'étudiant, les informations nécessaires pour décrire chaque situation sont très différentes et l'on ne peut donc pas allouer naïvement des champs dans une chaîne de bits. C'est pour cela que nous avons décidé, après une investigation du sujet, d'exclure le problème de la mémoire de notre projet de maîtrise et d'offrir la possibilité à quelqu'un d'autre d'implémenter une solution.

La troisième différence est principalement une question de présentation. Nous avons choisi de séparer la quatrième étape d'IDA en deux pour indiquer qu'il y a deux activités qui se passent pendant cette étape : premièrement, il y a les micro-processus de raisonnement qui réagissent sur le contenu de la scène, puis, il y a la formation des coalitions et une « compétition » entre eux. Selon Franklin, pendant sa quatrième étape : « Some of them [attention codelets] gather information, form coalitions and actively compete for access to consciousness. » (Franklin, 2005). Pour nous, l'action de recueillir des informations est différente de la compétition, et c'est pour cela que nous avons détaché une étape préliminaire de raisonnement.

De plus, la compétition du STC diffère légèrement de celle de Franklin. Premièrement, l'architecture STC n'a pas de micro-processus d'attention qui sont spécialisés à la formation de coalitions. Dans cette architecture, tout micro-processus sur la scène forme une coalition et participe à la compétition. Puis, pour l'architecture d'IDA, la force d'une coalition est simplement la moyenne des niveaux d'activation de ses membres, tandis que STC introduit le concept du micro-processus principal et un calcul différent pour obtenir la force de la

coalition : dans STC, la force d'une coalition s'établit par la somme de l'activation du micro-processus principal et de la moyenne des niveaux d'activation des autres.

Deux raisons justifient ce choix : premièrement, avec une simple moyenne, la coalition la plus forte est celle qui est constituée uniquement du micro-processus avec le niveau d'activation le plus élevé. Et nous préférons avoir des coalitions qui contiennent plus d'information. Deuxièmement, une moyenne simple défavorise les micro-processus représentant une collision. Quoique le micro-processus lui-même ait une très forte activation, il forme une coalition avec le micro-processus qui représente un module de la station, qui, à lui seul, n'est pas important. C'est pourquoi nous avons préféré que le micro-processus principal, le concept central de la coalition, influence davantage la force de la coalition.

Finalement, nous avons fusionné l'étape qui met en place le cadre de buts (*goal context hierarchy*) avec l'étape qui sélectionne l'acte à exécuter. Franklin prévoit cette étape dans son architecture pour instancier les enchaînements dans le réseau des actes. Ceci lui permet, pendant la sélection de l'acte, de ne considérer que la partie du réseau qui est pertinente à la situation. Notre implémentation du réseau des actes n'utilise pas ce mécanisme et elle n'a donc plus besoin d'une étape pour mettre en place le cadre de buts. Dans l'architecture STC, le cadre de buts s'exprime implicitement dans les niveaux d'activation du réseau des actes. Nous allons parler davantage de ce changement, ses implications et de sa justification dans le chapitre sur le réseau des actes.

4.3 Comparaison avec le modèle du tableau noir (*blackboard model*)

La scène est un répertoire central de l'information qui ressemble à un tableau noir. De plus, Franklin s'est inspiré de ce modèle pour l'implémentation de la conscience et c'est pour cela que nous allons comparer l'architecture STC avec le modèle du tableau noir dans cette section.

Les idées clés de ce modèle sont apparues dans le système de compréhension de la parole *Hearsay* en 1973. Ces idées ont été étendues pour donner l'architecture standard du tableau noir dans *Hearsay-II* , et ont ensuite été reprises dans une grande variété de systèmes en intelligence artificielle (Carver et Lesser, 1994).

Le modèle du tableau noir permet à plusieurs modules de coopérer à la résolution d'un problème (dans le cas de Hearsay, à la compréhension d'un enregistrement sonore). Le modèle du tableau noir prévoit trois composantes :

1. Le *tableau noir*, qui est un répertoire central de l'information. Il contient des données concernant le problème, et des hypothèses (des solutions partielles potentielles). Le tableau noir est divisé en niveau et dimension, ce qui facilite le repérage des informations.
2. Les *sources de connaissance* (« *knowledge sources* ») sont des modules indépendants qui examinent le contenu du tableau noir, formulent des nouvelles hypothèses à partir de ce contenu et les ajoutent au tableau noir. Chaque source de connaissance indique les conditions (décrivant les états du tableau noir pour lesquels elle est applicable) ainsi qu'une action (transformant le contenu du tableau noir). Aussi bien les conditions que l'action peuvent être plus complexes qu'une simple règle de production et faire intervenir des calculs importants sur les informations présentes sur le tableau noir. Ainsi, une source de connaissance est plus complexe qu'une simple règle de production et plus simple qu'un système d'expert.
3. Un module de *contrôle* qui détermine quelles sources de connaissance ont le droit de formuler de nouvelles hypothèses. Ce module est nécessaire pour éviter l'explosion combinatoire des hypothèses qui se produit quand chaque source de connaissance ajoute des hypothèses au tableau noir. Le contrôle doit donc déterminer les sources de connaissances les plus pertinentes pour la situation, et il ne permet d'agir qu'à elles.

Dans l'architecture STC, c'est la combinaison de la scène et de la conscience qui correspond au tableau noir. Tous les modules peuvent ajouter de l'information à la scène (en créant des micro-processus d'information qui la joignent – ce qui correspond à écrire sur le tableau noir), et tous les modules reçoivent la publication de la conscience (ce qui correspond à lire le tableau noir). Ainsi, la scène est un tableau noir dont l'accès en lecture est filtré par la conscience.

Les sources de connaissance correspondent aux micro-processus de raisonnement. Tous les deux réagissent sur l'information disponible et ajoutent de l'information au répertoire commun.

C'est sur le contrôle que l'architecture décrite dans ce mémoire diffère le plus du modèle du tableau noir. Il n'existe aucun contrôle direct sur le déclenchement des micro-processus. Par contre, en filtrant l'information par la conscience, moins d'information est disponible et donc moins de micro-processus sont applicables. C'est ainsi que la conscience résout le problème de l'explosion combinatoire des informations.

Ainsi, le contrôle dans l'architecture du tableau noir régit les sources de connaissances, tandis que celui de l'architecture d'agent conscient filtre les informations.

De plus, tandis que le contrôle dans l'architecture du tableau noir est centralisé, la « conscience » exerce un contrôle hybride : le niveau d'activation est déterminé de façon distribuée, tandis que la sélection de la coalition la plus forte est centralisée. Ceci permet d'adapter la partie distribuée (détermination du niveau d'activation) aux particularités de chaque micro-processus, tout en gardant un contrôle centralisé simple qui prend en compte les options disponibles. Ainsi, la « conscience » va choisir une coalition très faible si elle est la plus forte disponible.

L'atelier global de cette architecture est donc similaire au modèle du tableau noir, avec un nouveau mécanisme de contrôle qui est en partie distribué.

Finalement, l'architecture décrite dans ce mémoire ne se limite pas à l'atelier global. Elle contient aussi un module de perception et de planification, et elle prévoit des mémoires à long terme. Elle est donc une architecture d'agent complète, tandis que le modèle du tableau noir s'intéresse uniquement à la difficulté de résoudre un problème par la collaboration. Ceci amène à une utilisation très différente : le tableau noir est utilisé pour résoudre *un* problème à la fois (par exemple comprendre un enregistrement sonore), tandis que la conscience est utilisée pour mieux comprendre la situation actuelle, même si celle-ci n'est pas nécessairement un problème. De plus, la situation varie constamment suite à des nouvelles perceptions et aux actions prises par le réseau des actes.

4.4 Comparaison avec l'architecture BDI

Il est possible de tirer des parallèles entre l'architecture BDI et le modèle STC.

Les informations sur la scène (représentées par des micro-processus) et les états actifs sont une représentation du monde ; ils correspondent donc aux croyances du modèle BDI. Les

désirs du réseau des actes ressemblent à des intentions initiales : l'agent est engagé à les réaliser et ils guident le choix des autres intentions. Nous les nommons intentions *initiales*, parce qu'elles ne servent pas à satisfaire une autre intention, elles sont des buts en elles-mêmes. Les actes du réseau des actes sont à la fois des désirs (au sens BDI) et des intentions : dans l'architecture BDI, la différence est l'engagement. Dans le réseau des actes, l'activation ressemble à un degré d'engagement, mais il n'y a pas d'engagement absolu.

Le fait qu'il n'y a pas d'engagement absolu résout partiellement un problème de l'architecture BDI : la question à savoir quand l'agent se désengage-t-il d'une intention? Dans le réseau des actes, l'activation d'un acte (l'équivalent d'une intention) inutile ou irréalisable décroît. Il y a donc un désengagement graduel. Le problème du désengagement n'est pas complètement résolu, parce que la vitesse de désengagement doit toujours être paramétrée, mais il a été mitigé.

Nous avons affirmé au deuxième chapitre que notre architecture est mieux adaptée à un système tutoriel intelligent que le modèle BDI.

La raison qui nous amène à cette conclusion est le manque de contrôle sur la fonction de révision des croyances. Cette fonction doit, à partir des entrées des capteurs, déduire de nouvelles croyances. Dans un système tutoriel, ceci est un processus fortement complexe.

Prenons par exemple un danger de collision. Lorsqu'il est perçu, une première croyance émerge directement : il y a un risque de collision. Malheureusement, cette croyance n'est pas très utile pour le tuteur. Il a besoin de plus d'informations pour déterminer si ceci pose un problème. Un danger de collision pose toujours un problème. Mais pour une simple rotation du bras, il est plus difficile de déterminer cela. Quand l'étudiant dévie du meilleur chemin, est-ce acceptable, où est-ce que cela indique un problème? Quand le tuteur croit qu'il existe un problème, ses besoins d'informations augmentent : il veut connaître la raison à l'origine de la difficulté de l'apprenant. Il a donc besoin d'un diagnostic. Et ceci peut prendre du temps, en fait, trop de temps pour en accomplir un après chaque manipulation du bras.

L'architecture d'agent conscient résout ce problème grâce au mécanisme de la « conscience ». La perception ne fait qu'une reconnaissance rapide de la situation et – dans l'exemple donné – détecte le danger d'une collision. Le fait qu'il y a ce danger va être publié et par la suite, des micro-processus vont amener plus d'informations sur la scène. Aussi longtemps que ces

informations sont les plus activées, l'agent va continuer d'interpréter la situation en rajoutant de plus en plus de détails. Ainsi, de façon opportuniste, l'agent va améliorer sa compréhension de la situation aussi longtemps qu'il n'y a rien de plus important à faire.

Pour atteindre un comportement similaire avec l'architecture BDI, il faut ajouter des « intentions internes », c'est-à-dire des intentions dont l'exécution produit un effet interne. Par exemple une intention de faire un diagnostic dont l'effet est d'ajouter la conclusion de ce diagnostic aux croyances de l'agent.

De plus, l'idée d'entreprendre une intention « s'il n'y a rien de plus urgent à faire » s'unit mal avec l'architecture BDI dans laquelle l'agent est soit engagé à réaliser une intention, soit non engagé. Il est évidemment possible de classer les intentions selon leur importance (obligatoire, désirable, optionnelle)[1], et de réexaminer les intentions moins importantes plus souvent, ainsi que de choisir l'intention à exécuter selon sa priorité. Cependant, avec tous ces changements, il ne s'agit plus d'une architecture BDI, mais plutôt d'une nouvelle architecture, développée à partir de l'architecture BDI, qui risque de perdre les avantages de cette architecture (intuitive, avec un modèle formalisé).

4.5 Comparaison avec ACT-R

Autant qu'IDA cherche à implémenter et valider un modèle de la *conscience* humaine, autant ACT-R est une architecture qui implémente et valide un modèle de la *cognition* humaine.

Depuis ses débuts en 1993, ACT-R a été perfectionné et testé dans une multitude de domaines et ses performances ont été comparées à celles d'un humain pour vérifier que l'architecture produit des résultats similaires (Anderson, 2004). Ainsi, ACT-R est la simulation de la cognition humaine la mieux validée.

Le modèle d'ACT-R est basé sur un ensemble de tampons par lesquels des modules communiquent avec une partie centrale (voir la figure 4.1). Les modules ne peuvent pas communiquer entre eux, toute communication doit passer par les tampons de la partie centrale.

[1] L'architecture BOID (Belief-Obligation-Intention-Desir) est une architecture en partie inspiré de BDI (quoique les *désirs* ont une sémantique différente) qui inclut des obligations. Elle n'inclut toutefois pas d'intentions optionnelles. (Broersen, 2001).

Chaque tampon ne peut contenir qu'un seul élément, un « chunk ». Un tel « chunk » est constitué d'un nom et de liens étiquetés vers d'autres « chunks », formant ainsi un réseau sémantique.

La partie centrale modifie les tampons en suivant un ensemble de règles. Chacune de ces règles est composée de deux parties : d'abord les conditions, qui indiquent pour quelles configurations des tampons la règle est applicable, ensuite la portion action, qui indique comment elle modifie les tampons. Ces règles sont spécifiques à l'application, mais ACT-R fournit des méta-règles pour choisir la règle à exécuter, car il n'y a qu'une seule règle qui est exécutée à chaque cycle.

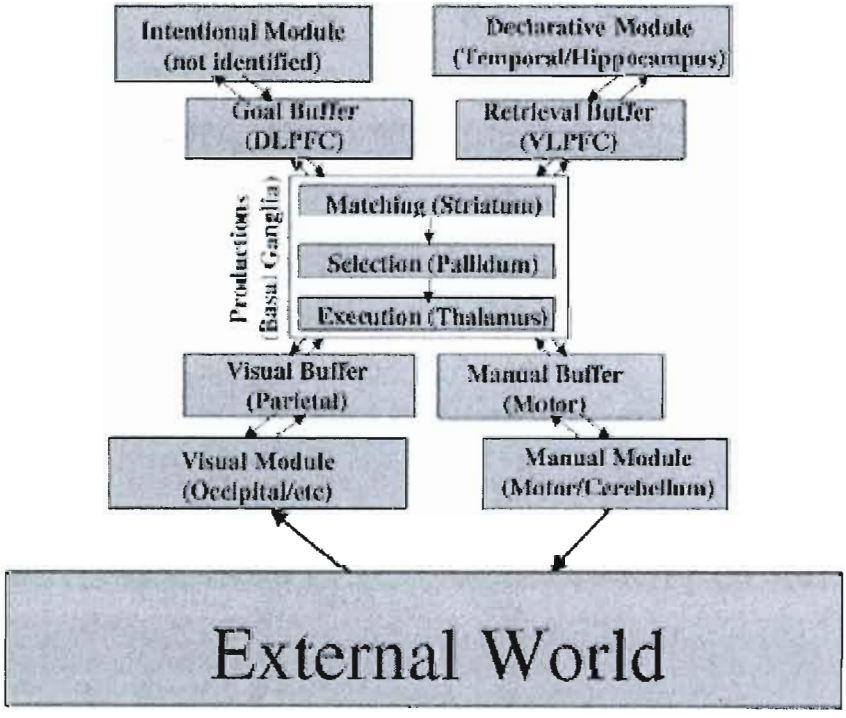

Figure 4.1 L'organisation de l'information dans ACT-R 5.0 (Anderson, 2004)

Il y a un nombre de similitude entre ACT-R et l'architecture STC.

Les « chunk » correspondent à aux micro-processus d'information. Comme ils ont une étiquette sur les liens (et non sur les nœud), ils offrent une expressivité supérieure à la nôtre.

Nous sommes en train d'évaluer les impacts d'une modification de l'architecture STC dans ce sens.

La partie centrale correspond vaguement à l'atelier global. Les tampons correspondent à la mémoire de travail (la scène). Et le module de règles fait allusion à la conscience : il ne peut déclencher qu'une règle à la fois, comme la conscience ne peut publier qu'une coalition à la fois.

Il y a cependant une première différence, similaire à celle qui distingue l'architecture STC du modèle du tableau noir : ACT-R exerce un contrôle sur le choix de la *règle*, tandis que la conscience exerce un contrôle sur le choix de *l'information*.

Il y a aussi une différence dans la manipulation de données complexes (par exemple, provenant de la perception). Dans l'architecture STC, toutes les données de la perception arrivent sur la scène et sont publiées l'une après l'autre, tant qu'il n'y a rien de plus important. Dans ACT-R, un seul percept arrive dans la partie centrale et une règle doit être déclenchée pour demander plus d'information du module de perception. Ainsi, le module central de ACT-R a plus de contrôle sur la perception que la conscience de l'architecture STC.

De plus, la scène ne limite pas le type des informations qu'elle peut contenir, ce qui peut autant constituer un avantage qu'un désavantage. En offrant la possibilité d'avoir sept micro-processus d'information concernant la perception en même temps sur la scène, ceci permet de monopoliser la scène pour traiter rapidement un problème. Avec ACT-R, chaque module est limité à son (ou ses) tampon(s). Ceci permet de formuler des règles plus simples, car chaque tampon est nommé et réservé à une utilisation unique.

De plus, les utilisations multiples d'ACT-R ont permis de déterminer les meilleures valeurs pour les paramètres de cette architecture. Ainsi, avec chaque nouvelle version, le nombre de paramètres diminue. IDA est une architecture trop récente pour avoir atteint ce niveau de maturité et il existe encore une quantité importante de paramètres à considérer.

Finalement, nous pensons que l'architecture proposée dans ce mémoire est néanmoins une meilleure structure pour un système tutoriel intelligent. ACT-R est une architecture mature qui guide bien le développeur, mais ceci se paye par un manque de liberté. Il faut utiliser les règles d'ACT-R et modéliser l'application à un niveau extrêmement fin (par exemple, pour additionner deux nombres avec plusieurs chiffres, changer de colonne est une action en

ACT-R), ce qui est difficile pour un système complexe. Tandis que l'architecture STC est plus malléable et permet d'utiliser des actions de plus haut niveau, ce qui facilite le développement d'un système complexe.

CHAPITRE V

ATELIER GLOBAL

5.1 Introduction

Ce chapitre est le premier des trois chapitres qui présentent l'implémentation concrète de notre système. Cette présentation commence avec l'atelier global parce que les deux autres modules – la perception et le réseau des actes – utilisent les éléments de l'atelier global, tandis que l'atelier global peut exister sans les deux autres. (Quoiqu'il n'aurait rien sur quoi raisonner.)

Ces trois chapitres sont des chapitres techniques qui décrivent le fonctionnement du système en termes de classes et formules. Nous supposons donc que le lecteur est familier avec les concepts de base de la programmation et avec le langage de programmation Java.

Le but de ces chapitres techniques est d'aider à la compréhension de notre code, afin de faciliter la réutilisation de l'architecture pour de nouvelles applications.

Ce chapitre présente le module central : l'atelier global. C'est un module parfaitement générique qui ne contient aucun code spécifique à une application, et il est possible de le compiler et exécuter indépendamment. En effet, ce module est un patron d'applications (*framework*) « conscient ».

Il fournit les mécanismes à la base de l'aspect « conscient » de l'architecture, c'est-à-dire la compétition et la publication. Ce module correspond au théâtre, à la scène et au spot de la conscience, proposés par la métaphore de Baars. Il fournit le cadre dans lequel agissent les micro-processus qui, eux, proviennent des autres modules et de l'application.

Dans ce chapitre nous allons d'abord présenter les services offerts par ce module, suivi d'une démonstration de ses capacités d'extension.

5.2 Services offerts

Comme nous venons de le dire, le module de l'atelier global fournit les mécanismes pour la compétition et la publication, ce qui nécessite une implémentation du théâtre et de la scène. De plus, il encadre l'étape du raisonnement en fournissant une classe de base pour les micro-processus de raisonnement. Finalement, il offre quatre classes qui implémentent les micro-processus d'information.

Dans cette section, nous allons uniquement présenter les services offerts par l'atelier global, et nous allons expliquer dans la section suivante comment utiliser et étendre ce cadre.

5.2.1 Raisonnement par les micro-processus de raisonnement

Les micro-processus de raisonnement sont, d'une certaine façon, l'entrée de l'atelier global. L'atelier global fournit une classe de base (et des sous-classes), mais il ne crée aucun micro-processus. Ce sont les autres modules (et l'application[1]) qui créent des micro-processus de raisonnement, et l'atelier forme le cadre dans lequel ces micro-processus agissent.

La classe de base des micro-processus de raisonnement implémente les comportements génériques de l'étape du raisonnement : la décroissance progressive de l'activation et les mécanismes pour joindre et quitter la scène. Les particularités de chaque micro-processus sont représentées par des méthodes abstraites pour en faciliter l'extension. Celles-ci seront présentées dans la section 6.3.2.

Pour faire agir le micro-processus à chaque cycle cognitif, Franklin utilise un processus léger (*thread*) pour chaque micro-processus, ainsi que pour la plupart des autres éléments de son architecture (par exemple chaque élément du réseau des actes). Quoique ceci correspond bien au traitement distribué et en parallèle qui existe dans le cerveau humain, c'est hautement inefficace dans un ordinateur. De plus, le débogueur de Java n'arrive plus à déboguer une application avec 200 processus légers concurrentiels[2], ce qui limite considérablement le développement. C'est pour cela que nous avons abandonné l'idée d'un processus léger par

[1] La méthode `main` de l'application peut créer des micro-processus de raisonnement et d'information et les ajouter au théâtre avant même de démarrer le cycle cognitif. Ces micro-processus ne proviennent donc pas d'un autre module, mais directement de l'application.

[2] Et déjà un exemple simpliste (compter jusqu'à cinq) génère autant de processus légers.

micro-processus et que nous utilisons plutôt une boucle dans le théâtre qui appelle chaque micro-processus une fois par cycle.

5.2.2 Compétition pour la conscience

Dans notre implémentation, c'est la scène qui est en charge de la compétition, avec deux classes auxiliaires, le *gestionnaire des coalitions* et le *contrôleur du spot*, deux titres tirés des publications de Franklin. À chaque cycle, la scène appelle ces deux classes auxiliaires pour mener la compétition. D'abord, elle appelle le gestionnaire des coalitions. Celui-ci initie la création d'une coalition pour chaque micro-processus sur la scène. Ensuite il retourne la collection de coalitions à la scène qui la transmet au contrôleur du spot. Celui-ci retient simplement la coalition avec la plus grande force.

De plus, il existe un seuil de force minimal qu'une coalition doit avoir pour être publiée. Quand il n'y a aucune coalition dont la force dépasse ce seuil, rien n'est publié. Ceci évite de recruter des ressources inutilement pour une coalition qui a juste été publiée parce qu'il n'y avait rien d'autre. Quand rien n'est publié, le seuil décroît progressivement pour que, ultimement, une coalition soit publiée dans un cycle cognitif subséquent. Dès qu'il y a une publication, le seuil revient à sa valeur initiale pour continuer d'empêcher une publication peu utile.

La valeur de ce seuil et la vitesse avec laquelle il décroît font partie des nombreux paramètres de l'architecture STC. La force d'une coalition peut varier entre 2 et 0[3]. Présentement, l'implémentation utilise un seuil de 0.9, ce qui se situe dans les alentours de la force d'une coalition décrivant la perception d'un changement.

La logique derrière la formation des coalitions et celle du calcul de sa force sont regroupées dans la classe *coalition*. Elles ont été décrites dans la section 3.5.3.

5.2.3 Publication de la coalition gagnante

Pour la publication, c'est le *gestionnaire de la publication* qui s'en occupe. Cette classe implémente le patron publication-abonnement (*publish-subscribe*) et l'acte de la publication

[3] La force de la coalition s'établit par la somme de l'activation du micro-processus principal (qui varie entre 1 et 0) et de la moyenne des activations des autres micro-processus (qui varie aussi entre 1 et 0 aussi).

consiste à envoyer la coalition gagnante à tous les abonnés. Ceci équivaut à l'envoyer à tout le système, et pour des raisons de performance, la publication n'est envoyée qu'aux parties intéressées. Un micro-processus d'information, par exemple, existe simplement et ne s'intéresse pas à la publication. Un micro-processus d'arbitrage par contre accomplit son travail grâce aux informations publiées et s'abonne donc à la publication.

5.2.4 Micro-processus d'information

Le module de l'atelier global fournit quatre sous-classes concrètes pour les micro-processus d'information. Ces classes concrètes peuvent être utilisées telles quelles par les autres modules – contrairement à la classe de base des micro-processus de raisonnement qui doit être étendue pour chaque besoin spécifique. Ces quatre classes sont :

- `FixedInformationCodelet` : Cette classe sert à produire les micro-processus qui représentent toujours la même information. Seuls les liens et l'activation du micro-processus peuvent donc varier. Nous pouvons donner l'exemple des micro-processus représentant chaque joint du bras canadien, il se trouve donc un micro-processus qui représente le joint WE. Sa clé (« Joint ») et sa valeur (« WE ») sont fixes et ne changent pas.
- `VariableInformationCodelet` : Cette classe sert à produire les micro-processus à valeur variable, exprimée par une chaîne de caractères. L'implémentation l'utilise pour les micro-processus qui indiquent une collision. Leur clé est invariable (« Collision »), mais la valeur peut changer entre « possible » (quand le système perçoit qu'un joint est près d'un élément de la station dans le micro-monde) et « effective » (quand le joint touche l'élément).
- `FloatInformationCodelet` : Cette classe sert à produire les micro-processus d'information qui ont une valeur variable en terme d'un nombre réel. Elle est utilisée, entre autres, pour les micro-processus qui représentent le degré de rotation d'un joint. Leur clé est invariable (« Rotation »), mais la valeur change quand le système perçoit un changement de la rotation du joint dans le micro-monde. Quoiqu'il sera possible d'utiliser la classe `VariableInformationCodelet` pour ces micro-processus, cette classe spécialisée permet d'accéder facilement la valeur numérique pour faire des calculs.

- `OneShotInformationCodelet` : Les instances des trois classes précédentes continuent d'exister dans le théâtre après avoir été publiées. Ainsi, le micro-processus « Joint=SY » va représenter ce joint aussi longtemps que le système s'exécute. Ceci permet de retrouver toute information concernant ce joint grâce à une référence vers ce micro-processus. Parfois, cependant, le besoin d'envoyer une information volatile survient. Cette classe satisfait ce besoin : elle est un `FixedInformationCodelet` qui cesse d'exister après avoir quitté la scène.

Toutes ces classes possèdent une méthode `wasPerceived()` qui prend en paramètre une nouvelle activation et (pour ceux qui ont une valeur variable) la nouvelle valeur. Cette méthode indique au micro-processus sa pertinence dans la situation, le degré de pertinence étant donné par l'activation fournie. En réaction à cela, le micro-processus d'information va joindre la scène au prochain cycle pour se faire publier afin de communiquer l'information pertinente à tout le système. Nous avons choisi ce nom parce que cette méthode est appelée suite à une perception. Mais son utilisation n'est pas limitée à la perception et tout client d'un micro-processus d'information va vouloir utiliser cette méthode.

Voici maintenant comment une application peut utiliser un micro-processus d'information.

Pour l'utiliser de façon standard (nous allons décrire l'autre façon dans quelques instants), il faut d'abord appeler le constructeur et ensuite la méthode `start()`, qui fait participer le micro-processus au cycle cognitif. Dans la plupart des cas, le programmeur va vouloir appeler la méthode `linkTo()` ensuite pour créer des liens avec d'autres micro-processus. Ceci pourra aussi se faire plus tard, si, au moment de la création du micro-processus, son contexte n'est pas déterminé ou que son contexte est dynamique. Nous trouvons un exemple de contexte dynamique avec le micro-processus représentant un écran : la caméra montrée sur un écran peut changer au cours d'une séance, ce qui obligera à remplacer le lien qui existait avec le micro-processus représentant l'ancienne caméra. La méthode `removeLinkTo()` existe pour défaire des liens.

Quand le micro-processus d'information devient pertinent, il faut appeler la méthode `wasPerceived()` décrit plus haut. C'est aussi cette méthode qu'il faut appeler quand le micro-processus redevient pertinent à un moment ultérieur.

La deuxième façon d'utiliser un micro-processus d'information est comme attribut à un autre micro-processus. Un micro-processus ne peut contenir qu'un couple de chaînes de caractères. Souvent, cela ne suffit pas pour représenter l'information et il faut attacher d'autres micro-processus au premier. Parfois, cette information est un concept digne d'un micro-processus d'information normal. Par exemple, le micro-processus pour une collision est lié à celui du joint qui y est impliqué. Dans ce cas, l'implémentation crée simplement un lien entre ces deux micro-processus. Par contre, il arrive aussi que l'information additionnelle n'est qu'un attribut de plus. Par exemple, pour un mouvement du bras, l'amplitude du mouvement est un attribut. Une amplitude toute seule ne présente qu'un chiffre sans signification. « Mouvement », par contre, possède une signification propre, même s'il n'est pas spécifié ce qui bouge.

Pour un tel attribut, il est possible de créer un micro-processus d'information sans appeler ensuite la méthode `start()`. Ainsi, le micro-processus ne participe pas au cycle cognitif et ne sert qu'à compléter un autre micro-processus possédant un lien vers ce micro-processus d'attribut.

5.3 Points d'extension

Dans cette section, nous allons décrire comment étendre l'atelier global pour l'adapter à une application spécifique.

5.3.1 Théâtre et scène

Comme, dans la métaphore de Baars, le *théâtre* regroupe tout, nous avons choisi de faire du théâtre l'objet central du système qui contient et appelle (directement ou indirectement) tous les autres objets. Le théâtre est aussi l'objet qui contrôle le cycle cognitif et qui appelle chaque étape de ce cycle. Encore, il est une des deux classes centrales pour l'extension de ce module, l'autre étant la classe de base des micro-processus de raisonnement.

La figure 5.1 illustre la place du théâtre dans l'architecture : il contient des micro-processus (« Codelets ») et il fait le lien entre la perception, le réseau des actes, les mémoires (dès qu'elles seront implémentées) et la scène.

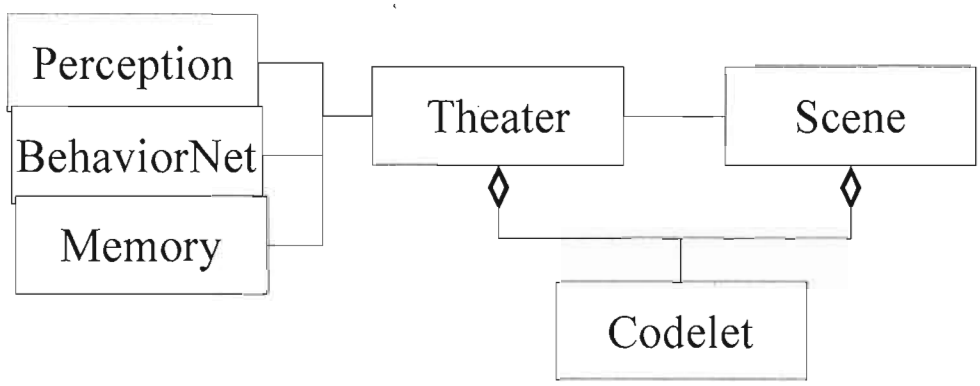

Figure 5.1 La place du théâtre dans l'architecture

Il y a deux façons d'étendre le module de l'atelier global : la plus habituelle est d'y ajouter les trois modules additionnels (la perception, le réseau des actes et les mémoires), l'autre – beaucoup plus rare – consiste à étendre directement le théâtre ou la scène.

Le théâtre est un singleton puisqu'il ne peut en exister qu'une seule instance. Pour permettre une configuration de cette instance, il y a deux méthodes statiques qui permettent de le configurer, une pour chaque façon d'étendre l'atelier global. Ces deux méthodes doivent être appelées avant qu'une instance du théâtre soit créée, c'est-à-dire avant que la méthode `Theater.getInstance()` soit appelée pour la première fois.

Pour ajouter des modules à l'atelier global, on appelle la méthode `setPlugInManger()`. La classe `PlugInManager` est une classe très simple qui ne contient que trois méthodes, une pour chaque module. De base, chaque méthode retourne un objet vide (*null object*) qui implémente le module avec des instructions ineffectives (*no ops*). Ainsi, pour ajouter un ou plusieurs modules, l'application cliente doit créer une sous-classe de `PlugInManager` qui retourne une implémentation fonctionnelle pour les modules implémentés. Dans l'implémentation exploratoire, ceci est fait pour la perception et le réseau des actes, tandis que la méthode `getMemory` continue de retourner un objet vide (ceci sera changé dès que les mémoires seront implémentées).

Il peut être utile d'étendre le théâtre ou la scène, surtout pour visualiser le comportement du système, afin de le comprendre mieux. Dans cette implémentation, la scène est étendue pour qu'elle signale l'arrivée et le départ des micro-processus, afin d'afficher les micro-processus présentement sur la scène dans une fenêtre (voir la figure 5.2). Pour permettre ceci, la classe

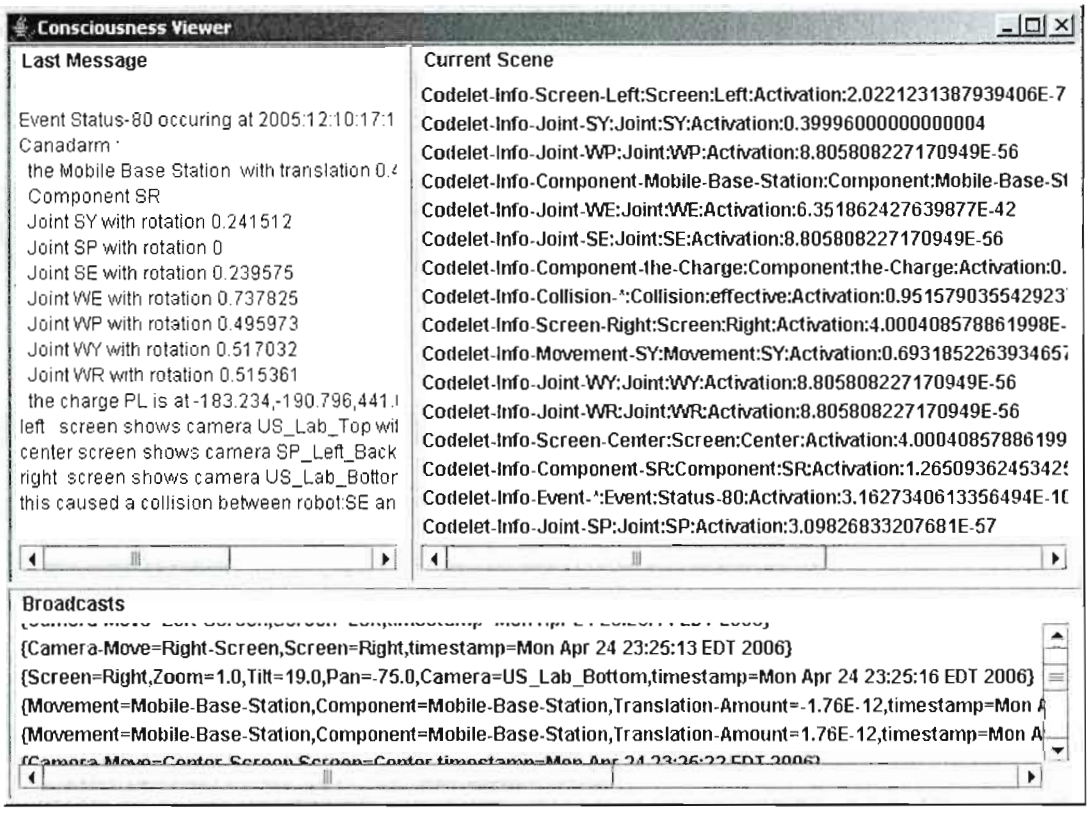

Figure 5.2 Visualiseur de la scène

`Theater` offre la méthode `setFactory(TheaterFactory factory)` qui permet de contrôler la création du théâtre et de la scène. La classe `TheaterFactory` possède deux méthodes : une pour créer le théâtre et une pour créer la scène. En étendant cette classe pour qu'elle crée une instance d'une sous-classe pour le théâtre, on peut étendre ce singleton. Une telle extension du théâtre n'existe pas dans l'implémentation exploratoire.

5.3.2 Micro-processus de raisonnement

Une application va vouloir créer ses propres micro-processus de raisonnement pour les réflexions nécessaires dans son domaine. Pour garantir que ces micro-processus de raisonnement se conforment aux principes de l'architecture, l'atelier global fournit la classe de base `Codelet`. Cette classe abstraite implémente les activités communes de l'étape de

raisonnement du cycle cognitif et impose des méthodes abstraites pour l'adaptation des sous-classes.

Dans cette section, nous allons présenter ces méthodes, préciser quand elles sont appelées, donner des conseils pour leur implémentation et illustrer à l'aide de l'exemple des micro-processus d'arbitrage comment une application peut les étendre. Nous allons présenter les méthodes selon l'ordre dans lequel elles vont être appelées lors d'une exécution normale.

Les méthodes appelées quand le micro-processus est sur la scène sont différentes de celles appelées quand il se trouve en dehors de la scène. Comme un micro-processus commence sa vie en dehors de la scène, nous allons d'abord présenter les trois méthodes appelées lorsque le micro-processus se trouve hors de la scène :

- `actOutsideScene()` est la première méthode appelée. L'intention de cette méthode est de permettre au micro-processus d'agir de façon générale, sans abuser d'une autre méthode pour cela. Par exemple, un micro-processus d'arbitrage utilise cette méthode pour mettre à jour le nombre de cycles qui se sont écoulés et pour calculer son « impatience ».
- `wantJoinScene()` est appelée ensuite. Dans cette méthode, le micro-processus vérifie s'il veut joindre la scène, s'il est pertinent à la situation. Un micro-processus d'arbitrage vérifie s'il peut annoncer le résultat de la délibération parce qu'une proposition est demeurée sans opposition assez longtemps (en fonction de « l'impatience »).
- `prepareToJoinScene()` finalement permet au micro-processus de s'ajuster avant de joindre la scène. Un micro-processus d'arbitrage va déterminer son niveau d'activation, et créer les liens vers la proposition acceptée, ainsi formant le contexte nécessaire.

Ainsi, le micro-processus arrive sur la scène.

Quand le micro-processus gagne la compétition, sa méthode `getBroadcastInfo()` est appelée. Cette méthode doit retourner un couple de chaîne de caractères, dont la première est l'étiquette de l'information et la deuxième sa valeur. Un micro-processus d'arbitrage pour le diagnostic retourne toujours le couple « Response=Diagnosis » pour indiquer qu'il s'agit d'une réponse à une demande de diagnostic. Les autres informations (la proposition choisie et le problème sur lequel porte le diagnostic) sont données par les liens, créés dans la

méthode `prepareToJoinScene()`. Il importe de créer les liens **avant** de joindre la scène, car ce sont eux qui dirigent la création des coalitions.

Au cycle suivant, une de deux méthodes est appelée. Si le micro-processus reste sur la scène (il n'a pas été publié toute suite et son activation n'est pas trop basse), la méthode *`actOnScene()`* est appelée. Celle-ci permet au micro-processus d'agir sur la scène. Très souvent, au moins une partie de cette méthode va être similaire à `actOutsideScene()`. Par exemple, un micro-processus d'arbitrage continue de compter le nombre de cycles écoulés. Au delà des actions communes, quand il est sur la scène, il vérifie qu'il n'ait pas reçu une opposition à la proposition qu'il prévoit d'annoncer comme réponse. Si tel est le cas, il quittera la scène afin d'éviter de publier une fausse réponse et pour attendre une meilleure proposition.

Lorsque le micro-processus quitte la scène, la méthode *`cleanupAfterLeavingScene()`* est appelée et ce, indépendamment de la raison (que ce soit qu'il a été publié, que son activation soit trop faible, ou qu'il se retire volontairement comme le micro-processus d'arbitrage qui reçoit une opposition à sa proposition). Comme le nom de la méthode indique, elle est appelée *après* que le micro-processus ait quitté la scène. L'implémentation de cette méthode peut donc présumer que le micro-processus n'est plus sur la scène. En général, cette méthode devrait défaire les effets de `prepareToJoinScene()`. Justement, un micro-processus d'arbitrage utilise cette méthode pour défaire le lien qu'il a vers la proposition gagnante. De plus, un micro-processus d'arbitrage choisit de mourir après (et uniquement après) qu'il ait été publié.

Finalement, lorsqu'un micro-processus meurt, la méthode *`cleanupWhileDying()`* est appelée. Ceci permet au micro-processus de défaire ce qu'il a fait dans son constructeur. Cette méthode est appelée en dernier, après que le micro-processus ait quitté le théâtre. Si un micro-processus meurt pendant qu'il se trouve sur la scène, il quittera d'abord la scène, puis `cleanupAfterLeavingScene()` est appelée, ensuite il quitte le théâtre et finalement *`cleanupWhileDying()`* est appelée. Un micro-processus d'arbitrage s'abonne à la publication lors de sa création et il se désabonne dans son implémentation de cette méthode.

5.3.3 Publication

Comme nous avons dit dans la section 5.2.3, la publication utilise le patron publication-abonnement. Pour s'abonner à la publication, un objet doit implémenter l'interface `BroadcastListener` et puis s'abonner auprès du gestionnaire de la publication. Comme celui-ci est un singleton, ceci se fait avec l'appel :

```
BroadcastManager.getInstance().addBroadcastListener(this);
```

Pour se désabonner (ce qui est nécessaire à la fin de vie pour que l'objet puisse être réclamé par le nettoyeur (*garbage collector*)), il suffit d'appeler la méthode `removeBroadcastListener()`.

Comme nous venons d'écrire, un micro-processus de raisonnement qui s'intéresse à la publication s'abonne préférablement dans son constructeur et se désabonne dans la méthode `cleanupWhileDying()`.

Finalement, l'abonnement à la publication ne se limite pas aux micro-processus. La publication s'adresse à tout le système et toute composante peut donc s'y abonner.

5.4 Exemple

Nous avons illustré comment une application peut utiliser les services offerts par le module de l'atelier global et comment elle peut l'étendre. Maintenant, nous allons reprendre l'exemple d'un cycle cognitif donné au chapitre 3 et détailler davantage les étapes 2, 3 et 4 qui sont prises en charge par l'atelier global.

La première étape (la perception) sera détaillée au prochain chapitre. Par conséquent, nous n'allons décrire ici que la situation telle qu'elle se trouve au début de la deuxième étape, sans spécifier comment cette situation a été créée.

Au début de la deuxième étape, il y a 34 micro-processus dans le théâtre, dont onze se trouvent sur la scène. Tous ces micro-processus sont des micro-processus d'information créés par la perception[4]. Nous allons donc décrire l'origine de ces micro-processus dans le

[4] Ceci résulte du fait que le scénario est très simple. Une application complète va avoir d'autres sortes de micro-processus dans le théâtre et sur la scène.

chapitre suivant. Comme ce sont des micro-processus d'information, la seule action qu'ils peuvent prendre pendant l'étape de raisonnement est de rejoindre ou quitter la scène.

Ainsi, comme une nouvelle perception vient de se produire, cinq micro-processus d'information joignent la scène pendant l'étape de raisonnement. Ils se déclarent pertinents, parce que la perception vient d'appeler leur méthode `wasPercieved()`.

Un de ces cinq micro-processus est un `VariableInformationCodelet`, dont l'information a comme clé « Collision » et comme valeur « possible ». Ce micro-processus représente un rapprochement (donc une possibilité de collision) entre le joint WE du bras et l'élément S1P1TrussRight01 de la station spatiale. Il est donc lié aux deux `FixedInformationCodelets` qui les représentent. De plus, il a un attribut qui est la distance entre ces deux objets. Ainsi, il est lié à un `FloatInformationCodelet` avec l'information « Distance=6.95931 ».

Aucun micro-processus ne quitte la scène pendant cette étape. Ceci s'explique par le fait qu'au cycle précédent, aucune coalition n'a dépassé le seuil de force minimal pour être publiée, et il n'y a donc pas eu de publication. Ainsi, aucun micro-processus ne vient d'être publié et aucun micro-processus ne quitte la scène.

À la fin de la deuxième étape, il y a les 16 micro-processus sur la scène qui sont visibles dans la figure 5.3.

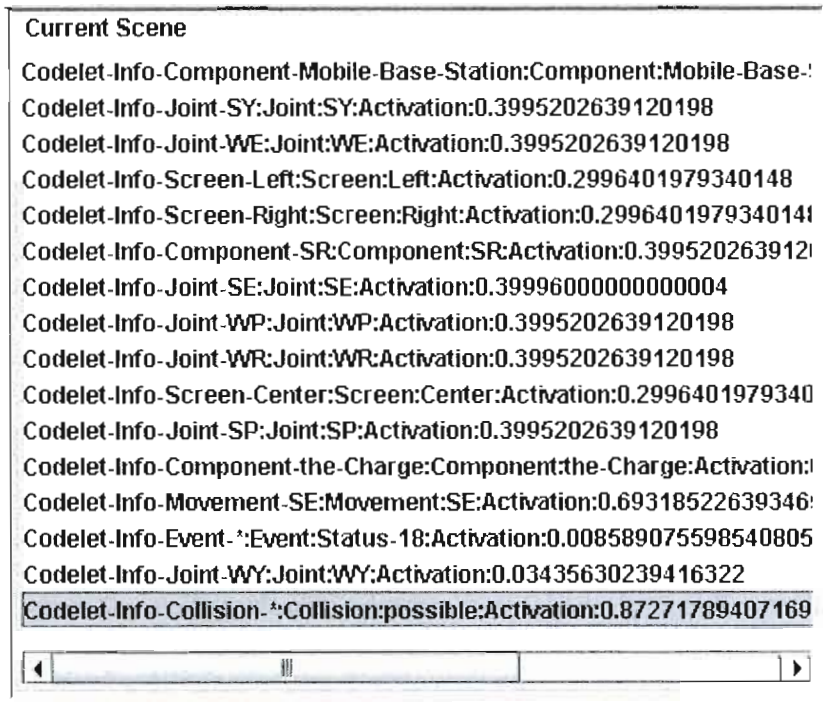

Figure 5.3 Contenu de la scène après l'étape deux de l'exemple

Lors de la troisième étape, la compétition, chacun de ces 16 micro-processus forme une coalition avec les micro-processus auxquels il est lié.

La coalition la plus faible (avec une force de 0,0172) est formée par le micro-processus d'information « Event=Status-18 » et son attribut « Event-Timestamp ». Ce micro-processus indique simplement qu'il y a eu un évènement perçu et nous avons décidé de lui attribuer une très faible activation, parce que le système s'intéresse beaucoup plus aux détails de l'évènement qu'au fait qu'il y a eu un évènement. Comme l'attribution de l'activation est faite dans le module de la perception, nous allons décrire nos choix davantage dans le chapitre suivant.

La deuxième coalition la plus faible (0,0515) est celle formée par le micro-processus représentant le joint WY et son attribut décrivant sa rotation. Ceci est dû au fait que la position de ce joint n'a pas changé depuis la dernière fois qu'il a été publié.

La coalition gagnante concerne la proximité. Celle-ci possède une force de 1,2968, ce qui lui permet de remporter la compétition. Sa rivale la plus proche concerne le mouvement du joint SE, avec une force de 1,2398.

La force de la coalition gagnante est la somme de 0,8727 (provenant du micro-processus principal « Collision=possible ») et de 0,4241, qui provient de la moyenne entre 0,0[5] (l'activation du micro-processus « Station-Element=S1P1TrussRight01 »), 0,3995 (« Joint=WE ») et 0,8728 (« Distance=6,95931»).

Comme cette force se situe au-dessus du seuil minimal (qui est de 0,81 parce qu'il avait un cycle sans publication), elle est publiée à tous les abonnés de la publication lors de l'étape 4. Comme, pendant ce cycle cognitif, il n'y a aucun micro-processus abonné à la publication, la publication ne va être entendue que par les états du réseau des actes. Leur réaction suite à cette publication sera décrite au chapitre 8.

[5] Les éléments de la station ne sont pas utilisés dans la version actuelle de l'implémentation exploratoire. C'est pour cela qu'ils ont une activation de 0,0.

CHAPITRE VI

PERCEPTION

6.1 Introduction

Contrairement à l'atelier global que nous venons de présenter, le module de la perception est spécifique à l'application. Il a été conçu et développé spécifiquement pour percevoir l'environnement virtuel du bras canadien.

Le module de perception est le module qui varie le plus avec l'application, parce qu'il est spécifique à ce qui est perçu, donc à l'environnement de l'agent – par exemple, la perception du système tutoriel conscient est très différente de celle d'IDA, qui interprète des courriels. Nous l'avons donc conçue indépendamment.

Contrairement au chapitre précédent, organisé autour de l'aspect de la généricité de l'atelier global, le présent chapitre se structure autour de l'aspect « transformation » de la perception. Le rôle de la perception est de transformer : transformer ce qui est capté en un format interne, les micro-processus d'information.

Nous allons donc commencer ce chapitre en spécifiant l'entrée de la perception, la forme sous laquelle les capteurs lui fournissent l'information. Ensuite, nous allons présenter la sortie de la perception, pour que le lecteur comprenne l'objectif des transformations qui seront présentées par la suite.

6.2 Entrées

Pour des raisons historiques (le projet STC est la fusion de deux projets initialement distincts), le simulateur de la station spatiale est écrit en C++, et l'implémentation de notre

architecture en Java. Après une investigation des possibilités de communication entre ces deux langages, nous avons choisi de communiquer par envoi de messages textuels.

Les capteurs fournissent donc des chaînes de caractères à la perception. Pour faciliter la compréhension par un humain, le système utilise des messages ressemblant à l'anglais. Un exemple d'un tel message est donné à la figure 6.1.

```
Event Status-18 occuring at 2005:12:10:17:3:13.265
Canadarm :
  the Mobile Base Station  with translation 0
  Component SR
  Joint SY with rotation 0.5
  Joint SP with rotation 0.2
  Joint SE with rotation 0.575
  Joint WE with rotation 0.7
  Joint WP with rotation 0.5
  Joint WY with rotation 0.5
  Joint WR with rotation 0.5
  the charge PL is at -91.1979,-563.361,326.017
left   screen shows camera US_Lab_Top with 1X zoom,
yaw -63, pitch 0
center screen shows camera SP_Left_Back with 1X zoom,
yaw 0, pitch -12
right  screen shows camera ZCM_Bottom with 1X zoom,
yaw 0, pitch 0
this caused the danger of a collision between robot:WE
and element S1P1TrussRight01 (distance : 6.95931)
```

Figure 6.1 Exemple d'un message envoyé par le simulateur

Nous avons rejeté un format XML, parce qu'il réduirait la lisibilité des messages. Néanmoins, même si les messages utilisent des mots anglais, ils suivent une grammaire formelle, ce qui permet un traitement efficace par ordinateur.

Un deuxième choix que nous avons fait consiste à limiter le contenu des messages à la description de *l'état* actuel de l'environnement. Comme l'environnement observé est un micro-monde virtuel, il aurait été plus efficace de communiquer uniquement les changements de l'environnement. Par contre, un des buts du projet est de rester proche d'un modèle de l'esprit humain ; et un humain ne capte que l'état actuel de son environnement. Le cerveau

doit lui-même détecter les changements. C'est pour cela que le simulateur envoie à chaque fois une description de l'état actuel.

Le simulateur envoie un tel message au tuteur après chaque action de l'utilisateur. Il est capté par un processus léger dédié à cette tâche et mis dans une file d'attente. Ainsi, ni le simulateur, ni le cycle cognitif doivent attendre que le message soit traité. Au début d'un cycle cognitif, la perception traite tous les messages contenus dans cette file[1].

Avant de présenter le fonctionnement interne de la transformation, nous allons d'abord présenter le résultat attendu.

6.3 Sorties

Le résultat de la transformation par la perception est un ensemble de micro-processus d'information qui vont ensuite œuvrer dans l'atelier global.

Nous rappelons qu'un micro-processus d'information est défini par une étiquette, une valeur, des liens vers d'autres micro-processus et par un niveau d'activation. Quand la perception crée (ou modifie) un micro-processus, elle doit déterminer chacun de ces quatre éléments. Le résultat de la perception est donc un réseau de micro-processus d'information semblable à celui de la figure 6.2.

[1] Actuellement, dû à la fréquence des messages, il n'y a jamais plus qu'un message dans la file.

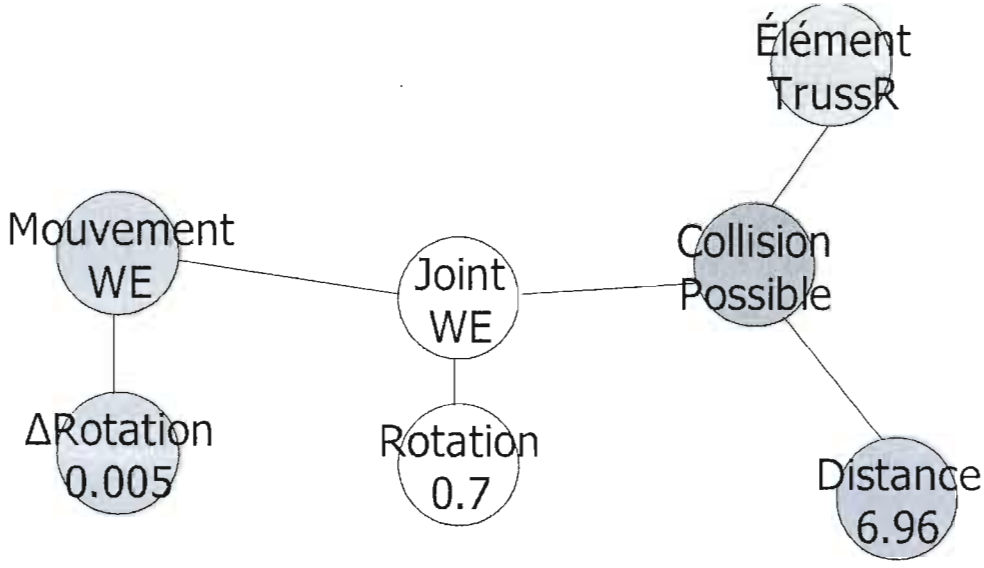

Figure 6.2 Exemple simplifié d'un réseau de micro-processus d'information

De plus, la perception va modifier des micro-processus d'information (si possible), au lieu d'en créer un deuxième. Ceci évite qu'une information dépassée, qui ne reflète plus l'état actuel du micro-monde, persiste sur la scène indéfiniment. La perception doit donc être capable de retrouver les micro-processus qu'elle a créés pour les modifier ensuite.

6.4 Transformation

La transformation elle-même se fait en plusieurs étapes :

1. Comme nous l'avons résumé dans le chapitre 3, une analyse syntaxique crée d'abord un arbre syntaxique.
2. Ensuite, des « *walkers* » traversent cet arbre syntaxique. Quand un « walker » rencontre un nœud qui le concerne, il fait appel au micro-processus de perception spécialiste de l'objet représenté par ce nœeud. Par exemple, un nœud `AJointArmComp` représente un joint et il existe un micro-processus de perception (une instance de la classe `JointCodelet`) spécialisé pour ce joint.
3. Ces micro-processus de perception interprètent les informations contenues dans le nœud pour mettre à jour les micro-processus d'information qui décrivent l'objet dans l'atelier global.

Nous allons maintenant détailler ces trois étapes.

6.4.1 Analyse syntaxique

L'implémentation utilise un outil de génération d'analyseur syntaxique (aussi appelé un compilateur de compilateur) qui génère automatiquement un analyseur syntaxique à partir d'une grammaire formelle. L'outil choisi est SableCC. Cet outil génère du code Java qui transforme le message en un arbre syntaxique typé (avec une classe Java pour chaque type de nœud), ce qui facilite le développement des « walkers ».

Cette étape est donc prise en charge par SableCC, qui utilise les techniques traditionnelles de la compilation : l'analyse lexicale et syntaxique par machines à états finis.

Nous admettons qu'un humain n'utilise pas une machine à états finis pour analyser un texte. Comme le projet CTS se préoccupe principalement de l'utilité d'une architecture d'agent « conscient » *pour un système tutoriel intelligent*, nous avons choisi de concentrer nos efforts sur les aspects nécessaires au tutorat. Et c'est pourquoi cette implémentation comprend une solution non-humaine, mais plus simple, pour l'analyse des messages.

Le résultat de cette étape est un arbre syntaxique typé, comme le montre la figure 6.3.

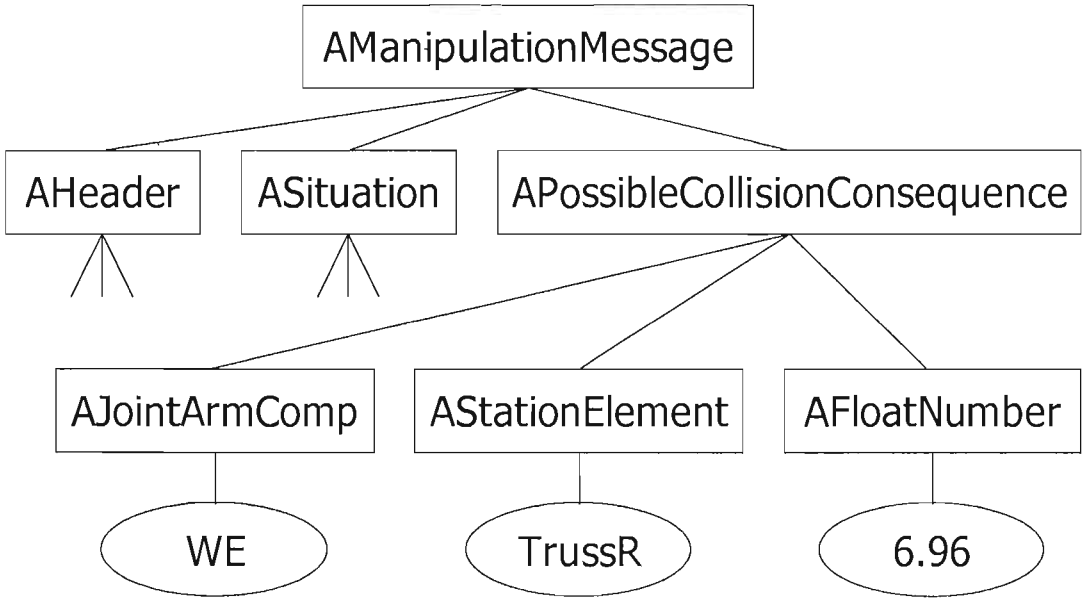

Figure 6.3 Extrait d'un arbre syntaxique

6.4.2 Parcours de l'arbre syntaxique

Cet arbre est ensuite traversé par plusieurs « walkers ». Chacun est capable d'interpréter une partie de l'arbre syntaxique. Quand il rencontre un nœud qu'il peut interpréter, il y extrait de l'information et la fournit au micro-processus de l'objet concerné.

Par exemple, quand le « walker » en charge des collisions traverse l'arbre syntaxique représenté dans la figure 6.3, il rencontre un nœud `APossibleCollisionConsequence`. Il extrait l'information sur les deux objets en collision – le joint WE et l'élément de la station S1P1TrussRight01 (abrégé TrussR dans les figures). Par la suite, il trouve le micro-processus de perception responsable des collisions entre ces deux objets (et il le crée s'il n'existe pas encore) et il lui communique la distance séparant les deux objets.

Ainsi chaque « walker » à sa spécialisation. Présentement, l'implémentation en présente quatre :

- Canadarm – Ce « walker » est responsable de la partie de l'arbre syntaxique qui décrit la configuration du bras canadien, c'est-à-dire la position de chaque composante et l'angle de rotation de chaque joint.
- Screen – Celui-ci se charge des informations par rapport aux écrans, incluant la caméra choisie et pour chacun ses paramètres (pan, tilt & zoom).
- Collision – Il s'occupe des informations par rapport aux collisions et aux risques de collisions.
- Event – Le dernier « walker » traite les méta-données qui décrivent l'évènement, comme l'heure à laquelle l'évènement s'est produit.

6.4.3 Traitement par les micro-processus de perception

Les micro-processus de perception contiennent les règles qui permettent d'interpréter l'information contenue dans le message. L'analyse syntaxique et les parcours de l'arbre résultant ont permis d'extraire l'information du message. On peut maintenant procéder à l'interprétation de l'information.

La tâche élémentaire des micro-processus de perception est de faire le lien vers les micro-processus d'information. Parfois ceci implique de changer la valeur d'un `VariableInformationCodelet`, parfois de créer et défaire des liens entre les micro-processus d'information. Ainsi, quand le micro-processus de perception responsable des collisions entre le joint WE et l'élément de la station S1P1TrussRight01 est informé que la distance entre ces deux objets est 6,96, il va trouver (ou créer) le `FloatInformationCodelet` qui contient cette distance et le mettre à jour.

Au-delà de cette simple correspondance, les micro-processus de perception détectent aussi le changement. À la suite du message de la figure 6.1, le micro-processus de perception pour le joint WE est informé que la rotation de ce joint est de 0.7. Par la suite, il va trouver le micro-processus qui représente cette rotation et vérifier sa valeur. Avec 0.7, rien n'a changé. Au cas contraire, le micro-processus de perception va détecter ce changement et le signaler au micro-processus de perception de mouvement. Celui-ci va créer deux nouveaux micro-processus d'information pour signaler ce changement. Si l'ancien rotation était 0.695, ces deux micro-processus vont être « Mouvement=WE » et « Rotation-Amount=0.005 ». La figure 6.4 illustre ce processus.

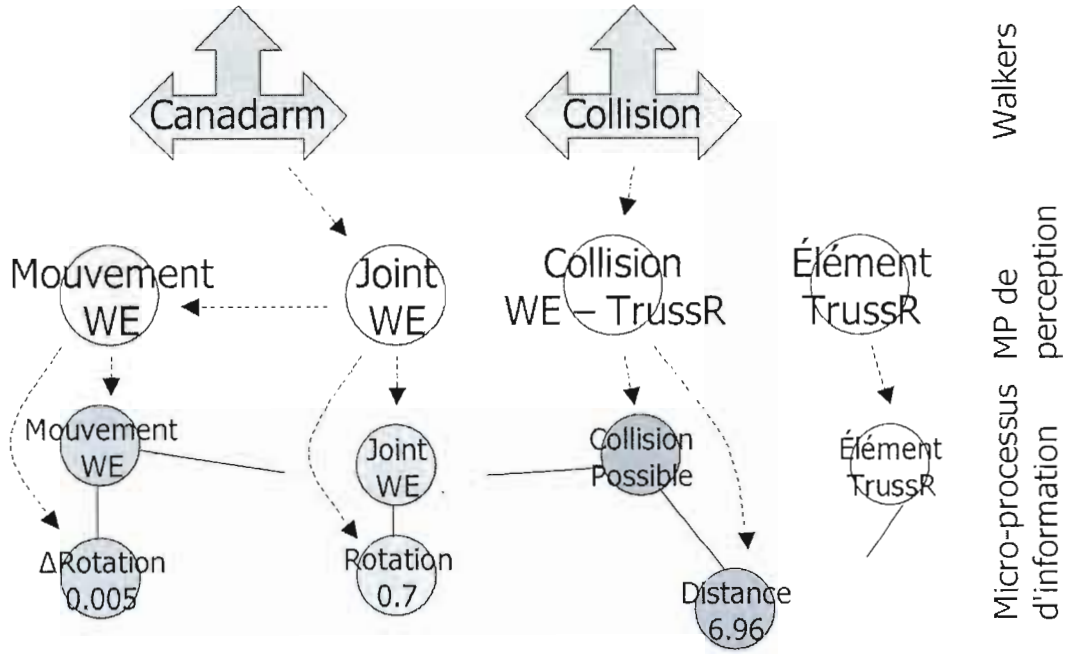

Figure 6.4 Les micro-processus de perception font le lien vers les micro-processus d'information

Finalement, les micro-processus de perception attribuent l'activation (et donc l'importance) aux micro-processus d'information qu'ils manipulent. Comme le comportement de l'agent dépend des informations publiées (qui dépendent de l'activation), cette attribution est une tâche cruciale et délicate.

Pour attribuer l'activation dans la perception, la perception utilise quatre heuristiques :

1. La nature de l'information : pour l'application d'entraînement d'astronautes, une collision est – par sa nature même – plus importante que l'angle de rotation d'un joint. L'implémentation attribue donc un niveau d'activation de base à chaque information, ceci déterminant une partie de son activation.
2. Le variation de l'information : pour cette même application une information qui change est plus importante qu'une information invariante. Ainsi, la perception attribue plus d'activation au micro-processus représentant un joint, quand celui-ci a bougé.

3. L'amplitude du changement : quand il y a un changement, la perception applique un léger ajustement à l'activation en fonction de l'amplitude du changement.
4. La « bienfaisance » de l'information : une amélioration est moins importante qu'un développement vers le pire. Prenons par exemple la possibilité d'une collision qui persiste. Quand la distance entre les objets grandit, ceci est une amélioration de la situation qui ne nécessite pas une intervention immédiate ; la perception lui attribue donc un niveau d'activation bas. Cependant, si la distance a diminué et que le bras robotique se rapproche d'une collision, ceci est un événement qui nécessite une intervention, donc très important.

Les tests effectués montrent que ces règles d'attribution font publier les micro-processus dans l'ordre prévu. Par contre, comme ce mémoire se limite à l'implémentation de l'architecture, des tests futurs sont nécessaires pour évaluer si elles sont adéquates pour un système tutoriel conscient complet.

6.5 Exemple

Juste avant le début de ce cycle, les capteurs ont capté le message de la figure 6.1. Lors de la transformation, celui-ci est transformé d'abord en l'arbre syntaxique de la figure 6.3. Les « walkers » inspectent cet arbre et transmettent l'information aux micro-processus de perception.

Ces micro-processus de perception comparent les nouvelles informations avec celles encore présentes dans les micro-processus d'information. Contrairement à ce qui a été utilisé comme exemple plus haut, le joint WE n'a pas bougé. (Mais c'est bien lui qui est en risque de collision.)

Le micro-processus de perception du joint WE constate donc qu'il n'a pas bougé. Il constate aussi que son micro-processus d'information se trouve encore sur la scène, qu'il attend d'être publié. Vu que rien n'a changé, le micro-processus de perception n'entreprend aucune action. Si le micro-processus d'information avait déjà été publié, il lui aurait donné une activation faible (qui dépend du temps qui s'est écoulé depuis la dernière manipulation du joint) et l'envoyer sur la scène pour qu'il rappelle au système que le joint existe encore. Comme l'activation est faible, il y a peu de chance que ce micro-processus soit publié.

Le micro-processus du joint SE constate par contre qu'il a bougé. Le message indique qu'il est dans la position 0,575, tandis que son micro-processus d'information a conservé la valeur 0,570. Le micro-processus de perception lui redonne donc une activation de 0,4. Ceci est l'activation choisie pour un joint qui présente une nouveauté.

Le micro-processus de perception du joint communique le changement de la rotation au micro-processus de perception qui s'occupe du mouvement de ce joint. Celui-ci crée les deux micro-processus d'information « Mouvement=SE » et son attribut « Rotation-Amount=0,005 ». Pour calculer l'activation à donner à ces deux micro-processus, il commence avec une activation de 0,7, ce qui est la base pour les mouvements. Il ajuste cela par -0.0225 (c'est une diminution), parce que l'amplitude de la rotation est faible. L'ajustement lui-même est fait en utilisant la formule :

```
1 - ( ( 1 - ancienne activation ) * ( 1 - ajustement ) )
```

Cette formule permet « d'ajouter » de l'activation, sans se soucier de dépasser 1. Après l'ajustement, l'activation se trouve à 0,6933.

Finalement, le micro-processus de perception pour les collisions crée les micro-processus « Collision=possible » et son attribut « Distance=6,96 ». Il calcule l'activation en partant de 0,85 (la base pour une collision possible), qu'il ajuste par 0,152 (pour la distance) pour arriver à une activation finale de 0,8728[2].

[2] Au chapitre précédent, nous avons donné la valeur 0,8727 pour l'activation de ce micro-processus. Ceci est dû à la décroissance de l'activation qui a lieu lors de la deuxième étape de chaque cycle cognitif.

CHAPITRE VII

RÉSEAU DES ACTES

7.1 Introduction

Pour qu'un agent soit capable d'actions proactives, il doit avoir des buts et désirs et savoir décider quelles actions lui permettront d'atteindre ces buts et désirs. Il doit être capable de planifier. Dans l'architecture présentée dans ce mémoire, ces besoins de planification et direction sont satisfaits par le réseau des actes[1]. En fournissant ces services, le réseau des actes correspond au directeur de la métaphore de Baars, qui représente les fonctions exécutives du cerveau. Il dirige l'agent afin qu'il atteigne ses buts.

La perception et l'atelier global sont des modules réactifs ; la perception perçoit l'environnement et l'atelier global interprète et raisonne sur le contenu de la mémoire de travail. C'est le réseau des actes qui donne une direction à l'agent, qui lui permet de poursuivre des buts, guidé par les désirs de l'agent.

Alors que l'atelier global est complètement générique et que la perception est complètement spécifique, le réseau des actes possède ces deux caractéristiques à la fois. Nous avons créé un module générique qui contient un moteur de planification, basé sur les travaux de Maes et Negatu et Franklin. Et pour que le module soit complet, il a besoin d'une instanciation qui utilise ce moteur de planification.

Le réseau des actes fournit essentiellement un service de planification. Dans le cadre de notre projet, nous utilisons un réseau des actes pilotant les actions tutorielles.

---

[1] Les architectures STC et IDA utilisent un réseau des actes pour la planification. Cependant, l'aspect « conscient » et le mécanisme de planification sont indépendants. Un agent conscient (doté de la conscience d'accès de l'atelier global) peut utiliser un autre mécanisme de direction et planification. Et un réseau des actes peut être utilisé dans un agent sans conscience d'accès.

Nous présenterons le réseau des actes d'une manière similaire à celle du chapitre 4 ; nous présenterons d'abord les services offerts par la partie générique, puis montrons comment instancier cette partie générique pour une application donnée. Après avoir continué l'exemple des chapitres précédents, nous concluons le présent chapitre avec quelques questions ouvertes.

7.2 Services offertes

Le réseau des actes offre essentiellement deux services : la direction de l'agent par les désirs ; et la planification qui permet de savoir comment atteindre ces désirs.

Nous allons donc présenter ces deux services en premier. Suite à cela, nous allons décrire comment le réseau des actes exécute l'acte choisi et nous terminerons la présentation de deux implémentations de micro-processus d'action fournis par le réseau des actes.

7.2.1 Direction

La planification permet de savoir comment atteindre un but, comment satisfaire un désir. Avant de pouvoir planifier, l'agent a donc besoin de buts et de désirs.

Dans ce réseau des actes, les désirs sont plus fondamentaux que les buts. L'agent va chercher à satisfaire ses désirs pour eux-mêmes, tandis qu'un but n'est considéré que pour son utilité vers la satisfaction d'un désir ou l'atteinte d'un autre but.

L'agent est donc dirigé par ses désirs, et le réseau des actes gère ces désirs.

Nous avons esquissé dans la section 3.5.6 l'idée que le réseau des actes planifie en faisant circuler de l'activation dans le réseau. Les désirs de l'agent sont la source d'activation la plus importante et c'est ainsi que les désirs dirigent la planification et les actions de l'agent.

Comme les désirs de l'agent varient avec l'application, chaque application doit spécifier ses propres désirs. Un désir « veut » que certains états soient atteints. Si ces états ne sont pas atteints, le désir versera, à chaque cycle, le même montant d'activation au réseau. Ceci est le comportement par défaut qui peut être adapté à l'application (cf. section 7.3.2).

7.2.2 Planification

La deuxième tâche du réseau des actes est la planification des séquences d'actes. Nous allons brièvement rappeler ici les mécanismes de la planification, déjà présentés au chapitre 3 :

- Il n'y a pas de représentation explicite des plans. Ils existent implicitement dans le niveau d'activation de chaque acte, qui représente son degré de pertinence dans la situation actuelle vers l'accomplissement des désirs actuels de l'agent.
- L'activation circule selon les règles de Maes (Maes 1989).
- À chaque cycle, l'acte exécutable avec le plus haut niveau d'activation est déclenché.

Le modèle de Maes prévoit aussi quatre paramètres qui permettent d'influencer le comportement du réseau. En modifiant ces paramètres, on peut rendre le réseau :

- plus dirigé par ses désirs, ou plus opportuniste par rapport à l'environnement
- plus minutieux dans sa planification, ou plus prompt à prendre une décision
- plus enclin à poursuivre rigidement un plan engagé, ou à sauter d'un plan à l'autre
- plus porté à protéger rigidement les (sous-) buts atteints et risquer un interblocage, ou indifférent à défaire n'importe quel (sous-) but et risquer de tomber dans un « plan » circulaire.

Malheureusement, quoique Maes indique l'effet d'une modification de chacun des quatre paramètres, elle ne donne pas de recette pour les déterminer sans expérimentation.

7.2.3 Exécution des actes

Une fois un acte sélectionné pour l'exécution, cet acte est simplement accompli en exécutant tous les micro-processus d'action qui le constituent.

7.2.4 Micro-processus de confirmation

Un micro-processus de raisonnement lié au réseau des actes est le micro-processus de confirmation. Ce micro-processus vérifie que les résultats attendus se réalisent. Le module du réseau des actes en fournit une implémentation : `ExpectationCodelet`.

Cette classe implémente un micro-processus de confirmation qui attend la création d'un état donné pendant un certain temps. Cette implémentation est simple et convient à tous nos besoins.

L'état attendu, le temps (en cycle) qu'il attend, ainsi que le nom de l'attente sont spécifiés au constructeur lors de la création du micro-processus. Quand le micro-processus est lancé (en appelant sa méthode `start()`), il attend le nombre de cycle spécifié. Si l'état attendu apparaît pendant ce temps, le micro-processus se « suicidera » (cesse d'exister), sinon il sautera sur la scène avec une activation de .8 pour participer à la compétition en tentant de se faire publier.

Quand il est publié, son type proclame « Failed-Expectation », et sa valeur contient le nom de l'attente. De plus, il est lié à un micro-processus d'information « Expected-State=nom de l'état attendu ».

Par exemple, si le diagnostic ne se termine pas en 20 cycles, une coalition « Failed-Expectation=Diagnosis completed; Expected-State=Cause known » sera publiée.

Un micro-processus de confirmation ne remédie pas à la situation inattendue qu'il constate. Il tente de rendre cette situation « consciente », laissant à l'application le soin de spécifier des actes ou des micro-processus devant réagir à cette situation inattendue.

7.3 Points d'extension

Le réseau des actes est un mécanisme de planification générique qui doit être instancié pour chaque application. C'est-à-dire que l'application doit fournir une liste des états, des désirs et des enchaînements d'actes et de buts, ainsi que des liens entre eux.

Pour faciliter cette tâche, cette implémentation du réseau des actes charge un fichier XML qui fournit justement cette liste. Pour certains éléments du réseau, le fichier XML suffit pour les décrire complètement. Un acte est complètement décrit par la liste des micro-processus d'action qui le composent et ses liens vers les états. Toutes ces informations se retrouvent dans le fichier XML. D'autres éléments du réseau doivent spécifier une classe les implémentant. Par exemple, un micro-processus d'action qui affiche un message.

Afin de faciliter encore plus cette tâche, nous avons développer un éditeur qui permet de créer visuellement le fichier XML décrivant la structure du réseau.

Nous allons présenter cet éditeur et quelques éléments du fichier XML, pour expliquer ensuite comment étendre les classes de base du réseau des actes.

7.3.1 Fichier XML et Éditeur

L'implémentation de Franklin était aussi dotée d'un mécanisme qui charge un fichier XML. Nous nous sommes inspirés de leur format de fichier pour construire le nôtre. Dans le but de préserver un maximum de compatibilité (pour éventuellement être capable d'échanger des scénarios), le nom des éléments que nous avons ajoutés commence avec « x- » pour faciliter leur identification.

De plus, notre fichier XML contient des informations concernant la représentation graphique, qui ne sont utiles qu'à l'éditeur (par exemple les coordonnées des éléments). Ces informations sont regroupées dans des balises `<extension-gdac>`.

Voici l'exemple d'un *état* étant représenté par une balise `<x-state>` :

```
<x-state element-name="Proximity">
  <x-icodelet>
    <x-info-type>Collision</x-info-type>
     <x-info-value>possible</x-info-value>
   </x-icodelet>
</x-state>
```

Les balises `<x-info-type>` et `<x-info-value>` spécifient l'information qui « intéresse » l'état. Quand une publication contient un micro-processus avec cet identificateur, l'état se met à vrai. Le caractère « * », lorsque spécifié pour la valeur (par exemple `<x-info-value>*</x-info-value>`), signifie que l'état se met à vrai quand une publication contient un micro-processus avec le type spécifié, indépendamment de la valeur.

Un autre élément qui sollicite une explication plus détaillée est la balise `<bcodelet>`. Elle représente un micro-processus d'action et doit se trouver à l'intérieur d'une balise d'acte (`<behavior>`). Pour accélérer la tâche du développeur, quatre types de micro-processus pour cette balise sont prédéfinis :

1. <bcodelet
   class -
   name="ca.uqam.dinfo.gdac.cts.behaviornet.bcodelets.Action"
   element-name="Action" type="Behavior Codelet" />

   Cette entrée décrit un micro-processus d'action. Le module va créer une instance de la classe Action, spécifiée par l'attribut « classe-name », et l'exécuter (voir la section 7.3.3) pour accomplir son acte. Cette classe doit être une sous-classe de la classe BCodelet.

   Les trois autres types ne décrivent pas des micro-processus d'action, mais plutôt des micro-processus de raisonnement. Il revient au module qui se charge de créer les micro-processus de lancement correspondant.

2. <bcodelet element-name="Expect Need Diagnosis"
   type="Expectation Codelet" wait-cycles="20" >
    <addition element-name="Need Diagnosis" proxy-id="d49a53"/>
   </bcodelet>

   Ceci déclare un micro-processus de confirmation, comme l'attribut « type » indique. Quand le module rencontre une telle déclaration, il va automatiquement créer un micro-processus de confirmation et un micro-processus de lancement pour lancer ce premier micro-processus. Lors de l'exécution de l'acte, le micro-processus de lancement va lancer le micro-processus de confirmation. Par la suite, ce dernier va attendre 20 cycles que l'état « Need Diagnosis » s'avère.

   L'implémentation abuse légèrement de la balise addition pour les micro-processus de lancement : au lieu d'indiquer un état que le micro-processus devrait créer, la balise indique un état dont le micro-processus de confirmation surveille la création.

   Il est possible d'avoir plusieurs balises « addition » à l'intérieur de la balise d'un micro-processus de confirmation. En ce cas, le micro-processus de confirmation les surveille tous.

3. <bcodelet
   class-
   name="ca.uqam.dinfo.gdac.cts.bnet.bcodelets.BadPlaceCause"
   element-name="Bad Place Cause" type="Reasoning Codelet"/>

   Voici la déclaration d'un micro-processus de raisonnement générique. La classe indiquée doit être une sous-classe de `Codelet`. Le module du réseau va créer un micro-processus de lancement pour démarrer ce micro-processus lors de l'exécution de l'acte l'englobant.

4. <bcodelet element-name="ThirdStepCodelet"
   type="Information Codelet"
   x-info-type="Counted" x-info-value="3" />

   Le dernier type pour cette balise spécifie de lancer un micro-processus d'information avec le type et la valeur spécifiés par les attributs.

   Ce type est moins utile que les autres, car il n'est pas possible de spécifier des liens pour ce micro-processus d'information. Ce sera donc un micro-processus d'information caractérisé uniquement par son identificateur et son activation. Quand une application a besoin de créer un micro-processus d'information qui possède des liens vers d'autres micro-processus, elle doit utiliser le premier type (qui crée un micro-processus d'action) et créer les liens dans le code de la classe qui implémente le micro-processus d'action.

Tous les paramètres du fichier XML sont modifiables à partir de l'éditeur. Cet éditeur a été réalisé dans le cadre d'application Eclipse et GEF (Graphical Editing Framework).

La figure 7.1 montre une capture d'écran de cet éditeur. L'espace principal permet d'assembler la structure d'un réseau des actes par « glisser-déposer » (terme du grand dictionnaire terminologique pour « drag-and-drop »). Les paramètres plus détaillés de chaque objet sont fournis dans la partie « Properties » en-dessous. Dans la figure 7.1, cette zone montre les paramètres « Information Codelet Type » (x-info-type) et « Information Codelet Value » (x-info-value) pour l'état « Proximity ».

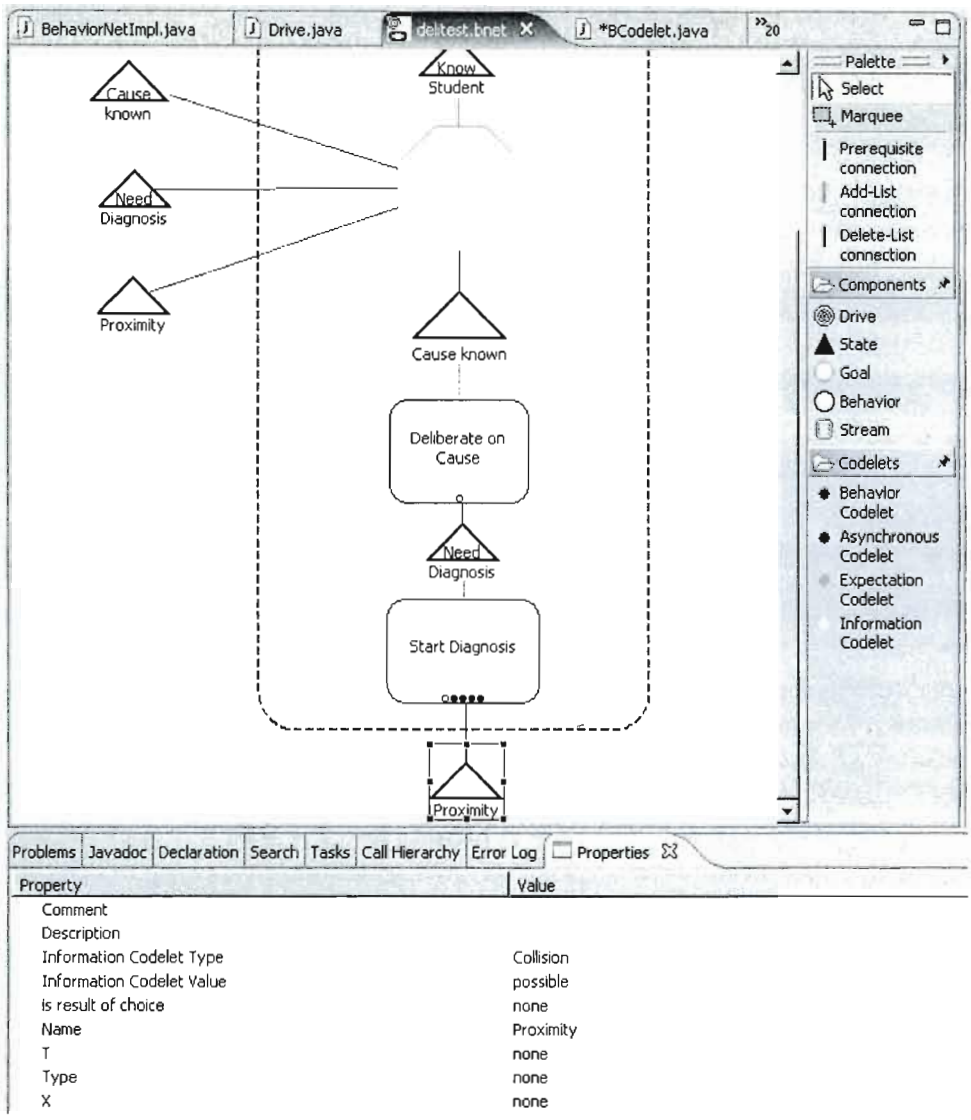

Figure 7.1 Capture d'écran de l'éditeur

Nous allons poursuivre ce chapitre en expliquant comment une application peut étendre les classes de ce module qui sont prévues à cet effet.

7.3.2 Désir

Par défaut, un objet de la classe `Drive`[2] verse une activation constante dans le réseau des actes.

Il est possible de modifier ce comportement en spécifiant une classe l'implémentant dans le fichier XML qui étend la classe `Drive`.

À chaque cycle, les états qui sont liés au désir vont appeler la méthode `getActivation()` pour déterminer l'activation versée par le désir. C'est cette méthode qu'une sous-classe va vouloir redéfinir.

Parce que cette sous-classe va être instanciée à l'aide de l'API de réflexion, elle doit avoir un constructeur qui prend exactement un argument : une chaîne de caractère indiquant le nom du désir.

7.3.3 Micro-processus d'action

La classe `BCodelet` est la classe de base des micro-processus d'action. Cette classe est beaucoup plus simple que la classe de base des micro-processus de raisonnement.

Et au lieu d'avoir plusieurs méthodes à redéfinir, il n'y en a qu'une seule : `execute()`. C'est cette méthode qui est appelée quand le micro-processus d'action est exécuté pour accomplir son acte.

Afin d'éviter un blocage de l'agent, tout implémentation de cette méthode doit se terminer rapidement. Quand le micro-processus d'action a besoin d'effectuer une action qui prend un temps non-négligeable, le développeur est encouragé à lancer un nouveau processus léger qui accomplit l'action sans paralyser le cycle cognitif.

Finalement, comme cette classe est aussi instanciée à l'aide de l'API de réflexion, elle doit avoir un constructeur qui prend exactement un argument : une chaîne de caractère indiquant le nom du micro-processus d'action.

---

[2] Nous maintenons l'appellation « drive » de Franklin en anglais pour conserver un vocabulaire commun.

7.3.4 Micro-processus de raisonnement

Lorsque la classe d'un micro-processus de raisonnement est spécifiée dans le fichier XML, le module va automatiquement l'instancier au besoin. Pour que cela fonctionne, cette classe aussi doit avoir un constructeur qui prend exactement un argument : une chaîne de caractères indiquant le nom du micro-processus de raisonnement.

7.3.5 Exemple d'un réseau des actes

Voici (figure 7.2) l'exemple d'un réseau des actes très simple que nous avons utilisé pour nos tests :

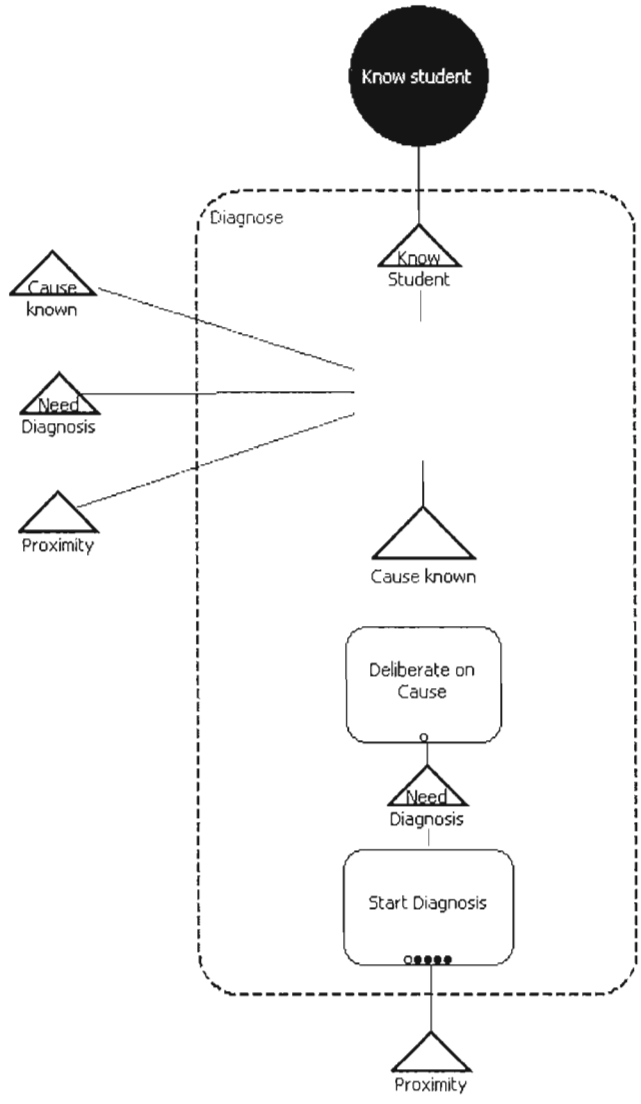

Figure 7.2 Exemple d'un réseau des actes

Ce réseau ne contient qu'un désir (connaître l'étudiant) et qu'un enchaînement (diagnose). L'enchaînement contient un but et deux actes : le premier lance le diagnostic et, vu que le diagnostic est une action prolongée, le deuxième attend son résultat.

7.4 Exemple d'exécution

Nous reprenons notre exemple au moment où la coalition {« Collision=possible », « Station-Element=S1P1TrussRight01 », « Joint=WE » et « Distance=6,95931»} est publiée.

Les trois éléments du réseau des actes ont les valeurs suivantes : le but (« Know Cause ») a une activation de 30,4, l'acte au milieu (« Deliberate on Cause ») une activation de 18,4 et le premier acte (« Start Diagnosis ») 11,2.

Pendant la circulation de l'activation, « Start Diagnosis » reçoit de l'activation aussi bien du prochain acte que de sa précondition (l'état « Proximity »). Ainsi, après la circulation de l'activation, celle de « Start Diagnosis » a augmenté à 16,7.

Comme « Start Diagnosis » est le seul acte exécutable, et que son niveau d'activation est supérieur au seuil minimal de 12,6 (il n'y a pas eu d'acte choisi pendant plusieurs cycles), « Start Diagnosis » est choisi pour être exécuté.

Il lance un micro-processus de confirmation, un micro-processus d'arbitrage et trois autres micro-processus de raisonnement. Ces trois micro-processus vont délibérer, dans les cycles subséquents sous la direction du micro-processus d'arbitrage, sur la cause de ce rapprochement.

7.5 Questions ouvertes

Le réseau des actes est le module le plus complexe de ceux présentés dans ce mémoire et plusieurs questions ne sont pas encore résolues complètement :

- Est-ce que le réseau de Franklin *planifie* où est-ce qu'il *choisit* un plan partiel préexistant?

  Le réseau de Maes est conçu pour trouver une séquence d'actes, accomplissant un but dans un réseau connexe où tout acte est envisageable. Avec l'introduction des enchaînements (qui sont des plans partiels), la tâche du réseau ressemble plus à un choix parmi des plans préexistants qu'à une vrai planification.

  Un mécanisme plus simple sera possible si le seul but du réseau est de choisir entre des plans préétablis.

- Est-ce que l'instanciation est utile?

Franklin instancie uniquement la partie du réseau qui est présentement utile. Par contre, si le réseau est vraiment un graphe fortement connexe, tout le réseau est potentiellement utile, parce que chaque nœud est lié (au moins indirectement) à un nœud utile. Et si le réseau n'est pas connexe, cela revient à la première question : est-ce que Franklin utilise le réseau pour planifier ou pour choisir entre des plans?

C'est pourquoi le réseau des actes n'utilise pas d'instanciation présentement.

- Comment distinguer les variables?

  Lorsque deux collisions se produisent à court intervalle, le système tutoriel évite présentement d'avoir deux diagnostics simultanés. Il continue le premier diagnostic entrepris. Il sera intéressant de pouvoir interrompre un diagnostic pour recommencer un diagnostic plus urgent.

- Quand et comment défaire des états?

  Présentement, quand un acte indique que son exécution défait un état, le réseau le défait automatiquement. Il sera préférable d'avoir un mécanisme plus général pour cette tâche. Malheureusement, il est facile de déduire qu'une situation existe à partir d'une publication, mais beaucoup plus difficile de prouver qu'une situation n'existe plus.

CONCLUSION

Dans ce mémoire, nous avons décrit l'architecture d'un agent « conscient » que nous avons adoptée pour supporter un système tutoriel intelligent. Cette architecture permet de filtrer un grand nombre d'informations afin de focaliser le système sur les informations les plus importantes à un instant donné. Cette capacité de filtrer l'information est réalisée grâce à une théorie sur la conscience humaine.

La conscience d'accès permet aussi à différents micro-processus de collaborer à la résolution d'un problème ; ainsi, la conscience est similaire à un tableau noir, avec la différence principale que le contrôle dans le modèle du tableau noir s'exerce sur les sources de connaissance, tandis que la conscience exerce un contrôle sur l'information.

Nous constatons déjà un bénéfice de cette caractéristique : afin de profiter des travaux de recherche préexistant dans le domaine des STI, nous envisageons d'intégrer un modèle-apprenant classique dans l'architecture. Vu que le contrôle est sur l'information, nous n'éprouvons aucune difficulté à intégrer ce module étranger et non-prévu.

Au delà de rendre possible une collaboration, la conscience permet aussi d'entreprendre *plusieurs* de ces collaborations en même temps. En ne publiant qu'**une** coalition (provenant d'une seule collaboration) la conscience « alloue » effectivement les ressources de l'ordinateur à celle qui est la plus importante dans une situation donnée. Ainsi l'architecture permet à l'application d'effectuer des activités optionnelles « s'il n'y a rien de plus urgent à faire » et de les interrompre automatiquement pour des activités d'une priorité plus haute. Dans notre exemple de tuteur, cette capacité lui permet de réévaluer et d'améliorer le modèle de l'apprenant pendant que ce dernier réfléchit.

De plus, cette architecture fournit un module de planification opportuniste (le réseau des actes) qui dirige l'agent et lui permet d'être proactif, toute en restant réactif.

Nous avons donc créé une base pour la suite du projet STC qui consiste à créer un tuteur qui s'appuie sur cette architecture. Cette base contribuera aux travaux actuels dans le domaine

des STI sur les architectures cognitives capables de soutenir le fonctionnement d'un tuteur et repousse ainsi les limites des résultats actuellement obtenus dans ce domaine. Nos résultats ont été publiés à plusieurs forums importants (Dubois, Nkambou et Hohmeyer, 2006a, 2006b & 2006c).

La principale limite de cette architecture est le manque d'une méthode établie pour créer des applications : l'architecture comporte un très grand nombre de paramètres (chaque micro-processus décidant de son activation lui-même, cela amène un (ou même plusieurs) paramètre(s) à spécifier par micro-processus) qui doivent être déterminés par des tests empiriques. Aussi, une partie considérable de l'intelligence du système *émerge* de la coopération entre les micro-processus, la conscience et le réseau des actes. La remarque de Wooldridge sur l'émergence (quoiqu'écrit par rapport aux agents purement réactifs) s'applique aussi à l'architecture proposée dans ce mémoire :« *But the very term "emerges" suggests that the relationship between individual behaviours, environment, and overall behaviour is not understandable. This necessarily makes it very hard to* engineer *agents to fulfill specific tasks. Ultimately, there is no principled* methodology *for building such agents: one must use a laborious process of experimentation, trial, and error to engineer an agent.* » (Wooldridge 1999).

Une autre limite est le fait que le cycle cognitif implémenté a été testé à travers des processus peu complexes (« jouets »). Un des enjeux de la poursuite du projet STC est d'observer le déroulement de ce cycle dans un contexte réel de formation où l'on déploie des processus complexes et complets liés à l'apprentissage, implémentant le « coaching », un diagnostic sophistiqué, etc.

Ainsi, notre équipe développera un tuteur basé sur l'architecture présentée dans ce mémoire. Les expériences accumulées permettront de répondre aux questions du projet initial :

- Est-il faisable et utile de bâtir un système tutoriel intelligent sur une architecture d'agent conscient?
- Est-ce que l'architecture choisie est assez flexible et puissante pour supporter un système aussi complexe?
- Est-ce qu'un système tutoriel « conscient » démontre un comportement ressemblant au comportement d'un tuteur humain?

Un nouvel enjeu du projet sera de mettre à contribution l'expertise acquise afin de réduire le nombre de paramètres dans l'architecture et de fournir des guides pour les futurs chercheurs qui travailleront sur la suite de ce projet.

GLOSSAIRE

Acte. Un ensemble micro-processus d'action. Unité atomique dans la planification (l'agent exécute tous les micro-processus d'action de l'acte ou aucun).

Apprenant. L'utilisateur (dans notre cas un astronaute) que le système forme.

Coalition. Ensemble de micro-processus qui décrivent un concept. Participant à la *compétition.*

Compétition. Comparaison des forces des coalitions. La coalition la plus forte est *publiée.*

Conscience (d'accès). Elle contrôle l'accès aux informations sur la scène. Implémentée par la *publication.*

Cycle cognitif. Une séquence d'étapes dans un agent cognitif qui est répétée perpétuellement.

Étudiant. Un autre mot pour « *apprenant* ».

IDA. Intelligent Distribution Agent. L'agent et l'architecture développés par Stan Franklin

Micro-monde. Un monde virtuel qui permet à un apprenant d'expérimenter

Micro-processus. L'implémentation d'un processus simple de l'esprit. L'élément de base des architectures IDA et STC.

Micro-processus d'action. Une action élémentaire.

Publication. Implémentation de la conscience d'accès. Le contenu de la publication, fourni à tout le système, devient « conscient ».

Réseau des actes. Le mécanisme de direction et planification de l'architecture d'agent conscient. Basé sur les travaux de Maes.

Scène. La mémoire de travail dans la métaphore de Baars. Antichambre de la *conscience.*

Simulateur. Un modèle virtuel de la station spatiale qui crée un micro-monde pour l'apprenant.

STC. Système tutoriel conscient. Notre agent et architecture.

Système tutoriel intelligent. Système informatique qui forme un humain en s'adaptant à chaque apprenant individuellement.

Tuteur. A) Le système tutoriel intelligent au complet ;
B) Le module du système tutoriel intelligent qui prend les décisions pédagogiques.

RÉFÉRENCES

Aleven, Vincent, Bruce M. McLaren, Jonathan Sewall et Kenneth R. Koedinger. 2006. « The Cognitive Tutor Authoring Tools (CTAT): Preliminary evaluation of efficiency gains ». *Proceedings of the 8th International Conference on Intelligent Tutoring Systems (ITS 2006)*, Springer Verlag, Berlin, pp. 61-70

Anderson, John R. et al. 2004. « An Integrated Theory of the Mind ». *Psychological Review*, vol. 111, no 4, pp. 1036-1060.

Anwar, Ashraf et Stan Franklin. 2003. « Sparse distributed memory for 'conscious' software agents ». *Cognitive Systems Research*, vol. 4., no 4, pp. 339-354.

Baars, Bernard J. 1997. « In the theater of consciousness ». *Journal of Consciousness Studies*, vol. 4, no 4, pp. 292-309.

Beck, Joseph, Mia Stern et Erik Haugsjaa. 1996. « Applications of AI in Education ». *Artificial Intelligence, ACM Crossroads*, no 3.1 (automne).

Bratman, Michael E., David J. Isreal et Martha E. Pollack. 1988. « Plans and resource-bounded practical reasoning ». *Computational Intelligence*, vol. 4, no 4.

Broersen, Jan, Mehdi Dastani, Joris Hulstijn, Zisheng Huang et Leendert van der Torre. 2001. « The BOID architecture: Conflicts between beliefs, obligations, intentions and desires ». *Proceedings of he Fifth International Conference on Autonomous Agents (AA2001)*, ACM Press, pp. 9-16.

Brooks, Rodney A. 1986. « A robust layered control system for a mobile robot ». *IEEE Journal of Robotics and Automation*, vol. 2, no 1, pp. 14-23.

Carver, Norman et Victor Lesser. 1992. « The Evolution of Blackboard Control Architectures », *Expert Systems with Applications – Special Issue on the Blackboard Paradigm and Its Application* , vol. 7, no 1, Pergamon Press, pp. 1-30.

Dubois, Daniel, Roger Nkambou et Patrick Hohmeyer. 2006. « How 'Consciousness' Allows a Cognitive Tutoring Agent Make Good Diagnosis During Astronauts' Training ». *Proceedings of the 8th International Conference on Intelligent Tutoring Systems (ITS'2006)*, *LNCS*, no 4053, Springer-Verlag, Berlin, pp. 154-163.

Dubois, Daniel, Roger Nkambou et Patrick Hohmeyer. 2006. « The Role of «Consciousness» for the Diagnosis Process in a Tutoring Agent ». *Proceedings of the Florida Artificial Intelligence Research Society (FLAIRS'2006)*, AAAI press, pp. 539-540.

Dubois, Daniel, Roger Nkambou et Patrick Hohmeyer. 2006. « Supporting Simulation-Based Training Using a «Conscious» Tutoring Agent ». *Proceedings of the IEEE-ICALT'2006*, IEEE press, pp. 936-938.

Franklin, Stan et F. G. Patterson, Jr. 2006. « The LIDA Architecture: adding new modes of learning to an intelligent, autonomous, software agent ». *Integrated Design and Process Technology*, IDPT-2006. À apparaître en juin 2006.

Hofstadter, Douglas R. 1984. « The Copycat project: An experiment in nondeterminism and creative analogies ». *AI Memo*, no 755. Artificial Intelligence Laboratory, MIT.

Johnson, Lewis W. 2001. « Pedagogical Agent Research at CARTE ». *AI Magazin*, vol. 22, no 4, pp. 85-94. Hiver.

Kabanza, Froduald, Roger Nkambou, Khaled Belghith et Leo Hartman. 2005. « Path-Planning for Autonomous Training on Robot Manipulators in Space ». *Proceedings of IJCAI'2005 (International Joint Conferences on Artificial Intelligence)*, pp. 1729-1732.

Kanerva, Pentti. 1988. « Sparse Distributed Memory ». The MIT Press.

Maes, Petty. 1989 « How To Do the Right Thing ». *Connection Science Journal, Special Issue on Hybrid Systems*, vol. 1.

Negatu, Aregahegn S et Stan Franklin. 2002. « An action selection mechanism for "conscious" software agents ». *Cognitive Science Quarterly*, vol. 2, pp. 363-386.

Nkambou, Roger, Khaled Belghith et Froduald Kabanza. 2006. « An Approach to Intelligent Training on a Robotic Simulator using an Innovative Path-Planner ». *Proceedings of the 8th International Conference on Intelligent Tutoring Systems (ITS'2006)*, *LNCS*, no 4053, Springer-Verlag, Berlin, pp. 645-654.

Frasson, Claude, Thierry Mengelle et Esma Aimeur. 1997. « Using pedagogical agents in a multi-strategic intelligent tutoring system ». *Proceedings of the AI-ED '97 Workshop on Pedagogical Agents*, pp. 40-47.

Sabah, Gérard. 1997. « Consciousness: a requirement for understanding natural language ». in S. O. Nuallàin, P.M.K., E.M. Aogàin ed. *Advances in Consciousness research*, John Benjamins, Amsterdam, pp. 361-392.

Wooldridge, Michael. 1999. « Intelligent Agents ». Dans Weiss, G., editor: *Multiagent Systems*, The MIT Press, Avril 1999.