UNIVERSITÉ DU QUÉBEC À MONTRÉAL

# ELABORATION, IMPLÉMENTATION ET INTÉGRATION D'UN MODULE DE GESTION DU DIALOGUE TUTORIEL EN LANGAGE NATUREL DANS LE CADRE D'UN AGENT COGNITIF

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR

JEAN-FRANCOIS QUINTAL

JANVIER 2011

# REMERCIEMENTS

J'aimerais remercier ici toutes les personnes qui m'ont apporté leur aide et qui ont contribué à l'élaboration de ce mémoire.

Je remercie d'abord Roger Nkambou, directeur de ce mémoire, de m'avoir accepté comme étudiant malgré mes contraintes de temps, ainsi que pour l'aide et le temps qu'il m'a consacré et sans qui ce mémoire n'aurait jamais existé.

Je remercie aussi l'équipe du GDAC pour leurs travaux préalables sur le système CTS qui ont fourni une base de travail à la recherche que ce mémoire soutient. Daniel Dubois reçoit plus particulièrement mes remerciements pour nos discussions et les explications sur l'architecture de CTS et ses fondements théoriques, ainsi que pour m'avoir remis en contexte lorsque mes solutions divergeaient de ces fondements.

Finalement, je tiens à remercier Josée Dauphinais, mon épouse, pour sa compréhension et son soutien moral dans mes démarches académiques, sans quoi le présent mémoire n'aurait jamais vu le jour.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

# RÉSUMÉ

Les systèmes tutoriels intelligents (STI) sont un grand pas vers une réforme dans l'éducation. Ces systèmes offrent une souplesse d'enseignement que les autres aides pédagogiques informatiques n'ont pas. De ce fait, ils pourraient, s'ils sont bien intégrés dans un programme éducatif, décharger les professeurs pour qu'ils consacrent une attention particulière aux étudiants plus faibles.

Les systèmes tutoriels intelligents atteignent cette souplesse grâce à la combinaison de sous-systèmes; l'un d'entre eux est la communication. Plusieurs recherches ont été effectuées dans ce sens notamment pour la communication en langage naturel. Cette communication peut être divisée en trois parties soit la compréhension du langage naturel, la génération de texte en langage naturel et la planification des dialogues. Cette dernière représente la base de ce type de communication.

CTS (Cognitive Tutoring System) est un moteur de système tutoriel intelligent basé sur la conscience d'accès développée par le GDAC. CTS a été intégré à Canadarm Tutor pour son développement. Ce mémoire traite de l'ajout d'un système de planification du dialogue basé sur les travaux effectués sur Beetle.

Dans un premier temps, plusieurs correctifs seront apportés au fonctionnement du Réseau des Actes pour tenter de stabiliser son comportement; d'autres amèneront le système plus près de ses fondements notamment l'ajout de la délibération.

L'ajout du planificateur tel que décrit dans le STI de Beetle s'effectuera dans un second temps et utilisera l'architecture unique de CTS pour le faire. Cette combinaison d'architecture apportera plusieurs avantages et donnera un système de planification de dialogue générique et augmentable.

# CHAPITRE I

## INTRODUCTION

Les systèmes tutoriels intelligents représentent l'avenir dans le domaine de l'enseignement. Dans le système éducatif actuel où les classes sont de plus en plus peuplées, les enseignants doivent toujours faire un choix entre l'aide qu'ils peuvent donner aux étudiants plus faibles et l'enseignement de nouvelles matières aux autres étudiants. Ce choix a pour effet que souvent les étudiants les plus faibles ne reçoivent pas l'aide nécessaire à leur compréhension de la matière, nuisant ou même bloquant la progression de leur apprentissage sur la nouvelle matière. Aussi, les étudiants ayant plus de facilité à apprendre se retrouvent souvent ralentis dans leur apprentissage. La conséquence est une possible démotivation pour les étudiants se retrouvant aux deux extrêmes du spectre : les étudiants très faibles se découragent de leur médiocrité scolaire et éventuellement abandonnent leurs cours. Les étudiants plus forts deviennent, pour leur part, paresseux dans leur apprentissage à force de toujours attendre après les autres et peuvent développer une faiblesse face aux difficultés rencontrées sur certains sujets souvent occasionné par la perte de la capacité de faire un apprentissage actif (un exemple serait une déficience dans la méthodologie d'étude.)

Dans une telle situation, on peut facilement voir le principal avantage que les systèmes tutoriels intelligents apporteraient aux enseignants et aux étudiants. Les étudiants travaillant sur ces systèmes avanceraient à leur rythme, le système donnant la matière au fur et à mesure que l'étudiant l'assimile et la maîtrise, tout en libérant les enseignants pour aider les étudiants plus faibles, que le système n'arrive pas à aider avec les théories pédagogiques à sa disposition ou parce qu'il a épuisé sa banque d'explications pour un sujet donné. Les enseignants n'ont plus ainsi à faire un compromis entre le rythme d'enseignant pour la

nouvelle matière et l'aide à apporter aux étudiants plus faibles, puisque le rythme d'enseignement est déterminé par la capacité d'apprentissage de l'étudiant.

Pour pouvoir arriver à cette fin, un système tutoriel intelligent doit posséder plusieurs parties, soit un modèle expert du domaine qu'il enseigne, un modèle de l'apprenant, qui représente à la fois le profil de l'étudiant (préférences, états émotionnels, etc.) et son état d'apprentissage (ce qu'il sait ou ne sait pas et son niveau de maîtrise des différents sujets.) Le système doit aussi être capable de décider ce qui doit être montré et comment ce sera montré. Pour cela, il doit posséder la capacité de comprendre ce que l'utilisateur dit et fait ainsi que la capacité de se faire comprendre par l'utilisateur, soit par communication textuelle/vocale soit par action visuelle de son interface (affichage d'images, déplacement de la souris/pointeur, exécution d'une séquence (vidéo ou autre), etc.) Sous-jacent à cette capacité d'interaction, le système doit être capable de planifier les interactions en question.

## 1.1 Contexte de l'étude

Mon mémoire prend comme point de départ l'un des systèmes tutoriels développés au laboratoire GDAC, soit celui visant l'entraînement des astronautes à la manipulation du bras robotique Canadarm2.

Ce système tutoriel est composé de deux parties. La première partie est un simulateur du bras canadien situé sur la station spatiale internationale. Ce simulateur représente graphiquement l'environnement de contrôle du bras canadien. Entre autres, on retrouve à l'écran les trois moniteurs dont l'astronaute dispose pour observer le bras canadien et son environnement. Ce simulateur permet à l'étudiant de manipuler les vues sur les moniteurs en choisissant les caméras à utiliser ainsi que leurs angles de vue et le facteur de rapprochement. Il permet aussi de manipuler le bras simulé en sélectionnant un joint et en modifiant son angle. Certaines fonctionnalités de tutorat ont été intégrées au simulateur, entre autres un planificateur de trajet qui lui permet de trouver la meilleure solution pour déplacer un objet du point A au point B.

La deuxième partie du système est un agent cognitif, lui aussi, développé au GDAC qui se nomme CTS (Conscious Tutoring System.) Cet agent cognitif, basé sur l'agent IDA de Franklin, possède une conscience d'accès aux informations. Cette propriété permet à l'agent de travailler dans un environnement contenant beaucoup d'informations sans en être submergé. Le système arrive à choisir les informations importantes à traiter dans chaque cycle cognitif. CTS constitue le cerveau du système tutoriel intelligent développé par le GDAC. Les modèles/experts du domaine et de l'apprenant sont contenus ici ainsi que la capacité du système à décider de l'action à exécuter en réponse à l'état de l'environnement.

## 1.2 Problématique

Un côté important du tutorat est la communication entre le tuteur et l'étudiant. Aussi, le rôle du tuteur est d'amener l'étudiant à comprendre ce qui lui est montré. Pour arriver à cette fin, le tuteur doit être capable de :

1. Exposer et expliquer les concepts du domaine.
2. Superviser les démarches de résolution de problème par l'étudiant
3. Répondre aux questions de l'étudiant.
4. Questionner l'étudiant pour vérifier l'état des connaissances transmises.

CTS, bien qu'il soit un système capable de prendre des décisions d'action selon le contexte, n'a pas la capacité de planifier un discours ou de suivre une conversation à long terme. Le réseau d'actes sur lequel reposent les décisions de CTS est trop rigide pour la planification de dialogue. Cette rigidité du système sera décrite plus tard.

En supposant présentes les connaissances du domaine, plusieurs éléments supplémentaires sont nécessaires pour donner ces capacités de communications à CTS. Entre autres, il lui manque un système de compréhension du langage naturel et un système de génération de texte en langage naturel pour faciliter la communication entre le système et l'étudiant. Ces deux systèmes, bien qu'ils aident à faciliter la compréhension bidirectionnelle,

ne donnent pas au STI les capacités nécessaires décrites plus haut. Il faut donc aussi doter le STI d'un planificateur de dialogue qui lui permettra de planifier et de suivre les communications. Le planificateur permettra au système de choisir parmi plusieurs méthodes d'enseignement, la méthode la mieux adaptée pour la situation en cours, que ce soit par un discours constructiviste, par des explications magistrales, par un indice lors d'un exercice, etc. Le planificateur permettra aussi de suivre la conversation et de savoir à qui est le tour de parler.

## 1.3 Objectif

L'objectif de ce travail est de doter l'agent tutoriel CTS d'un planificateur qui lui donnera les capacités exposées ci-haut. Ce planificateur devra entre autres :

1. Respecter et utiliser l'architecture CTS;
2. Choisir parmi plusieurs méthodes de communication disponibles la méthode la plus appropriée au contexte;
3. Pouvoir changer de plan lorsque le plan actuel est inefficace ou se retrouve inadapté au nouveau contexte;
4. Effectuer des modifications ponctuelles au plan en cours sans perdre le fil de la discussion (exemple, durant une explication, l'étudiant pose une question, CTS devra pouvoir répondre à la question puis continuer son explication);
5. Utiliser différents modes de communication pour faire passer le message. (Exemple : Utiliser du texte en langage naturel, des images, des actions de l'interface utilisateur);

Pour atteindre ces objectifs, d'autres objectifs secondaires, mais tout aussi importants devront être remplis, soit :

1. Ajouter certains modules de CTS qui sont actuellement absents;

2. Améliorer certaines simulations de module pour avoir un environnement plus riche;

3. Corriger quelques problèmes de CTS qui l'empêchent d'avoir un fonctionnement stable à long terme;

Finalement, deux autres modules seront ajoutés soit un module de génération de langage naturel et un module de compréhension du langage naturel pour permettre les communications avec le système. Ces modules seront, bien entendu, simples en fonctionnement puisque le présent mémoire porte sur la planification du langage naturel et non sur la génération ou la compréhension du langage naturel.

## 1.4 Méthodologie

Tout d'abord, une architecture de planification devait être choisie. Une comparaison a été faite entre différentes architectures existantes dans différents systèmes tutoriels. L'architecture répondant le plus aux objectifs fixés est l'architecture en trois tiers proposée par Zinn et al. 2002 et utilisée dans le STI Beetle.

L'étape suivante était d'apprendre et de comprendre le fonctionnement interne de CTS et de décider comment adapter l'architecture de planification en 3 tiers à CTS. Lors de l'apprentissage du fonctionnement interne de CTS, plusieurs faiblesses du système ont été trouvées ainsi que quelques disparités entre l'implémentation et la vision du système. Il a alors fallu réparer les faiblesses et régler les disparités qui interféraient avec l'implémentation du planificateur. Ces réparations ont été effectuées en tentant de minimiser les modifications au code source original. Aussi, nous avons constaté que certains modules nécessaires pour le fonctionnement du planificateur (sans faire en partie) sont manquants ou très sommairement simulés. Les simulations de ces modules ont dû alors être refaites ou créées pour au moins tester le planificateur.

Ensuite, le planificateur a été implémenté dans le système suivant l'architecture des trois tiers. Cette implémentation s'est faite graduellement ajoutant une à une les caractéristiques voulues. C'est à cette étape que nous avons implémenté les modules de compréhension et de génération du langage naturel.

Des vérifications de fonctionnement ont été effectuées tout au long du développement pour s'assurer de la validité des traitements. Ces vérifications ont été faites à l'aide de tests tout au long du développement des différentes parties. Aussi, un scénario a été élaboré pour pouvoir tester la bonne collaboration entre les différents modules et sous-systèmes.

## 1.5 Organisation du mémoire

Le premier chapitre est la présente introduction. Les deux chapitres suivants du mémoire donneront une description des systèmes, soit, dans le deuxième chapitre, un état de l'art sommaire sur les différentes architectures de planificateurs de communication en langage naturel, et le troisième chapitre présentera le système de CTS. Au chapitre suivant, j'exposerai les raisons du choix du planificateur à trois tiers.

Dans le cinquième chapitre, il sera question des modifications apportées au fonctionnement général de CTS ainsi que les raisons de ces modifications. Au sixième chapitre, il sera question des différents modules liés principalement au planificateur et le septième chapitre contiendra la description des scénarios utilisés pour les tests. Une courte conclusion résumera les contributions faites à CTS par le présent travail et une liste de travaux futurs qui devront être considérés. À la fin du document, deux annexes seront disponibles, soit le modèle du domaine utilisé situé dans l'annexe A et le Réseau des Actes utilisé situé dans l'annexe B.

# CHAPITRE II

# LES PLANIFICATEURS DE DIALOGUE DANS LES STI

La planification de dialogue dans le cadre d'échange en langage naturel est très importante. D'une certaine manière, elle est même plus importante que les tâches de compréhension et de génération de langage naturel. En effet, cette partie dicte à un système quand et comment une intervention sera effectuée, quelle forme elle prendra, le nombre de tours de dialogue qu'elle devrait nécessiter pour arriver au but voulu, etc.

Plusieurs systèmes tutoriels intelligents utilisent le langage naturel pour communiquer avec leurs utilisateurs. Chacun de ces systèmes a développé son planificateur selon les besoins propres à la matière qu'il enseigne. Selon les auteurs de BEETLE (Zinn et al. 2002), il y a trois types de planificateurs qui existent. Le premier type est les machines à états finis. Ce type de planificateur est un système de dialogue où chaque acte du dialogue est écrit d'avance ainsi que la liste des réponses possibles ainsi que la liste des interventions suivant la réponse. Le deuxième type est un planificateur de dialogue à base de formulaire. Dans ce cas-ci, le système possède une série de formulaires qu'il tente de remplir selon les informations obtenues de l'utilisateur. Le système possède une série d'actes qui lui permet d'obtenir plus d'information de l'utilisateur pour l'aider à remplir ses formulaires. Le troisième type est une approche modulaire qu'ils ont nommée composantes de dialogue. Avec cette approche, chaque dialogue est divisé en sous-dialogues jusqu'à représenter des actes de dialogues atomiques. Cette forme de planificateur permet de créer des dialogues complexes en combinant les actes de dialogues atomiques.

Dans les sections suivantes, nous allons voir quelques exemples de planificateur pour chacun de ces types de planificateur, soit les cas d'Atlas et d'AUTOTUTOR pour les machines à états finis, et le cas de BEETLE pour les planificateurs à composante de dialogue.

## 2.1 Machine à états finis

### 2.1.1 Atlas

L'université de Pittsburgh a développé Atlas, un système de communication en langage naturel (Freedman et al. 2000) qui se veut indépendant du domaine. Il se compose de deux parties : un module de compréhension du langage naturel nommé CARMEL et un planificateur nommé APE (*Atlas planning engine*). Ce système a tout d'abord été utilisé et testé dans le STI Andes (Atlas-Andes (Freedman et al. 2000)), puis dans d'autres systèmes tutoriel tels que WHY2-Atlas (Jordan et al. 2006). Une partie d'Atlas, soit le planificateur APE, est utilisée dans CAPE (CIRCSIM-Tutor/APE (Kim et al. 2006).) Dans les sections qui suivent, nous verrons sommairement CARMEL, puis APE. Finalement, nous présenterons un outil développé pour aider à la construction de KCD (*Knowledge Construction Dialogue*) pour faciliter l'utilisation d'Atlas.

#### 2.1.1.1 CARMEL

CARMEL (Freedman et al. 2000) est un module de compréhension du langage naturel basé sur une approche syntaxique profonde fondée sur le cadre. Ce module commence par effectuer une correction grammaticale sur le texte de l'apprenant. CARMEL effectue ensuite une recherche lexicale où sont attachées les fonctions sémantiques à chacun des mots de la phrase. Cette attache sémantique des concepts aux mots de la phrase couplée à l'analyse syntaxique de la phrase produit un graphe sémantique. Normalement, ce graphe sémantique devrait être le même (ou du moins très similaire) entre 2 phrases écrites différemment, mais représentant la même idée. Une autre capacité de Carmel est d'effectuer des analyses partielles sur des phrases trop longues ou mal construites. Cette analyse permet au système de déterminer des parties manquantes de la réponse, ainsi que les erreurs et les incompréhensions.

Un des défauts de ce système est sa difficulté de reconnaitre des phrases de même sens qui utilisent des mots très différents. Malgré le fait que le graphe sémantique permet au système de comprendre des phrases qui utilisent des constructions différentes (inversion, passif, actif), mais avec des mots semblables, il n'arrive pas à déterminer la ressemblance entre une phrase utilisant un mot et une autre utilisant la définition ou la description de ce mot (ex. : Lune et satellite naturel de la Terre.)

#### 2.1.1.2 APE

APE (Freedman et al. 2000) est un planificateur réactif qui travaille sur des plans partiels qui sont étendus et raffinés seulement lorsque nécessaire. Ce planificateur est composé d'épisodes tutoriels représentant les actions nécessaires pour aider l'apprenant à effectuer une étape dans la résolution du problème.

Le fonctionnement logique d'APE est assez simple à comprendre. Tout d'abord, un but initial est choisi. Le système recherche ensuite dans sa bibliothèque d'opérateurs toutes les opérations qui accomplissent le but recherché et dont les conditions préalables et les conditions du filtre sont satisfaites. En général, ces opérateurs possèdent plusieurs étapes. Le planificateur remplace alors le but par les étapes de l'opérateur choisi. Ces étapes de fonctionnement du planificateur sont répétées jusqu'à ce qu'une étape atomique soit trouvée. Le tuteur effectue ensuite cette étape atomique et selon que l'étape correspond à une question ou à une phrase déclarative (exemple : une explication), le tuteur arrête son tour ou continue d'exécuter le plan respectivement.

Le planificateur possède différentes habiletés qui lui permettent de s'adapter à l'étudiant et aux situations changeantes. La première habileté est sa capacité de changer de plan en cours d'exécution. Ceci est effectué à l'aide de trois types de plans différents :

1. Sauter le reste du plan choisi si les circonstances ont changé
2. Changer le plan courant par un nouveau plan qui répond au même but
3. Remplacer une série de buts en tête d'agenda

Cette dernière possibilité est plutôt utilisée pour répondre à une question de l'étudiant sans déranger le plan global.

Une deuxième habileté est de pouvoir adapter la réponse au contexte. Ceci se fait de deux façons. La première utilise un champ dans l'opérateur qui conserve le but ayant appelé le plan. La seconde s'appuie sur la mise à jour et la vérification de prédicat dans une base de données. Selon les concepteurs d'APE, la grande différence entre les machines à états finis habituelles et APE est sa capacité de gérer des prédicats complexes qui sont évalués lors de l'exécution traitant une classe d'entrée.

2.1.1.3 KCD Tool Suite

La machine à états finis d'APE est représentée par un réseau. Dans ce réseau, les phrases du tuteur représentent les états, les arcs correspondent aux bonnes réponses de l'étudiant et les « *push* » représentent les mauvaises réponses. Ces poussées appellent un sous-dialogue qui permet de revenir au dialogue initial lors de leur complétion.

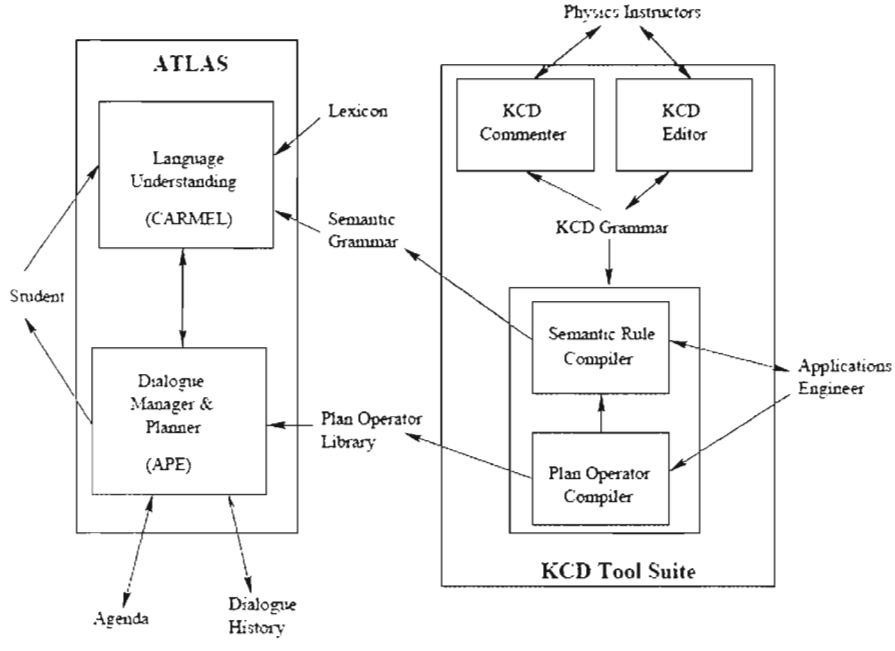



**Figure 2.1** Atlas et la suite d'outils KCD (Freedman et Al. 2000)

Une suite d'outils a été développée pour faciliter la construction du réseau de plans pour Atlas. Elle comporte quatre outils, soit d'un éditeur, d'un commentateur, d'un compilateur de plans et d'un compilateur de règles sémantiques. Ces outils partagent tous une grammaire commune (grammaire KCD) représentant les différents éléments du réseau. Ces outils produisent les opérateurs d'APE et la grammaire sémantique de CARMEL.

L'éditeur offre une interface graphique qui permet d'entrer les phrases en langage naturel représentant les réponses des étudiants, ainsi que la façon dont le tuteur doit y répondre. L'auteur doit aussi relier ces phrases entre elles de la bonne façon pour que les compilateurs fonctionnent correctement.

Le commentateur permet à d'autres personnes telles que professeurs et coéquipiers de commenter le contenu du KCD, donnant à l'auteur une rétroaction sur les explications, ainsi que des correctifs à apporter ainsi que des façons alternatives pouvant être utilisées pour montrer différents concepts.

Ces deux outils transforment les phrases du langage naturel en éléments de grammaire KCD avant d'être transmis aux outils suivants.

Le compilateur de plans traduit le résultat des deux outils précédents en opérateurs utilisés par APE. Ces opérateurs contiennent les informations nécessaires permettant de relier les différentes actions tutorielles aux réponses de l'étudiant. Aussi, le compilateur utilise les intentions tutorielles pour que le planificateur puisse se promener dans le réseau sans que l'auteur ait à expliciter tous les liens transitifs.

Le compilateur de règles sémantiques extrait du réseau les réponses attendues des étudiants, puis il aide l'auteur à grouper des parties de réponses similaires et à les étiqueter sémantiquement. Le compilateur génère ensuite les règles de grammaire sémantique qui vont relier les réponses de l'étudiant aux différents plans du planificateur. CARMEL utilise cette grammaire sémantique pour analyser les réponses étudiantes et y attacher les concepts sémantiques au terme qu'il reconnait. Cette transformation permet au système tutoriel de comprendre les réponses de l'étudiant même s'il n'utilise pas les mots exacts de l'auteur.

### 2.1.2 AUTOTUTOR

AUTOTUTOR est un système tutoriel intelligent développé à l'université de Memphis (*Graesser* et Al. 2004) pour aider les étudiants à acquérir les connaissances de base sur les systèmes d'exploitation, l'Internet et le matériel informatique. Le planificateur d'AUTOTUTOR est une autre machine à états finis. Aussi, AUTOTUTOR utilise une méthode statistique pour la compréhension du langage naturel, soit la technique LSA (Latent Semantic Analysis.) Tout d'abord, nous allons voir en quoi consiste la technique LSA, puis nous allons examiner le fonctionnement du planificateur.

#### 2.1.2.1 LSA

LSA est une méthode statistique de compréhension du langage naturel de type sac de mots. Cette approche ne tient pas compte de l'ordre des mots dans la phrase, mais uniquement des mots eux-mêmes. L'idée ici est que des mots utilisés dans un contexte similaire comportent des informations sémantiques semblables. La technique produit donc une matrice qui représente les mots et leur nombre d'apparitions dans le texte. Cette matrice est ensuite décomposée en une série de poids et de vecteurs liés un à un avec les mots et les textes. La somme normalisée des vecteurs des termes du texte devient le vecteur du texte. La différence sémantique entre 2 textes équivaut à la distance entre les vecteurs représentant chacun des textes. Dans AUTOTUTOR, les vecteurs représentant les réponses attendues ainsi que les différentes mauvaises réponses démontrant des incompréhensions et les réponses incomplètes sont déjà calculés. AUTOTUTOR transforme la réponse donnée par l'étudiant en un vecteur normalisé et la compare ensuite avec ses vecteurs réponses. Les mesures prises par AUTOTUTOR visent la complétude de la réponse et la compatibilité avec la réponse idéale. La réponse est considérée comme bonne lorsque le résultat de ces mesures est supérieur à une certaine valeur.

Cette technique ne peut déterminer les informations manquantes dans les réponses. Cela explique que les réponses incomplètes et les réponses dénotant une incompréhension sont précalculées. Autre limitation, cette technique ne peut faire la différence entre deux phrases utilisant les mêmes mots, mais dans un ordre différent ou avec des virgules placées à des endroits différents provoquant une différence importante dans le sens des phrases.

2.1.2.2 Planificateur

AUTOTUTOR possède 5 composantes permanentes soit un répertoire de scénarios *curriculum*, des modules de calcul linguistique, un ensemble de documents, un glossaire et les vecteurs pour LSA. Les modules intéressants ici sont :

1. Le répertoire de scénarios qui contient tout le matériel nécessaire associé à une question. Il recèle pour chaque question la réponse idéale, l'ensemble des attentes, les groupes d'indices potentiels, de bonnes réactions aux indices et des assertions associées à chaque attente, la liste des incompréhensions possibles, ainsi que leurs correctifs possibles, une série de mots clés et de synonymes, un sommaire et des informations servant à la génération de texte et à la génération de gestes du tuteur animé.

2. Le dépôt de documents qui contient les textes, articles et autres matériels pédagogiques. Ces documents servent à répondre aux questions des étudiants en leur donnant un paragraphe ou un article sur le sujet.

3. Le glossaire qui contient les définitions des différents termes techniques du domaine enseigné. Ces définitions sont données à l'étudiant lorsqu'il pose des questions sur la signification d'un terme technique.

AUTOTUTOR conserve aussi différentes informations sur l'étudiant soit : ses habiletés, ses initiatives, sa verbosité (le nombre de mots employés par tour) et son progrès sur la question ou le problème en cours.

Le planificateur lui-même est une machine à états finis. Cette machine correspond à un réseau de transition d'états. Ce réseau comporte plusieurs parties soit :

1. Nœuds : Représentent les états des buts tutoriels (c'est-à-dire : ce que le tuteur cherche à faire en rapport avec ses attentes actuelles) et les états du dialogue (ce qui a été dit et à quel tour.)

2. Arcs : Représentent les actes que le tuteur peut poser durant le dialogue comme des demandes d'information et des offres d'indices, ainsi que des marqueurs liant ces actes. Un arc est traversé uniquement lorsque ses conditions sont remplies.

Pour le choix de l'arc à suivre, plusieurs algorithmes ont été considérés comme les règles de productions floues ou encore une suite fixe d'actes tutoriels permettant d'atteindre un but X. Le fonctionnement de base de leur planificateur reste simple, selon un état donné, choisir un acte tutoriel nous rapprochant du but à atteindre.

## 2.2 Planificateurs à composante de dialogue

### 2.2.1 BEETLE

BEETLE (Zinn et al. 2002) est un système tutoriel intelligent développé à l'université d'Edinburgh pour permettre de tester une nouvelle architecture de planification du dialogue. BEETLE cherche à enseigner les concepts de base en électronique et en électricité.

#### 2.2.1.1 Architecture générale

L'architecture de BEETLE se veut une architecture en 3 tiers situés dans la partie de la génération de réponse dans la figure 2.2. Elle contient 3 modules, soit :

1. Module d'interprétation : Ce module sert à interpréter, analyser et rendre compréhensible pour la machine toutes les entrées faites par l'utilisateur. C'est ici que se trouvent les composantes servant à la compréhension du langage naturel et à la détection des actions dans l'interface utilisateur graphique.

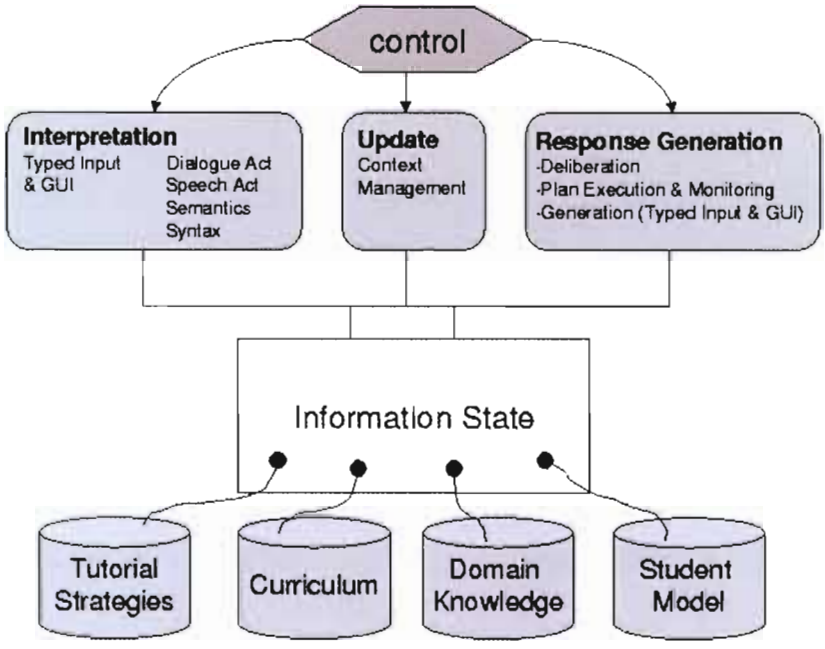



**Figure 2.2** Architecture générale de BEETLE (Zinn et al. 2002)

2. Module de mise à jour : Ce module contient les règles pour maintenir le contexte du système. Ce module aide à conserver les actes tutoriels effectués et aussi à déterminer ce à quoi le système doit s'attendre pour la suite (continuer de discourir, attendre la réponse de l'utilisateur, etc.)

3. Module de génération de réponse : Ce module sert à planifier et à produire les sorties du dialogue en langage naturel. Ce module se décompose lui-même en 3 tiers qui seront décrits un peu plus loin.

L'architecture possède aussi un espace où se retrouvent toutes les informations sur le contexte actuel et servant de pont avec les modules de données comme les stratégies tutorielles, le curriculum, le modèle du domaine et le modèle de l'apprenant. Cet emplacement que les auteurs nomment « Information State » sert de plateforme de communication entre les différents modules.

2.2.1.2 Architecture en trois tiers du planificateur

Le système de planification est situé dans le module de génération de réponse. Cette architecture en trois tiers est composée d'un planificateur délibératif, d'un système d'exécution de plan supervisant aussi leur déroulement et d'un système d'exécution. La figure 2.3 montre l'emplacement relatif de chacun de ces systèmes, ainsi que leur chemin de communication. Les différentes fonctions de chacun des tiers sont les suivantes :

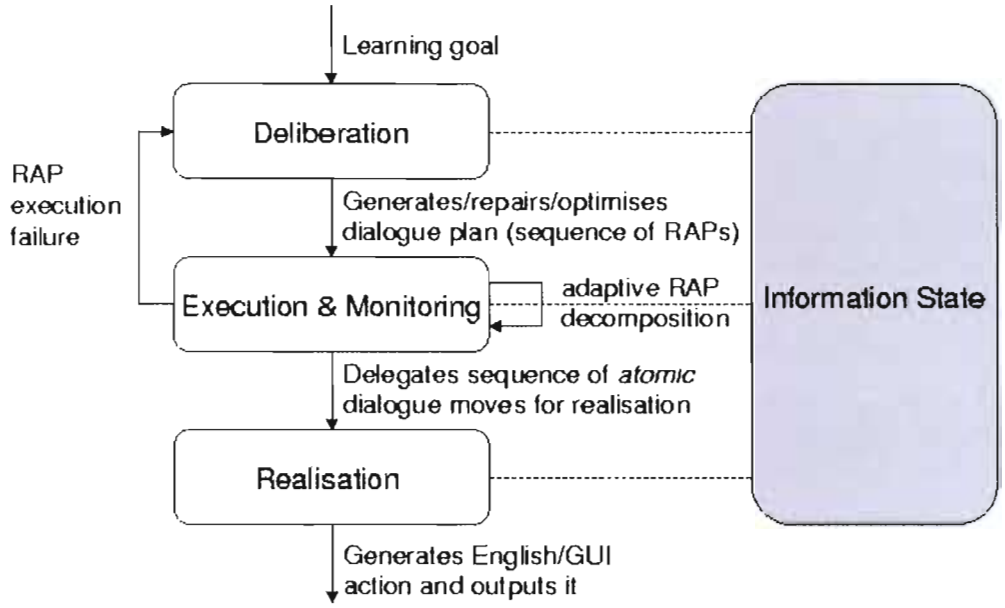



**Figure 2.3** Architecture en 3-tiers du planificateur (Zinn et al. 2002)

1. Planificateur délibératif : Ce module s'occupe de définir dans un haut niveau ce qui doit être fait pour atteindre un but. Il choisit d'une façon très large les actes à utiliser, sans pour autant choisir la méthode pour exécuter l'acte. Ce module vérifie toujours l'état du plan de dialogue pour tenter de relever les problèmes possibles et de les corriger ou encore d'optimiser le plan. Il sert aussi à trouver un acte alternatif lorsque l'acte tenté n'atteint pas les objectifs.

2. Raffinement de plan : Ce module contient toutes les techniques et les méthodes disponibles pour effectuer un acte choisi par le planificateur délibératif. Il choisit de façon contextuelle la meilleure méthode à appliquer, qui peut être soit une action atomique, soit une série de tâches. Dans le cas d'une série de

tâches, chacune est ajoutée au plan, puis un autre tour de raffinement est exécuté.

3. Système d'exécution : Ce module contient tout ce qui est nécessaire pour exécuter une action atomique. Il planifie la génération d'actions multimodales à exécuter et les exécute. Ce système se veut multimodal dans le sens où il peut, pour atteindre son but, produire du texte en langage naturel ainsi qu'agir sur l'interface graphique.

Après l'exécution d'une action, le système doit choisir s'il doit continuer à exécuter d'autres actions ou à attendre que l'étudiant ait exécuté une action. Ce choix est relativement facile à faire. Si l'acte que le système a posé demande une action de l'utilisateur (soit une réponse à une question ou une manipulation sur l'interface,) le système attend que l'étudiant ait effectué cette action. Dans les autres cas, le système garde son tour et continue d'exécuter ses actes. Si l'étudiant prend trop de temps à effectuer la tâche demandée, le système va décider s'il lui laisse plus de temps ou s'il lui offre de l'aide selon le contexte.

Tous ces systèmes de planification ont leurs avantages et leurs désavantages. Le choix du planificateur devra être fait selon le système de base utilisé soit CTS et le respect que l'implémentation du système de planification aura de la théorie de CTS. Le prochain chapitre décrira le fonctionnement général de CTS et de la théorie sous-jacente. Ensuite, nous regarderons chacun des systèmes de planification selon des exigences posées pour choisir le système de planification respectant le mieux l'architecture de CTS.

# CHAPITRE III

# DESCRIPTION DE CTS

CTS est un système tutoriel intelligent développé au laboratoire du GDAC. Il a été conçu par Daniel Dubois et implémenté par Patrick Hohmeyer. Il a ensuite été modifié et augmenté par d'autres étudiants à la maîtrise et au doctorat. Aussi et malheureusement, puisqu'il n'y a aucun développeur permanent sur ce projet, plusieurs problèmes de fonctionnement et de stabilité sont apparus. Ces problèmes seront énoncés dans les chapitres qui suivent ainsi que les correctifs que j'ai apportés.

CTS est un système fondé sur les principes de la conscience d'accès (aspect central de la théorie de l'atelier global (Baars, 1997)) basé sur les travaux effectués par Franklin et son système IDA (Franklin 2005.) IDA est un agent développé pour l'armée navale américaine dans le but de gérer les mutations des marins selon les souhaits de ceux-ci et les restrictions fournies par l'armée. L'agent IDA communiquait sous forme de courriel. En gros, le marin envoyait sa requête de mutation à IDA puis IDA effectuait le choix de l'endroit où le marin serait envoyé. Si IDA manquait d'information, il envoyait au marin un courriel demandant ces informations.

La principale caractéristique d'un système doté d'une conscience d'accès est sa capacité de choisir parmi plusieurs informations entrantes l'information la plus importante à traiter. Cette caractéristique sera décrite plus loin dans le chapitre.

## 3.1 Description générale de l'architecture

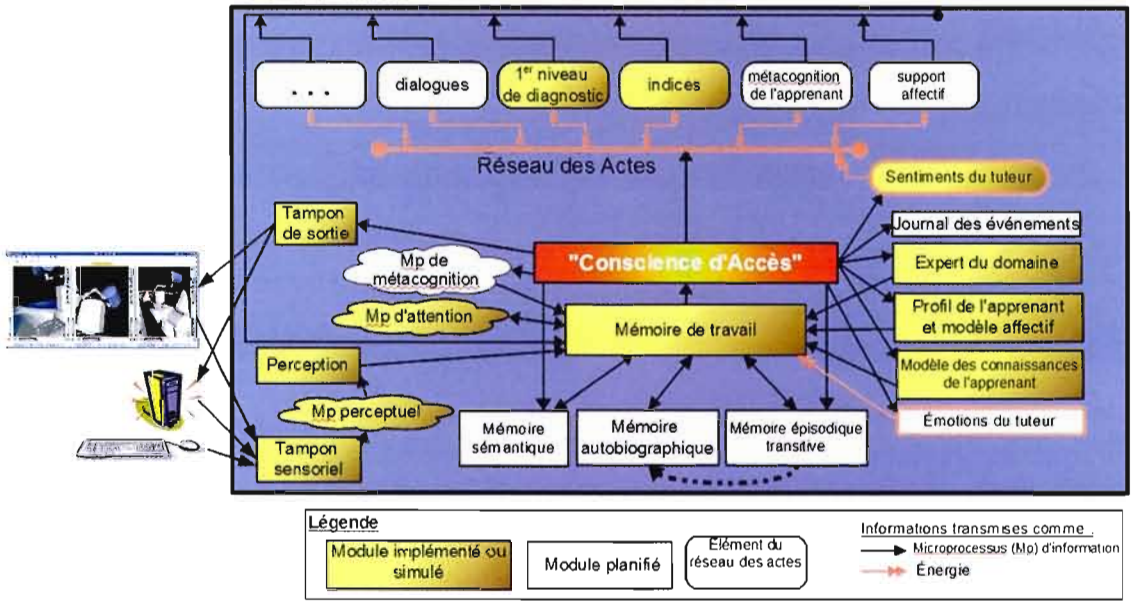



**Figure 3.1** Architecture générale de CTS

La figure 3.1 montre la structure interne de CTS. Nous retrouvons en jaune les parties qui ont été réalisées ou simulées.

1. **Tampon sensoriel, Microprocessus perceptuels, perception** : Ces éléments constituent le point d'entrée du système. Toutes les informations provenant de l'extérieur de CTS passent par ici et sont prétraitées pour qu'ils deviennent compréhensibles pour CTS. Dans la version initiale du système, seulement les actions du simulateur étaient perçues.

2. **Mémoire de travail** : Cet élément du système contient toutes les informations reçues par le système et qui compétitionnent pour être traitées.

3. **Mémoires sémantique, autobiographique et épisodique** : Ces éléments devraient contenir les données sur les expériences passées de CTS. Ces

éléments n'étaient pas implémentés ni même simulés dans la version initiale de CTS.

4. **Expert du domaine, modèles de l'apprenant** : Ces éléments contiennent les bases de données et les informations concernant des domaines de connaissances précis. Ils servent aussi à faire du traitement particulier sur les informations reçues selon leurs domaines particuliers. Ces éléments sont simulés dans la version initiale de CTS.

5. **Conscience d'accès** : Cet élément représente la transmission des informations déterminées prioritaire par le système à toutes les autres parties du système pour obtenir un traitement plus poussé et effectuer une prise de décision.

6. **Réseau des Actes** : Cet élément est l'élément du système qui lui permet de prendre des décisions. Selon les informations reçues provenant de la conscience d'accès, un acte est choisi pour être exécuté.

7. **Microprocessus** : Ces éléments représentent les plus petites unités de traitement du système. Il en existe de nombreuses variétés chacune ayant sa fonction propre. La liste des différents Microprocessus et leurs fonctions sera donnée un peu plus loin.

8. **Tampon de sortie** : Cet élément représente les différents points d'accès aux périphériques de sortie du système. Dans la version initiale de CTS, il correspondait au point de communication avec le simulateur et à une fenêtre de sortie texte.

CTS fonctionne selon le cycle cognitif développé dans IDA pour coordonner l'ordre d'exécution des différents modules. Ce cycle, qui sera décrit dans la prochaine section, effectue 10 rotations par seconde.

Dans les prochaines sections, nous allons voir plus en détail le théâtre (qui corresponds à la mémoire de travail) et le cycle cognitif, les différents Microprocessus et le réseau des actes.

## 3.2 Théâtre et le cycle cognitif.

Le théâtre est l'emplacement central de CTS. C'est ici que toutes les informations du système transigent. La boucle de fonctionnement centrale pour CTS se retrouve aussi dans ce module.

Les informations et Microprocessus du système peuvent être dits soit sur scène, ou hors scène. Ces deux états de l'information déterminent ce que le système doit faire comme traitement.

- **Hors scène** : les informations qui sont hors scène sont des informations qui sont tombées en dormance parce qu'il n'avait plus assez d'intérêt pour le système, mais qui peuvent encore se réveiller et monter sur scène si le système perçoit l'information en question de nouveau. Si après un certain temps, l'information en dormance n'est pas réveillée, elle sera retirée complètement du système. Dans le cas des Microprocessus qui sont hors scène, cela signifie qu'ils effectuent du traitement de façon « inconsciente ». Ces traitements inconscients peuvent correspondre à différentes tâches comme l'écoute d'information particulière dans le cas de Microprocessus de confirmation ou d'enrichissement d'information.

- **Sur scène** : les informations qui sont sur scène sont des informations perçues ou créées récemment par le système. Les informations ici compétitionnent pour devenir « consciente » et être publiées à toutes les autres parties du système.

La compétition des informations est faite par regroupement de ces informations ensemble et par enrichissement de ces groupes. Ces regroupements dans CTS sont appelés des coalitions. Dans la version actuelle de CTS, ces coalitions sont formées suivant un algorithme très simple : tous les Microprocessus d'informations liés ensemble forment la coalition. Aussi, la seule partie du système actuellement qui enrichit les coalitions alors qu'elles n'ont pas été publiées est les Microprocessus de confirmation. La coalition qui sera

publiée est la coalition ayant la plus grande valeur d'activation. Cette valeur est calculée en ajoutant à la valeur d'activation du Microprocessus d'information de tête, la valeur moyenne d'activation des fils du Microprocessus de tête.

Le cycle cognitif qui est la boucle principale de CTS comporte huit étapes. Ces étapes sont :

1. **Perception** : À cette étape, le système fait une lecture de son module de perception pour obtenir le nouveau contexte externe du système. Dans notre système, les informations provenant de la perception sont transformées ici en Microprocessus d'informations et transférées à la liste d'informations à transférer dans la mémoire de travail (la section sur scène du théâtre.)

2. **Obtenir les souvenirs** : Cette partie non implémentée prévoit l'augmentation des informations provenant de la perception par les informations contenues dans ses mémoires. Le fait que les mémoires du système ne sont pas implémentées explique que cette partie n'est pas implémentée.

3. **Ajout des Microprocessus en attente à la scène** : Ici, on ajoute tout simplement les Microprocessus en attente à la scène. Cette étape est utilisée pour synchroniser l'ajout des informations avec le cycle cognitif pour s'assurer du bon fonctionnement du système.

4. **Exécution des Microprocessus** : À cette étape, les Microprocessus d'attention et de confirmation s'exécutent pour examiner ce qui vient de rentrer sur scène et enrichir ces informations lorsqu'applicables.

5. **Compétition des informations** : Ici, les coalitions se forment pour déterminer l'information qui sera choisie pour être publiée.

6. **Publication de l'information choisie** : La coalition avec la valeur d'activation la plus élevée est choisie et est maintenant transférée à tous les autres modules du système. Cette coalition est considérée comme consciente et les traitements suivants du cycle sont désormais considérés comme conscient.

7. **Choix de l'Acte à poser** : À cette étape, la coalition agit sur le Réseau des Actes pour énergiser les liens du réseau et activer les actes. L'acte avec la plus grande activation et ayant toutes ses conditions remplies est choisi.

8. **Exécution de l'Acte choisie** : À cette étape, l'Acte choisi lance l'exécution des Microprocessus d'action qui lui sont associés. Le système agit sur l'environnement interne ou externe à ce moment.

9. **Fin du cycle**, retour à l'étape 1.

Ce cycle cognitif est exécuté 10 fois par seconde. Le nombre de cycles est fixé pour que le système ait un temps de réponse relativement prévisible et pour pouvoir synchroniser plus facilement d'autres modules qui rouleraient de façon autonome en parallèle.

Un autre point d'intérêt, les valeurs d'activation des informations sont déterminées par le concepteur pour chaque information ou type d'information au moment de leur implémentation et le système possède une fonction de dégradation de l'activation qui correspond au vieillissement de l'information.

## 3.3 Microprocessus

Les Microprocessus représentent la plus petite unité de traitement dans CTS. Ces Microprocessus se veulent atomiques dans leur fonctionnement et coopératifs pour effectuer une tâche. Atomique, car chaque Microprocessus devrait effectuer une seule tâche bien précise et indissociable. Un exemple à cela serait l'ajout, la suppression ou la modification d'information. Chaque Microprocessus ne devrait pouvoir effectuer qu'une seule de ces opérations. L'ensemble de ces Microprocessus devrait pouvoir donner des résultats plus complexes en coopérant. Malheureusement, dans la pratique, très peu de Microprocessus sont réellement atomiques. Ceci est dû à la façon dont ces Microprocessus sont appelés. Ils sont censés fonctionner de façon parallèle et donc ils ne peuvent pas coopérer pour effectuer une tâche avec des opérations séquentielles. L'atomicité des Microprocessus a donc été un peu réduite pour pallier à ce problème.

Il existe 3 grandes classes de Microprocessus dans CTS. Ces classes sont dissociées dans le sens qu'ils ne partagent pas la même superclasse. La première classe « Codelet », soit Microprocessus, aurait dû théoriquement contenir, ou surclasser, tous les autres types de Microprocessus. En ce moment, cette classe surclasse les Microprocessus suivants : Microprocessus asynchrones, Microprocessus de confirmations, Microprocessus d'information, Microprocessus de métacognition et Microprocessus émotionnels. La deuxième classe de Microprocessus est les Microprocessus perceptuels. Cette classe ne contient que les Microprocessus utilisés aux systèmes de perception de CTS. La troisième classe de Microprocessus est les Microprocessus d'action. Ces Microprocessus représentent les actions que le système sait faire.

### 3.3.1 Description des Microprocessus

1. **Microprocessus asynchrone** : Ces Microprocessus effectuent du traitement en parallèle dans le système sans que d'autres modules du système ne le leurs aient demandé. Le traitement est effectué lors de la publication de la coalition choisie.

2. **Microprocessus de confirmation (mpc)** : Ces Microprocessus vérifient si une action a obtenu le résultat voulu. Cela est fait par la vérification des Microprocessus qui entrent sur la scène pour trouver l'information voulue. Si après un certain nombre de cycles prédéterminés par le concepteur, l'information voulue n'est toujours pas arrivée, le Microprocessus de confirmation envoie une alerte dans le système.

3. **Microprocessus d'information (mpi)** : Ces Microprocessus ont une fonction très simple, mais aussi très importante. Ils servent à contenir les informations dans le système et à les véhiculer. Ces Microprocessus contiennent plusieurs champs soit : Un champ pour le type de l'information, un champ pour la valeur de l'information (l'information elle-même), un champ pour le degré d'importance de l'information et deux listes, chacune contenant respectivement les parents et les enfants du Microprocessus d'information. Le

degré d'importance d'une information est déterminé par le concepteur et non par le système.

4. **Microprocessus de métacognition** : Prévus par la théorie, mais non implémentés, ces Microprocessus ont pour tâche d'aider à l'apprentissage du système. Ces Microprocessus devraient en théorie pouvoir dégrader ou renforcer des liens entre deux éléments du système selon le résultat des actions. Des exemples à cela seraient de renforcer un lien entre deux actes dans le Réseau des Actes lorsque le résultat du premier acte amène souvent au deuxième acte ou encore diminuer la force d'un lien entre deux informations dans sa base de connaissance s'il découvre qu'il y a une corrélation plus faible entre ces deux informations que ce que le système croyait.

5. **Microprocessus émotionnel** : Ces Microprocessus, encore en développement par Usef Faghihi, membre du laboratoire GDAC, au moment de l'écriture de ce mémoire, devrait servir à ajouter un degré d'importance émotionnel à une information ou à une coalition d'information.

6. **Microprocessus perceptuel** : Ces Microprocessus servent à activer, réactiver ou créer des Microprocessus d'information en utilisant les informations reçues de l'extérieur du système. Ces Microprocessus sont le point d'entrée pour toutes les données provenant de l'extérieur. Théoriquement, chacun de ces Microprocessus recherche une seule information parmi toutes les informations reçues du système perceptuel auquel il est attaché et convertit cette donnée en mpi, soit en a créant une nouvelle, ou en activant une existante. Il est possible qu'en activant un mpi existant, il en change aussi la valeur.

7. **Microprocessus d'action (mpac)** : Ces Microprocessus représentent les traitements et les actions que le système peut faire. Ces Microprocessus sont souvent rattachés à des actes du réseau des actes, et exécutés uniquement lorsque l'acte auquel ils se rattachent est déclenché. Idéalement, les traitements et les opérations contenus dans ces Microprocessus devraient être le plus atomique possible, par contre, certains doivent contenir plus d'un traitement si

ces traitements doivent être exécutés séquentiellement. La raison est que le système n'est pas capable d'exécuter ces Microprocessus séquentiellement, mais uniquement en parallèle.

## 3.4 Réseau des Actes

Le Réseau des Actes est la partie du système qui prend action selon les informations dont le système a conscience. Ces actions peuvent être de nature physique (action sur son environnement comme une modification sur l'interface ou l'envoie d'un texte), ou mentale (action interne sur ses données comme une modification sur des coalitions, création de nouveaux mpi et mise à jour de ses bases de données.) Le Réseau des Actes utilisé par CTS utilise un algorithme de sélection d'actes développé par Maes (Maes 1989.)

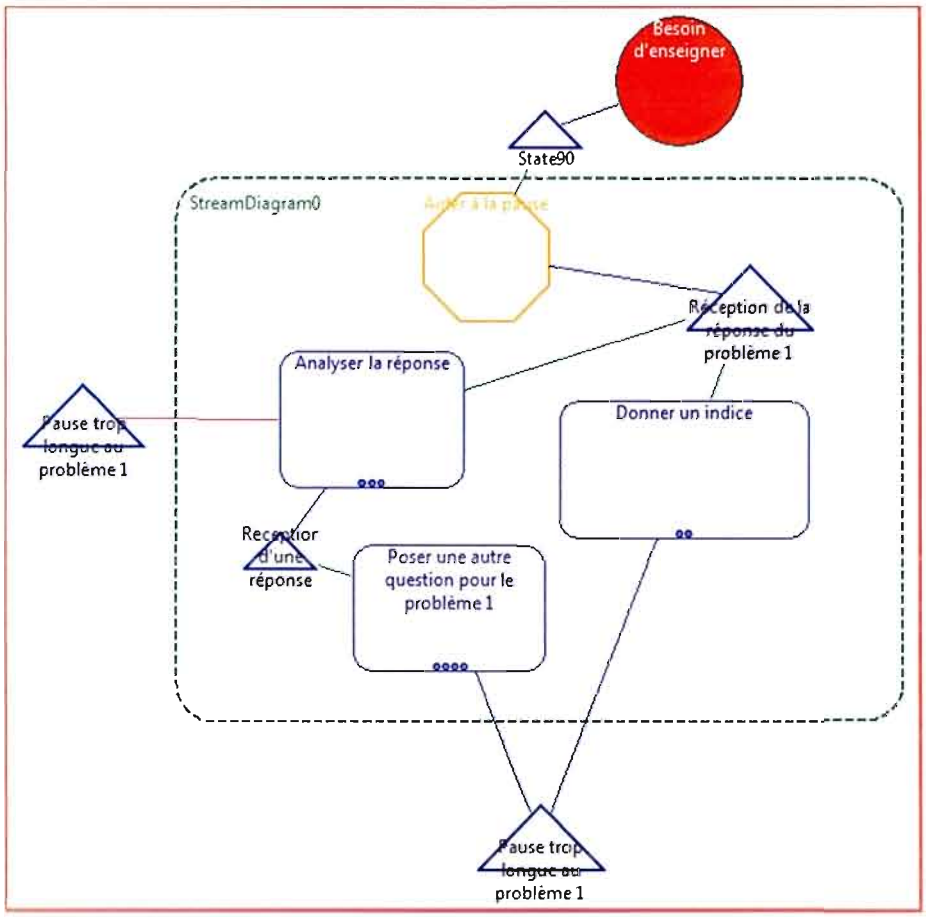



**Figure 3.2** Exemple de séquence dans le Réseau des Actes

À la figure 3.2, on peut voir un exemple fictif d'une séquence d'un réseau d'actes. Avant de survoler l'algorithme du mécanisme de sélection d'actes, jetons un œil sur les différentes composantes du Réseau des Actes.

1. **Motivateurs (cercle rouge)** : Les Motivateurs représentent les buts ultimes du système, ce dont le système a besoin. Si nous nous rapportons au cas des êtres vivants, les motivateurs représenteraient le besoin de sécurité, le besoin de survivre, le besoin à la propagation de l'espèce, etc. Cet élément du Réseau des Actes insuffle de l'énergie descendante dans la séquence d'Actes pour favoriser les actions qui répondent aux besoins du moment du système.

2. **États (triangle bleu)** : Les États représentent ce que le système croit présent ou absent de son environnement et de ses processus internes. Les États sont aussi générateurs d'énergie dans le réseau. Lorsqu'un État est vrai, il envoie de l'énergie ascendante, lorsqu'il est faux il laisse passer de l'énergie descendante et lorsqu'il est lié par un lien d'inhibition, il envoie de l'énergie négative.

3. **Actes (rectangle arrondi)** : Les actes sont composés de Microprocessus d'action qui produisent les traitements et les opérations voulus lorsqu'ils sont déclenchés. Ces actes peuvent représenter des actions internes au système, externes sur l'environnement ou les deux en même temps.

4. **But (octogone jaune)** : Les buts représentent ce que l'on veut atteindre comme résultat avec une séquence d'actes. Bien qu'ils soient implémentés dans le système, les buts ne sont utilisés que comme aide à la conception. Ils n'ont aucune autre fonctionnalité dans le système.

5. **Séquence (rectangle pointillé vert)** : Les séquences représentent une série plus ou moins complexe d'actes qui a pour objectif d'atteindre un but précis. Ces séquences ne sont utilisées que comme aide pour la conception des réseaux. Les séquences dans CTS sont un vestige d'un mode de fonctionnement utilisé dans le système IDA.

6. **Lien de précondition (ligne bleue)** : Ces liens lient les États qui doivent être vrais pour qu'un acte puisse être déclenché. Lorsqu'un État est vrai, il envoie de l'énergie d'activation à tous les actes qui lui sont reliés par ce type de liens. Lorsqu'un État est faux, l'énergie d'un acte est transmise via ces liens aux actes précurseurs.

7. **Lien d'inhibition (ligne rouge)** : Ces liens lient aux actes les États qui risquent d'être défaits par l'exécution de l'acte en question. Un État vrai envoie de l'énergie d'inhibition aux actes qui lui sont reliés par ce type de liens.

8. **Lien de postcondition (ligne jaune)** : Ces liens lient aux actes les États qui ont une chance de voir leur valeur devenir vraie par l'activation de l'acte. Ces liens aident à activer les actes qui ont une chance de rendre un État précurseur d'un acte subséquent à vrai, et aide aussi à activer les actes subséquents qui ont une chance de se déclencher suite à l'activation d'un État liant ces actes.

Le Réseau des Actes est calibré par cinq facteurs globaux dans le système soit :

1. $\theta$ : Représente le seuil de déclenchement d'un acte qui est réduit de 10 % par cycle où il n'y a pas eu de déclenchement d'acte. Lorsqu'un acte est déclenché, ce facteur revient à sa valeur initiale.

2. $\pi$ : Représente le niveau moyen d'activation du réseau. Cette moyenne doit être conservée dans le système de tour en tour par une fonction d'écrasement de l'énergie.

3. $\phi$ : Représente l'énergie envoyée par un État considéré vrai.

4. $\gamma$ : Représente l'énergie envoyée par un Motivateur activé.

5. $\delta$ : Représente l'énergie retirée par un lien inhibiteur lorsque son État est vrai.

La difficulté première avec le Réseau des Actes est la calibration de ces paramètres pour avoir un bon fonctionnement stable du système. Cette calibration doit être faite à la main et doit être ajustée lorsque le nombre d'éléments dans le réseau change. Aussi, dépendant si on veut un système plus réactif à l'environnement ou plus axé sur ses motivations, il faut calibrer $\phi$ et $\gamma$. Si $\phi$ est à 0, alors le système est purement axé sur ses motivations et si $\gamma$ est à 0, alors le système est purement réactif à son environnement.

## 3.5 Délibération

CTS se veut un système délibératif. La délibération ici indique la capacité aux différents sous-systèmes de CTS de communiquer entre eux par le biais des coalitions publiées et de modifier ces coalitions en y ajoutant de l'information. Cette délibération permettrait alors au système de prendre de meilleures décisions en mitigeant les informations reçues par la perception par les traitements effectués par ces sous-systèmes spécialisés. Malheureusement, bien que la délibération soit prévue dans la théorie de CTS, elle n'est pas implémentée dans la version de CTS décrite ici.

## 3.6 Sous-systèmes spécialisés

Les sous-systèmes spécialisés contenus dans CTS correspondent aux Experts du domaine et aux Modèles de l'apprenant. Ces sous-systèmes dans la version de CTS décrite ici ont été simulés à l'aide de Microprocessus asynchrones. Ces simulations sont malheureusement très pauvres, ne répondant qu'à des besoins très spécifiques, et ne sont pas réutilisables.

# CHAPITRE IV

# CHOIX DU PLANIFICATEUR

Le choix du planificateur dans CTS doit répondre aux critères qui ont été posés plus haut, soit :

1. Respecter et utiliser l'architecture CTS
2. Choisir parmi plusieurs méthodes de communication disponibles la méthode la plus appropriée au contexte
3. Pouvoir changer de plan lorsque le plan actuel est inefficace ou se retrouve inadapté au nouveau contexte
4. Effectuer des modifications ponctuelles au plan en cours sans perdre le fil de la discussion (exemple, durant une explication, l'étudiant pose une question, CTS devra pouvoir répondre à la question puis continuer son explication)
5. Utiliser différents modes de communication pour faire passer le message. (Exemple : Utiliser du texte en langage naturel, des images, des actions de l'interface utilisateur)
6. Un autre point important est aussi la capacité de pouvoir traiter et comprendre plusieurs sources d'entrées comme le texte et les actions effectuées dans le simulateur.

Nous allons tout d'abord examiner les raisons qui font que les planificateurs d'AUTOTUTOR et d'Atlas ont été rejetés.

## 4.1 AUTOTUTOR

Nous pouvons tout de suite éliminer le planificateur utilisé par AUTOTUTOR. Ce planificateur est en effet très mal adapté au domaine que CTS cherche à enseigner. Tout d'abord, le domaine concernant le contrôle du bras canadien est un domaine qui nécessite autant sinon plus de travaux pratiques que de théories. Cela signifie que bien que le domaine ait une part de théorie et donc peut contenir des textes et des articles, ce que le planificateur d'AUTOTUTOR peut traiter, il faut aussi que le planificateur choisi soit capable de réagir aux actions posées à l'intérieur du simulateur. Aussi, puisque le planificateur d'AUTOTUTOR a été conçu pour un domaine qui peut se montrer en utilisant uniquement du texte, il n'est donc pas conçu pour réagir aux actions provenant du simulateur sans avoir des modifications structurelles. Il ne répond donc pas au sixième critère.

AUTOTUTOR n'est capable que de donner des textes à ses utilisateurs pour fournir des explications et pour répondre à leurs questions. Il n'a donc pas été conçu pour pouvoir communiquer en utilisant des actions sur l'interface utilisateur par exemple. Ceci fait qu'il ne répond pas au cinquième critère.

Finalement, le planificateur d'AUTOTUTOR aurait été utilisé dans CTS comme un module externe et n'aurait alors pas répondu au premier critère de la recherche.

## 4.2 Atlas

Atlas, et plus particulièrement le planificateur APE, répond à plusieurs des critères du travail, mais pas à tous. Il est capable de changer de plan en cours d'opération lorsque celui-ci n'est plus adapté et il est aussi capable de modifier son plan pour s'ajuster à une situation nouvelle comme une question de l'étudiant, sans perdre le fil de son plan en cours. Il répond alors aux critères 3 et 4. Aussi on peut facilement créer différentes méthodes d'enseignement permettant au système de pouvoir choisir parmi plusieurs méthodes de communication, celle qui est la plus appropriée, répondant ainsi au critère 2.

Par contre, rien n'indique qu'Atlas peut traiter plusieurs types d'informations ou même qu'il peut communiquer sous d'autres formes que textuel. Il ne répond pas alors aux critères 5 et 6, bien que cela soit mineur dans ce cas si, puisque CTS aurait été là pour faire le pont entre les actions de l'interface et le planificateur de dialogue.

Malheureusement, Atlas est un module complet de planificateur de dialogue et n'aurait pu être utilisé que comme module externe à CTS et n'aurait pas ainsi utilisé l'architecture de CTS. Ceci fait qu'Atlas ne répondait pas au premier critère.

## 4.3 BEETLE

BEETLE est une architecture décrite, mais qui n'est pas implémentée en module qui peut être directement utilisé. Ceci signifie qu'il faudrait en faire l'implémentation complète à l'intérieur de CTS. L'ajout de cette architecture sera un peu plus difficile que les précédentes, mais elle permet de s'intégrer et d'utiliser l'architecture développée pour CTS. En tenant compte de cette particularité, regardons sa correspondance avec les critères fixés plus haut.

Le planificateur de BEETLE est décrit comme un planificateur de dialogue multimodal, qui peut tout aussi bien comprendre les actions de l'usager sur l'interface utilisateur que de lui-même faire des actions sur l'interface utilisateur. Aussi, dans ce contexte, on pourrait facilement envisager que le planificateur prenne en compte les émotions de l'utilisateur à l'aide des données que peuvent fournir une caméra et un logiciel de reconnaissance des émotions. Ceci répond bien aux critères 5 et 6.

Le module de mise à jour permet à BEETLE de tenir compte en continu du contexte actuel. Aussi, l'architecture du générateur de réponse indique que pour chaque étape possible, il peut exister une ou plusieurs méthodes qui effectuent la tâche de l'étape en question et qui est choisie en fonction du contexte. Ceci indique qu'il répond au critère 2.

BEETLE prévoit dans chacun des tiers du module de génération de réponse la capacité à effectuer des changements ou des demandes de changements au plan en cours s'il s'aperçoit que le contexte ne favorise plus le plan et doit en avoir un autre, ou encore si l'exécution du plan échoue, alors il doit en choisir un autre. Ceci répond au troisième critère.

Les articles sur BEETLE ne décrivent pas comment on peut insérer dans le plan existant une réponse à une question posée par l'utilisateur comme le requiert le critère 4. Par contre, puisque BEETLE ici est un modèle d'architecture et non pas un module, rien ne nous empêche d'en ajouter la fonctionnalité lors de son implémentation.

Maintenant, pour dire que l'architecture de BEETLE est compatible avec l'architecture de CTS, il faut identifier les correspondances entre les deux ainsi que déterminer les parties manquantes dans CTS.

## 4.4 Correspondance CTS/BEETLE

Tout d'abord, recherchons les correspondances entre CTS et les modules d'interprétation et de mise à jour qui sont demandées par l'architecture de BEETLE.

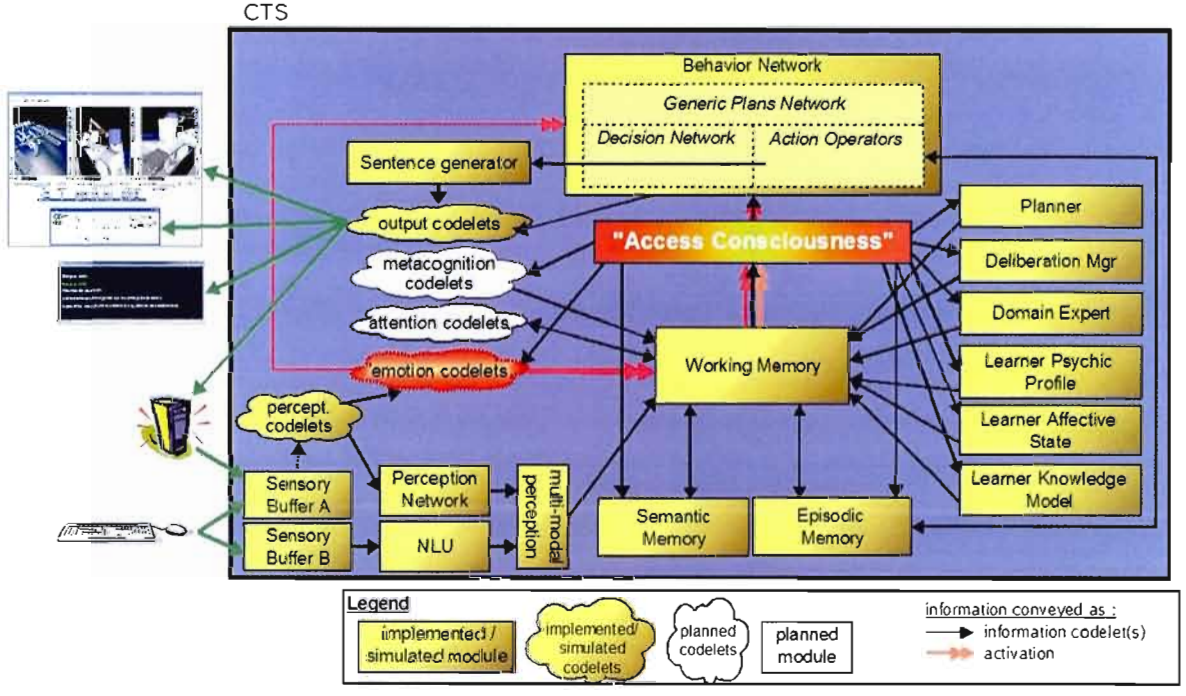



**Figure 4.1** Diagramme de CTS version 2

1. Module d'interprétation : Ce module dans BEETLE sert de point d'entrée. CTS possède déjà ce genre de module qui est représenté par la perception. En ce moment, la perception de CTS sert à recevoir les informations provenant du

simulateur, et nous devrons alors lui ajouter une perception qui lui permet de recevoir des entrées textes.

2. Module de mise à jour : Ce module dans BEETLE sert de mémoire contextuelle et épisodique. CTS possède sa mémoire de travail qui représente ici la mémoire contextuelle du système. Cette mémoire couplée aux États du Réseau des Actes permet à CTS de prendre des décisions contextuelles. La mémoire épisodique est prévue dans la théorie de CTS, mais elle n'est pas implémentée. La mémoire épisodique ici pourrait servir de lien entre ce que le système tente de faire, et les actions qu'il a posées dans le passé.

3. Module de génération de réponses : Ce module en trois tiers sert à effectuer la planification des dialogues et la génération des réponses.

    a. Planificateur délibératif : Cette partie sert à planifier le dialogue à haut niveau. Il n'y a pas de structure dans CTS permettant de conserver à long terme un plan d'action. Par contre, il est possible d'utiliser le Réseau des Actes pour choisir les plans génériques qui seront habillés durant la phase de délibération par les autres modules du système. Ces plans offrent une planification à court terme, et le RA peut être construit de manière à contenir des plans globaux.

    b. Raffinement de plan : Cette partie contient toutes les méthodes permettant d'accomplir chacune des étapes disponibles. Cette partie contient des RAP (*Reactive Action Package*) qui sont l'équivalent des actes et des séquences d'actes du Réseau des Actes dans CTS.

    c. Système d'exécution : Cette partie contient tous les traitements pour effectuer la tâche choisie, allant de la création et la génération de phrases jusqu'aux actions sur l'interface utilisateur. Dans CTS, cette partie correspondrait aux Microprocessus d'actions couplés à un module de génération de texte placé près du tampon de sortie.

Dans la prochaine section, nous allons voir les modifications et les ajouts effectués à CTS pour rendre le projet stable et plus près de la théorie pour pouvoir ajouter la capacité de planification telle que proposée par l'architecture de BEETLE.

# CHAPITRE V

# AJOUTS ET MODIFICATIONS POUR CTS

Pour stabiliser CTS et le rendre plus près de l'architecture de BEETLE, plusieurs ajouts et modifications ont dû être effectués. Dans ce chapitre, il sera questions de parties qui étaient prévues dans la théorie de CTS. Les ajouts comprennent la mémoire à court terme, le pilote de délibération, l'expert du domaine (en simulation) et les modules concernant l'apprenant (en simulation). Les modifications ont surtout été effectuées sur le Réseau des Actes et correspondent au retrait des États fantômes et à la conception du réseau lui-même.

## 5.1 Mémoire à court terme

La mémoire à court terme est une mémoire qui se veut précise sur les souvenirs qu'elle contient, capable de retracer les modifications d'une coalition et donc permettre de se rappeler le processus qui a permis d'atteindre un certain résultat. Aussi, elle se veut être capable de permettre de faire des associations entre différents souvenirs et permettre ainsi de conserver par exemple le contexte de la conversation. La mémoire à court terme correspondrait dans BEETLE à une partie du module de mise à jour, soit la mémoire épisodique.

### 5.1.1 Besoin relatif pour la conception de la mémoire à court terme

Tout d'abord, on veut avoir deux types d'accès aux souvenirs de la mémoire, soit un accès par rapport à un concept précis et aussi un accès par rapport à un souvenir précis. Ceci peut être considéré comme étant deux formes de mémoire distinctes, soit une mémoire associative ou sémantique pour un accès par rapport à un concept, et à une mémoire épisodique par rapport à un souvenir précis.

Une autre considération importante est la vitesse d'accès aux informations. La mémoire à court terme est une mémoire qui se veut rapide en accès et en ajout.

À partir de ces considérations, il faut choisir la structure de données qui pourra accommoder cette mémoire. Tout d'abord, l'utilisation de deux tables distinctes permettrait de distinguer les 2 formes d'accès à la mémoire, soit une table pour la mémoire sémantique et une autre pour la mémoire épisodique. Dans les deux cas, une table de hachage semble être la meilleure solution puisque ce type de table possède un temps constant pour ce qui est de l'accès à une information, ainsi que pour l'ajout. Ensuite, dans le cas de la mémoire épisodique qui doit conserver l'ordre d'arrivée des souvenirs, la classe *LinkedHashMap* représente probablement le choix le plus judicieux. En effet, cette classe conserve l'ordre d'entrée des informations, en plus d'être une table de hachage qui nous permet de définir nous-mêmes la clé des entrées. Dans le cas de la mémoire sémantique, un simple *HashMap* est suffisant. Dans ce cas-ci, la clé est un concept en particulier, et la valeur stockée est la clé du souvenir contenant le concept. La figure 5.1 montre un exemple démontrant l'organisation des données dans les 2 tables.

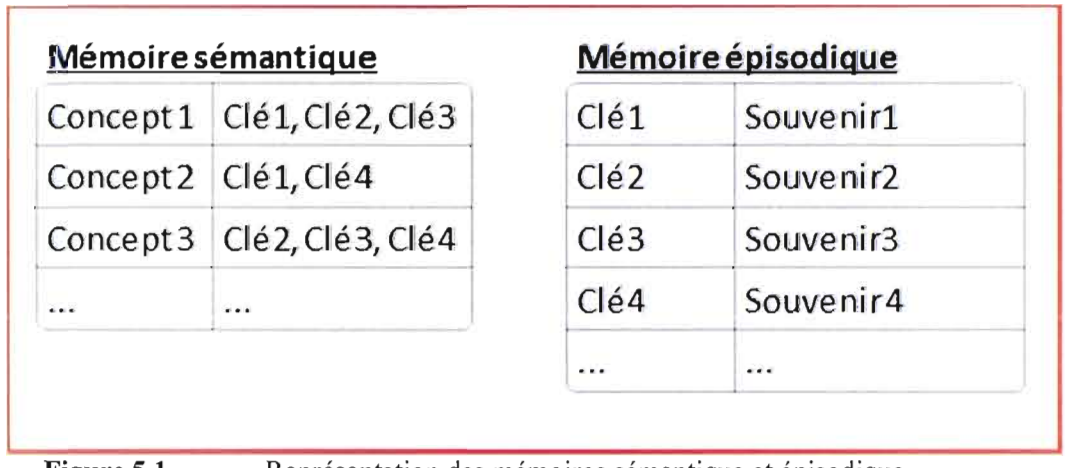



**Figure 5.1** Représentation des mémoires sémantique et épisodique

La liste de valeurs multiples dans la mémoire sémantique est représentée par une liste chaînée. Cette disposition des informations dans la mémoire sémantique permet aussi de faire des associations entre différents souvenirs. La table de la mémoire sémantique se retrouve alors avec le rôle d'une mémoire associative en même temps.

Pour la mémorisation des souvenirs, voici quelques considérations dont il fallait tenir compte, soit :

1. Les stades antérieurs des coalitions doivent être disponibles. Cette exigence a pour but de ne pas perdre les coalitions antérieures au cas où un traitement sur un souvenir ne serait pas terminé, mais qu'une autre version du souvenir est apparue.

2. La forme géométrique de la coalition doit être conservée. Cette exigence est là pour tenir compte du fait que le positionnement relatif des concepts est important au sens du message véhiculé par la coalition. Un exemple à ceci est deux personnes, Jean et Jacques, tous deux d'une ville X, vont souper dans un restaurant d'une ville Y. Jean et Jacques ont donc chacun ville X attaché à eux, et le restaurant a ville Y. On peut donc voir ici que si le système échange la ville entre Jean et le restaurant on vient de perdre le sens initial de l'idée sans pourtant avoir perdu de concept.

Une liste chaînée dans la table de la mémoire épisodique permet de remplir la première considération listée ici. Cette liste permettrait donc d'insérer de manière chronologique toutes les modifications apportées à une coalition lors des délibérations et des traitements effectués par le réseau des actes.

Pour la deuxième considération, deux options se sont présentées. La première option était de placer dans le souvenir les mpi eux-mêmes. Le grand danger qui a fait écarter cette option est la nature mutable des mpi. Un souvenir stocké par les mpi peut être modifié si le système change quelque chose sur le mpi, que ce soit sa valeur ou ses liens avec les autres mpi.

La deuxième option est de créer un arbre reproduisant la structure de la coalition de mpi que l'on stocke en mémoire. Cette option a pour avantage d'être immuable une fois créé. Le désavantage est qu'une coalition modifiée sera transformée en son entier en souvenir dupliquant ainsi beaucoup d'information. Ici, nous avons choisi la deuxième option pour son immuabilité.

Maintenant que les besoins ont été déterminés et que les choix généraux de la structure ont été faits, voyons à présent la structure et les fonctionnalités de la mémoire à court terme qui ont été mises en place.

### 5.1.2 Description de l'architecture

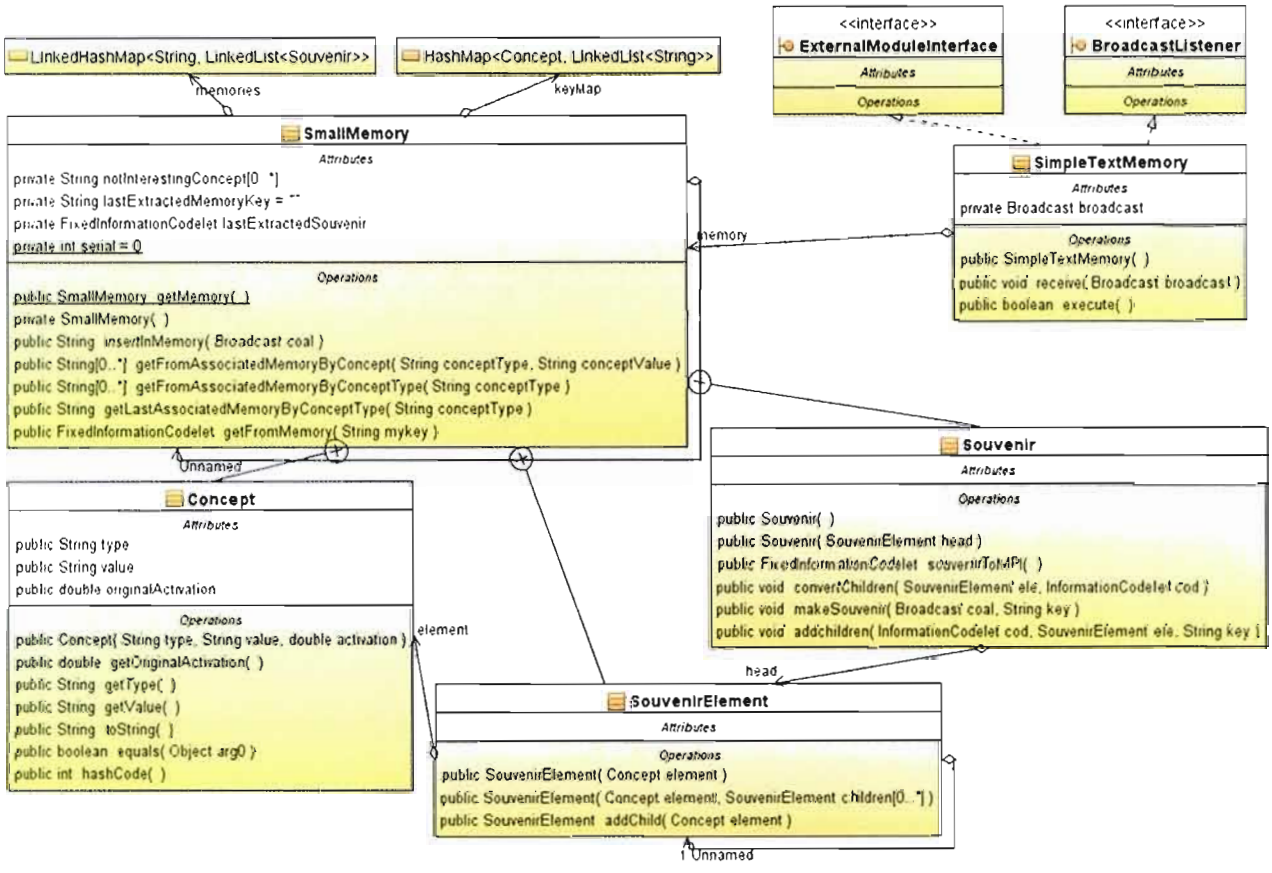



**Figure 5.2** Diagramme de classe de la mémoire à court terme

La figure 5.2 représente la structure de la mémoire à court terme. La classe *SimpleTextMemory* a pour fonction de faire le lien entre la mémoire à court terme et CTS dans le cycle cognitif. Le cycle cognitif a été légèrement modifié pour permettre d'ordonnancer l'exécution des agents « externes », plus spécifiquement pour que la mémoire à court terme soit le premier à s'exécuter après une publication et que la délibération soit le

dernier à s'exécuter après les autres agents externes tels que l'expert du domaine ou le modèle de l'apprenant. La raison pour cet ordonnancement est simple : dans le cas de la mémoire à court terme, l'enregistrement doit se faire avant toute modification du groupe mpi. Ici, je parle du groupe mpi, et non pas de la coalition, car dans l'État actuel de CTS, une coalition n'est qu'un sac contenant les mpi qui ont été choisis pour la publication. Si nous voulons conserver la structure des mpi, il faut alors descendre sous la couche de la coalition et malheureusement, il devient impossible lors de la lecture de l'arbre de mpi de savoir quels sont ceux qui ont été publiés. Dans l'algorithme de sélection actuel, c'est l'arbre mpi complet sur scène qui est sélectionné. Il faut donc que l'enregistrement soit fait avant l'exécution de tout autre module risquant de modifier l'arbre mpi.

La classe *SmallMemory* contient les mémoires à court terme. La mémoire épisodique est contenue dans un *LinkedHashMap* tandis que la mémoire sémantique associative est contenue dans un *HashMap*. La clé utilisée pour la mémoire épisodique est une clé de format clé-version. La partie clé de la clé est un nombre séquentiel fourni par la mémoire lorsqu'une coalition sans clé mémorielle est obtenue. La version représente un changement entre la coalition précédente et celle traitée si les deux possèdent le même numéro de clé. Lors de son insertion dans la mémoire épisodique, la partie clé est utilisée pour obtenir son emplacement dans la table de hachage et la partie version est utilisée pour l'insérer dans la liste de souvenirs associés à la clé. Dans le cas de la mémoire associative, la clé utilisée est un *Concept* qui représente un couple type valeur provenant d'un mpi. Pour chaque *Concept*, on y associe une liste de clés de souvenir où le concept est présent.

On dispose de trois fonctions d'obtention de souvenirs, soit :

1. *getFromMemory* : il permet d'obtenir un souvenir précis à partir de sa clé mémorielle. Cette fonction recrée l'arbre mpi contenu dans le souvenir et retourne le mpi de tête du souvenir. La mémoire se rappelle aussi le dernier souvenir extrait et en conserve le mpi de tête. De cette façon, si un souvenir est demandé pour un traitement par plusieurs Microprocessus différents, alors tous ces Microprocessus travailleront sur le même arbre mpi. C'est la fonction qui est la plus utilisée par le système.

2. *getFromAssociatedMemoryByConcept* : il permet d'obtenir la liste des clés mémorielles reliées aux souvenirs comprenant le Concept en question.

3. *getFromAssociatedMemoryByConceptType* : il permet d'obtenir la liste des clés mémorielles reliées aux souvenirs comprenant le type de Concept en question.

Avant de parler de la conversion de l'arbre mpi en souvenir, nous allons regarder les éléments composant un souvenir.

1. Classe *Souvenir* : Cette classe contient l'arbre de concept correspondant à un souvenir. Elle contient également les fonctions permettant de convertir un souvenir en arbre mpi et de convertir un arbre mpi en souvenir.

2. Classe *SouvenirElement* : Cette classe représente un nœud de l'arbre composant un souvenir. Elle contient un *Concept* et la liste des liens vers les *SouvenirElement* suivants.

3. Classe *Concept* : Cette classe contient les informations permettant de reconstituer un mpi soit le type de la valeur du mpi, sa valeur et son activation initiale. Cette classe est utilisée comme clé pour la mémoire associative en combinant les attributs type et valeur. La valeur d'activation initiale est utilisée lors de la création du mpi pour qu'il soit conforme au mpi initial.

L'algorithme de conversion d'un arbre mpi en souvenir et d'un souvenir en arbre mpi est sensiblement le même. La grosse différence est l'ajout des concepts dans la mémoire associative lors de la conversion vers un souvenir. L'algorithme de conversion utilise un simple algorithme de parcours d'arbre en profondeur récursif où l'on transforme chaque noeud en nœud correspondant à la conversion recherchée. Dans le cas de la transformation de l'arbre mpi en souvenir, on ajoute, pour chaque concept, la clé correspondante au souvenir dans la mémoire associative.

### 5.1.3 Faiblesses

Le but initial de cette mémoire à court terme était de traiter les arbres mpi produits dans le cadre de la planification de dialogue. Malheureusement, il existe des cas où la structure sous-jacente à une coalition n'est pas un arbre, mais un graphe cyclique, notamment dans le réseau perceptuel pour les données reçues du simulateur. Ceci est problématique lorsqu'il est couplé au fait qu'il faut aller lire la structure sous-jacente à la coalition pour obtenir la forme géométrique de l'arbre. Tel qu'il a été mentionné plus haut, lorsque nous lisons cette structure sous-jacente, le système voit tous les mpi du réseau perceptuel. Pour pallier à ce problème, la mémoire à court terme ne porte pas attention aux données provenant du simulateur en ignorant toute publication comportant ces mpi.

Une autre faiblesse est l'incapacité du système à comparer directement les arbres pour déterminer leurs différences minimes (versions) ou majeures (coalitions sur un autre sujet). Pour empêcher la mémoire de mémoriser plusieurs fois le même souvenir, le mpi réservé « *Same* » est utilisé pour indiquer que la coalition n'a pas changé depuis le tour précédent. Ce mpi est ajouté à la coalition par le Pilote des délibérations et retiré de la coalition par l'interface entre la mémoire à court terme et CTS.

Finalement, lorsque le système mémorise différentes versions d'un même souvenir, la nouvelle version du souvenir est enregistrée dans son entier. Ceci rend la mémoire à court terme gourmande en termes de mémoire système.

Dans de prochains travaux sur CTS, nous allons tenter de permettre à la mémoire de traiter des graphes circulaires et aussi de pouvoir conserver la forme géométrique de la coalition à l'intérieur de celle-ci, ainsi que d'étudier d'autres façons d'enregistrer les différentes versions d'un souvenir sans le dupliquer en son entier chaque fois.

## 5.2 Réseau des Actes

Le Réseau des Actes est parmi les modules les plus importants de CTS si ce n'est le plus important. Le rôle de ce système est la prise de décision sur l'acte à poser selon le contexte. Dans la prochaine section, nous verrons quelques difficultés clés de ce réseau et dans les sections subséquentes les modifications apportées pour le stabiliser et le rendre suffisamment opérationnel pour pouvoir implémenter la planification.

### 5.2.1 Difficultés et problèmes

#### 5.2.1.1 États fantômes

La première difficulté la plus évidente avec le fonctionnement du Réseau des Actes est ce que j'appelle les États fantômes. Ici, un État fantôme est un État qui continue d'exister même après que les éléments qui ont activé cet État ont disparu du système.

La raison de l'existence de ces États est double. La première raison est que le système n'a pas de façons de déterminer si le Microprocessus qui a enclenché l'État a disparu et par conséquent n'est pas capable d'éteindre l'État par lui-même. Dans mon exploration du fonctionnement du système, j'ai remarqué par contre une tentative de réparation de ce problème. Les liens d'inhibition entre un État et un acte comportent deux rôles.

Le premier rôle est le seul rôle qui lui est prévu par la théorie. Ce rôle est celui d'inhiber l'activation d'un acte lorsque celui-ci risque d'éteindre l'État inhibant.

Le deuxième rôle est une fonctionnalité ajoutée non prévue par la théorie et qui, à mon avis, fausse le fonctionnement du lien inhibant. Un lien inhibant dans CTS *désactive* l'État qui lui est attaché lorsque l'acte qui lui est relié se déclenche. Je considère ce fonctionnement comme faux pour les raisons suivantes. Tout d'abord, un lien inhibant, ou même un lien en postcondition, n'indique pas une certitude, mais plutôt une *possibilité* de l'effet de l'acte sur le système. Lorsqu'un acte se déclenche, il tente de faire une action qui peut donner comme résultats subséquents les États qui lui sont attachés, donc dans le cas de notre système, un État qui s'allumera s'il fait partie des postconditions ou un État qui s'éteindra s'il fait partie des inhibiteurs. Un exemple ici serait un robot qui prend une planche. L'action de prendre une planche est ici accompagnée d'un État de postcondition *Planche dans la Main* et d'un

État inhibiteur *Main Vide*. Dans un fonctionnement normal, lorsque le robot prend une planche, il a une *Planche dans la Main* et la *Main Vide* n'est plus. Il peut arriver que l'action échoue dans le cas où la planche tombe lorsqu'il la prend. Ici, aucun des États n'a changé, ce qui démontre que le changement d'État par l'acte n'est qu'une possibilité et non pas une certitude. Malheureusement, si l'on regarde la fonctionnalité ajoutée au lien inhibant, dans l'exemple de l'échec ci-dessus, le système croira alors qu'il n'a pas de planche dans sa main (la fonctionnalité des liens de postcondition reste conforme à la théorie), mais que sa main n'est plus libre non plus, ce qui risque de bloquer le système s'il n'est pas capable de réactiver l'État faussement éteint.

La deuxième raison à l'apparition des États fantômes est le fait que l'algorithme utilisé provenant des travaux de Franklin (Franklin 2005) a été modifié sans une compréhension approfondie des parties de l'algorithme qui ont été retirées. Dans l'algorithme initial de Franklin, le Réseau des Actes complet n'est pas fonctionnel dans le sens qu'il n'est pas utilisé dans son entièreté pour le calcul des décisions à prendre. Plutôt, lorsque des États apparaissent, le système de Franklin effectue une instanciation des séquences où ces États sont présents. Le calcul de la prise de décision est alors effectué sur ces instances et non sur le réseau complet. Lorsqu'une séquence est terminée soit par l'atteinte du but ou par l'inexistence d'États activés, elle cesse d'exister. Ceci retire en même temps tous les États fantômes restant avant qu'ils ne deviennent problématiques et empêche les niveaux énergétiques dans ces instances de réseaux de devenir instables.

Dans CTS, l'algorithme a été modifié pour n'avoir qu'une instance du Réseau des Actes en son complet. Je suis d'accord avec cette vision des choses bien que la majorité des problèmes d'instabilité du réseau vient de cette modification. La vision derrière cette modification est de donner au système le pouvoir de choisir en tout temps un acte provenant de l'entièreté de sa collection d'actes. Ceci donne la capacité au système de trouver une solution à un problème en suivant une séquence d'actes que le concepteur n'aurait pas imaginée. Malheureusement, cette instance « immortelle » s'empêche de nettoyer les États fantômes et les niveaux d'énergie circulant dans le réseau ne reviennent jamais aux valeurs initiales.

5.2.1.2 L'instabilité des niveaux énergétiques

Le Réseau des Actes choisit l'acte à poser dans un contexte donné à l'aide des deux conditions suivantes : que tous les États précurseurs à l'acte soient vrai (acte exécutable) et que son niveau d'activation soit le plus élevé parmi tous les actes exécutables et dépasse un seuil d'activation minimal. La première condition est facile à comprendre et n'occasionne pas de problèmes dans le système. Pour la deuxième condition, il faut que les niveaux d'énergie dans les actes aient une modification prévisible ce qui signifie que le transfert soit contrôlé et stable. Malheureusement, ce transfert d'énergie n'est pas stable et prévisible. Voici la liste des difficultés rencontrées et je doute qu'elle soit exhaustive.

1. Fonction d'écrasement de l'énergie dans le réseau peut faire gonfler le niveau d'activation dans les autres actes. Pour comprendre ceci, il faut d'abord savoir que Maes a spécifié que la *moyenne* des niveaux d'énergie dans le Réseau des Actes doit demeurer constante. Pour ce faire, la fonction utilisée dans CTS est la suivante :

$$En(x) = Ec(x) \times \frac{Mv}{Mc}$$

Où :

| | |
|---|---|
| En(x) = | Énergie normalisée d'un acte x |
| Ec(x) = | Énergie courante d'un acte x |
| Mc = | Moyenne courante de l'énergie dans le réseau |
| Mv = | Moyenne voulue de l'énergie dans le réseau |

Aussi, Maes indique qu'il faut réduire l'énergie d'un acte qui est exécuté à 0. Dans le cas où l'acte exécuté possédait une activation très supérieure à la moyenne, il peut alors arriver qu'après son exécution la moyenne du système tombe sous la moyenne voulue, et dans ce cas, la fonction d'écrasement gonflerait alors les énergies du réseau. Le tableau 5.1 donne un exemple de ce phénomène.

| | Courant sans exécution | Après normalisation | Courant après exécution | Après normalisation |
|---|---|---|---|---|
| Acte 1 | 15 | 12 | 15 | 22.5 |
| Acte 2 | 25 | 20 | 25 | 37.5 |
| Acte 3 | 35 | 28 | 0 | 0 |
| Moyenne | 25 | 20 | 13.3 | 20 |

**Tableau 5.1** Exemple de gonflement par la fonction d'écrasement

Comme nous pouvons voir dans ce tableau, pour ramener la moyenne à 20, il faut gonfler l'activation des autres Actes selon le facteur calculé; les actes déjà très énergisés reçoivent alors une plus forte dose d'énergie tandis que les actes qui sont vides d'énergie restent à 0.

2. Effet de « feed-back » : Cet effet a été décelé tôt comme étant une faiblesse de l'algorithme de Maes. Tyrell (Tyrrell 1994) a écrit un bon article décrivant bien ce phénomène dont je ne répèterai pas les détails ici. Ce phénomène est provoqué par l'acheminement de l'activation vers les actes suivants de façon à préparer les actes qui vont pouvoir s'exécuter après l'exécution de son prédécesseur et par l'acheminement de l'énergie vers l'arrière pour aider à lancer une séquence qui atteindra un but précis. L'énergie circulant dans ces deux directions provoque une boucle qui fait qu'une partie de l'énergie qu'un acte envoie lui est retournée au tour suivant. Une autre conséquence indésirable de ce phénomène est la transmission d'énergie qui est faite même si aucun État et aucun Motivateur ne sont activés.

3. Polarisation des actes : La combinaison des deux phénomènes décrits ci-dessus couplée à une valeur énergétique plafond provoque une tendance de polarisation chez les actes. En effet, les actes qui avec le temps deviennent fortement activés et qui sont reliés ensemble continuent de s'énergiser mutuellement avec à l'occasion un transfert d'énergie dépassant la fonction d'écrasement et donc ne perdant plus d'activation, tandis que les actes de séquences qui ont été déclenchés ont tendance à rester près de 0. Aussi, le gonflement artificiel aide à accélérer ce phénomène. Cette polarisation à

tendance à amener la valeur d'activation de la majorité des actes vers 0. Le tableau 5.2 montre cette tendance au cours des cycles.

| Valeur d'Activation | Nombre d'actes au tour : | | | |
|---|---|---|---|---|
| | Tour 6 | Tour 13 | Tour 488 | Tour 2310 |
| x > 50 | 0 | 0 | 2 | 2 |
| 25 < x < 50 | 1 | 1 | 0 | 0 |
| 10 < x < 25 | 2 | 1 | 3 | 3 |
| 2 < x < 10 | 23 | 20 | 3 | 2 |
| 1 < x < 2 | 25 | 25 | 0 | 0 |
| 0.5 < x < 1 | 0 | 4 | 0 | 0 |
| 0.1 < x < 0.5 | 65 | 0 | 15 | 15 |
| 0.0005 < x < 0.1 | 0 | 65 | 8 | 8 |
| 0 < x < 0.0005 | 0 | 0 | 68 | 49 |
| x = 0 | 0 | 0 | 17 | 37 |

**Tableau 5.2** Distribution des valeurs d'activation pour certains tours

4. Transmission d'énergie « *Front to back* » : Cette forme de transmission d'énergie est un cas aggravé de l'effet « *feedback* » décrit plus haut. La figure 5.3 montre l'effet sur 2 cycles.

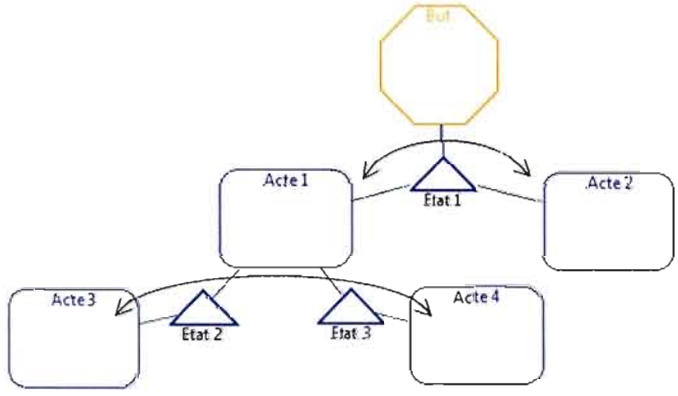



**Figure 5.3** Transmission d'énergie « ***Front to back*** » (sur deux cycles)

Dans la figure ci-dessus, on voit que pendant le premier cycle, les actes 3 et 4 transmettent de l'énergie vers l'acte 1 et les actes 1 et 2 transmettent de l'énergie vers le but. Au cycle suivant, par l'effet feedback, une partie de l'énergie ascendante du cycle précédent est retransmit du but vers les actes 1 et 2 et de l'acte 1 vers les actes 3 et 4. Ce qui signifie qu'une partie de l'énergie

ascendante de l'acte 2 est transmise en énergie descendante à l'acte 1 et vice-versa et une partie de l'énergie ascendante de l'acte 3 est transmise à l'acte 4 et inversement aussi. Cet effet aggravé de feedback peut envoyer de l'énergie destinée à favoriser le déclenchement de la séquence dont l'acte 2 fait partie dans la séquence de l'acte 1. Aussi, par la combinaison de l'énergie par feedback et « front-to-back », dans la figure 5.3, on peut facilement comprendre que l'acte 1 se verra rapidement énergiser.

#### 5.2.1.3 Manque d'expression

Une autre difficulté avec le Réseau des Actes est son manque d'expression lors de la description de ce qui peut déclencher un acte. La description du Réseau des Actes ne permet pas l'utilisation de liens ou lors de l'énumération de conditions déclencheurs. Ceci empêche la possibilité de décrire le déclenchement d'un acte comme une conjonction de groupe de conditions et. Un exemple serait de boire un liquide qui pourrait être déclenché par la condition a un verre et a du liquide ou a une tasse et a du liquide. Pour pouvoir exprimer cette condition dans CTS il va falloir dupliquer 2 actes identiques chacun avec un groupe de conditions et. Cette duplication demande aussi de devoir recopier tous les Microprocessus qu'il comporte, ajoutant ainsi beaucoup au temps de création du réseau. Une autre conséquence à ceci est l'incapacité de pouvoir créer des actes réutilisables. En effet, puisque chaque acte ne peut avoir qu'un seul groupe conditionnel, elle ne peut s'appliquer à une seule situation.

Un autre manque dans le Réseau des Actes actuel est l'absence d'État optionnel. Les États optionnels sont des États qui peuvent faciliter le déclenchement d'un acte en rendant cet acte plus précis, sans être obligatoire à son déclenchement. Ce manque oblige de créer plusieurs fois un acte avec une série de conditions différentes soit le cœur de condition obligatoire plus les conditions optionnelles. Pour un acte qui aurait 2 conditions obligatoires plus deux conditions optionnelles requerraient la création de quatre actes soit un pour les conditions obligatoires, une pour chaque condition optionnelle (conditions obligatoires + une condition optionnelle) et une pour les deux conditions optionnelles en même temps.

### 5.2.2 Solutions apportées

Je n'ai implémenté que les solutions qui ont demandé le moins de modifications possible dans le code source initial et qui ont stabilisé suffisamment le Réseau des Actes pour que je puisse placer le système de planification. Les raisons à ceci sont simples. La première est que ce mémoire porte sur l'installation d'un système de planification de dialogue et non sur le réseau des actes. Les problèmes énumérés plus haut ne représentent qu'une partie des difficultés rencontrées. La correction du réseau demanderait plus de temps que ce qui m'était disponible pour le présent mémoire. Aussi, je ne suis pas le seul à travailler sur CTS et je voulais que mes modifications soient facilement transférables aux autres versions de CTS.

Aussi, les modifications apportées ne règlent pas tous les problèmes cités plus haut. Par contre, il aide à stabiliser substantiellement le Réseau des Actes pour qu'il soit utilisable. Certains problèmes tels que l'instabilité des niveaux énergétiques se sont révélés très complexes dans leur provenance ou dans leur solution possible qu'ils ont été mis de côté parce que le réseau était suffisamment stable pour faire fonctionner le prototype.

#### 5.2.2.1 Clignotement des États

Cette solution aide à pallier au problème des États fantômes. L'idée derrière cette solution est très simple. CTS se veut un système conscient et que toutes les actions qu'il pose soient effectuées sur des informations dont il est conscient. Cette conscience d'informations provient au moment des publications du système. Il devient donc naturel de ne considérer que les informations qui sont publiées pour la prise de décision. Avec cette vision en tête, j'ai mis en place le clignotement des États. Cette technique comporte deux étapes : la remise à zéro des États et la remise sur scène des mpi publiés.

La remise à zéro des États est simple. Il suffit à la fin de chaque cycle d'éteindre tous les États allumés du réseau des actes. Cette simple remise à zéro permet de ne s'assurer qu'aucun État ne reflétant plus la situation continu d'exister dans le système.

La remise sur scène des mpi publiés permet à CTS de republier les groupes d'informations utilisés au cycle précédent. Il est nécessaire de faire cela, car CTS retire de scène toutes les informations qui ont été publiées et le Réseau des Actes peut prendre plus

qu'un cycle à obtenir le niveau d'activation nécessaire au déclenchement d'un acte. Le nombre de republication est géré par l'arbitre de délibération dont je parlerai un peu plus loin.

La remise sur scène ne garantit pas que la dernière coalition publiée soit choisie de nouveau pour la publication du cycle suivant. Elle respecte les valeurs d'activation de chacun des mpi du tour précédent ce qui peut occasionner le retrait normal des mpi de la scène par manque d'activation ou encore que des mpi plus importants soient choisis au cycle suivant. Lors du déclenchement d'un acte, la politique initiale de CTS de ne pas remettre sur scène les mpi de la coalition est respectée, ce qui empêche qu'un acte soit exécuté à répétition, si aucun autre mpi n'a suffisamment d'activation pour être publié.

5.2.2.2 Modification de vision et d'utilité de certains composants du Réseau des actes

Ces modifications n'ont pas nécessité de modifier le code source de CTS. Ils apportent des solutions temporaires à un problème d'expressivité et à un problème de stabilisation de l'énergie.

La première modification d'utilité a permis de placer dans le système des États optionnels. Ceci a été obtenu en élargissant le rôle des Motivateurs. Les Motivateurs dans CTS sont des générateurs d'énergie qui s'active ou se désactive selon le contexte du système. Il était donc simple d'ajouter la capacité de s'activer et de se désactiver selon la présence de mpi dans la publication. Cette capacité d'activation permet de favoriser certains actes selon la présence d'informations optionnelles sans pour autant brimer la capacité d'activation de l'acte lorsque ces informations optionnelles sont absentes. La figure 5.4 montre un exemple d'utilisation d'un État optionnel.

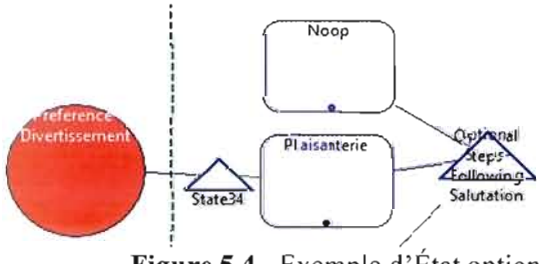



**Figure 5.4** Exemple d'État optionnel

Cet exemple montre une partie du plan de construction d'une salutation. Ici, l'État optionnel est *Préférence Divertissement* qui s'active lorsque le profil étudiant indique que l'étudiant aime les divertissements. Cet État optionnel vient insuffler de l'énergie dans l'acte plaisanterie qui va venir favoriser son activation. Bien que l'acte soit favorisé par cet État optionnel, cela ne garantit en rien son déclenchement, il est possible que d'autres facteurs aillent favoriser plus fortement un autre acte. Par contre, si l'étudiant n'a aucune préférence en ce qui a trait au divertissement, le système est libre de choisir entre l'acte *Plaisanterie* et l'acte *Noop* sans qu'il y ait de favoritisme dans le choix.

L'autre modification d'utilité est le retrait de l'utilisation des buts dans le système. Les buts n'ont aucune utilité système pour CTS, mais seulement une utilité marginale pour le concepteur. Jusqu'ici, il n'y a pas encore de raison pour les retirer, mais malheureusement, les buts ont tout de même un impact dans le fonctionnement du réseau des actes. Ils sont capables d'emmagasiner de l'énergie et de la redistribuer avec la transmission d'énergie vers l'avant et vers l'arrière comme les actes du système. L'avantage marqué du retrait complet de l'utilisation des buts est important. Il aide à réduire l'impact de la transmission d'énergie en « front-to-back ». Bien que ce retrait ne règle pas complètement ce problème (il reste encore le phénomène avec les actes comme nœuds intermédiaires), les séquences atteignant le même but se retrouvent avec une meilleur stabilité énergitique. Aussi, étant absents, les buts n'influencent plus le calcul de la moyenne de l'énergie dans le système. En effet, les buts emmagasinent de l'énergie, mais elle ne la vide pas. Ceci a pour conséquence d'accentuer la vitesse de polarisation du système.

L'utilité marginale des buts pour le concepteur provient de la nature floue de la définition d'un but dans le système. Un but actuellement dans le système était le résultat final recherché par l'exécution d'une séquence. Cette définition provient d'un vestige du fonctionnement de l'algorithme d'IDA. Dans IDA, le système choisissait les séquences à instancier en fonction des buts qu'elles atteignaient. Notre système n'utilisant plus les instanciations s'est retrouvé avec des éléments du Réseau des Actes devenu obsolète. Les buts font partie de ces éléments. J'ai proposé une nouvelle définition des buts dans le système qui simplifie la création des réseaux d'actes et qui ne nécessite pas la conception de buts individuels. Cette définition est la suivante : « un but est un désir de rendre vrai ou faux un ou

des États du système ». Cette définition permet de décrire pour chaque acte les buts qu'il peut atteindre en utilisant la conjonction entre un État et le lien le reliant à l'acte qu'il soit inhibant ou résultant. Bien que les implémentations des implications (pas tous définis encore) n'aient pas été faites, ce changement de définition ouvre de nouvelles voies dans l'utilisation du Réseau des Actes dans le CTS.

5.2.2.3 Description du Réseau des Actes en 2 tiers

La séparation du Réseau des Actes en 2 tiers a permis d'ajouter plusieurs fonctionnalités manquantes, mais voulues dans CTS sans en modifier le code. Tout d'abord, je vais décrire les différents tiers de cette façon de faire, ensuite un petit guide des règles à suivre lors de la construction d'un réseau et finalement, nous allons regarder les avantages et les inconvénients d'une telle façon de faire.

Le Réseau des Actes ici est divisé en 2 tiers soit un tiers décisionnel et un tiers opérationnel.

1. Tiers Décisionnel : Ce tiers correspond à la prise de décision dans le réseau des actes. On a nommé cette partie le Réseau des Actes décisionnel (RAD). Cette partie conserve la même fonctionnalité de base que le Réseau des Actes traditionnel c'est-à-dire l'énergie circule dans les différentes séquences existantes et déclenche des actes selon les mêmes règles que le Réseau des Actes traditionnel. La différence ici se situe en ce que fait l'acte en question. Il y a deux types d'action possible par les actes de ce niveau, pouvant donner lieu à une séparation en 2 tiers plus ou moins fortement couplés.

    a. Le premier de ces 2 tiers est la section des plans génériques. Ce tiers effectue les tâches de planification à haut niveau en fournissant des plans sur les étapes à suivre pour résoudre des problèmes plus complexes. Il est couplé au module de planification. Je donnerai plus d'information sur ce tiers dans la section sur la planification.

    b. La deuxième partie est la section des méthodes. Cette section contient les séquences d'actes pouvant effectuer différentes tâches que ce soit l'exécution d'une étape fournie par les plans génériques ou un autre

traitement portant sur des informations entrantes du système (exemple : diagnostic sur des entrées provenant du simulateur.) Cette partie est fortement couplée au tiers opérationnel. Par contre, aucun lien État à acte n'existe entre ces 2 parties. Le lien est fait par l'envoi d'un mpi représentant l'acte couplé à une clé mémorielle représentant la coalition qui a déclenché l'acte en question.

2. Tiers opérationnel : Cette partie n'est rien de plus qu'un ensemble d'actes contenant la logique de travail pour chaque acte que nous appelons ensemble des actes opérationnels (EAO). Il existe ici un acte pour chaque acte unique du RAD. Il est possible par contre d'avoir plus d'une représentation d'un acte dans le RAD (ie 1 acte opérationnel pour 1 à n actes décisionnels.) L'acte opérationnel contient tous les Microprocessus d'action (mpac) nécessaire pour effectuer sa tâche et tous les mpac de ce niveau contiennent la logique nécessaire pour extraire de la mémoire à court terme la coalition qui a déclenché l'acte. Aussi, à ce niveau aucun lien n'existe entre les différents actes.

La construction d'un réseau en utilisant cette technique est relativement simple. Tout d'abord, on construit le réseau décisionnel de la même façon qu'on le construisait avant. Par contre, on ne place pas le mpac de traitement dans ces actes, mais plutôt un mpac construit spécialement pour le RAD, soit le mpac *ActProposition*. *ActProposition* est un mpac qui envoie sur scène un mpi de type Acte avec comme valeur le nom de l'acte associé à un mpi de type *AssociatedKey* contenant la valeur de la clé mémorielle qui a déclenché l'acte. Une chose très importante, pour que *ActProposition* connaisse le nom de l'acte déclencher, il faut placer, à l'aide de l'éditeur de Réseau des actes, au niveau de l'ID du mpac, le nom de l'acte suivi d'un tiret et d'un numéro (ex. : *CréerTêteDePhrase-1*). Le tiret suivi du numéro assure que le nom est unique et *ActProposition* retire cette partie lors de la création du mpi acte.

Une fois le réseau décisionnel créé, on fabrique l'EAO. Pour chaque acte distinct du Réseau des Actes décisionnel (pour chaque nom d'acte ignorant le numéro), on crée un couple État acte. Le mpi déclencheur de l'État en question est de type *Acte* et sa valeur est le nom de l'acte. L'acte qui lui est associé contient tous les mpac qui vont effectuer le travail.

Une chose très importante, il ne doit pas y avoir de relation État-Acte entre le RAD et l'EAO. Concevoir autrement permettrait d'avoir un partage d'énergie entre ces deux sections et risquerait de fausser le fonctionnement du RAD.

Cette architecture nécessite l'utilisation de la mémoire à court terme pour le transfert des coalitions entre le RAD et l'EAO. Si la coalition était transférée sans être encapsulée dans la mémoire à court terme, le RAD risquerait de s'activer à répétition lors de l'apparition des informations contenues dans la coalition. Cette encapsulation permet donc au système d'avoir un meilleur contrôle sur le fonctionnement du RAD.

Ce type de fonctionnement apporte plusieurs avantages au système. Tout d'abord, le Réseau des Actes propose les actes qu'il veut effectuer avant de les exécuter. Cette proposition en mémoire de travail permet désormais aux différents sous-systèmes de s'opposer à un acte avant qu'il soit enclenché. Cette fonctionnalité était désirée dans la théorie de base de CTS.

Un autre avantage majeur de ce type de fonctionnement est la capacité de réutilisation des actes. En effet, il est maintenant possible d'utiliser des actes unitaires dans différentes séquences sans avoir à réécrire toute la logique de travail pour chaque acte. Ceci permet d'avoir des séquences plus courtes et plus réutilisables dans le RAD.

Aussi, puisqu'on n'a pas à réécrire toute la logique de travail de chaque acte dans le RAD, il devient alors plus facile de réutiliser à répétition le même acte pour créer des groupes conditionnels ((ET) OU (ET)). Ces avantages permettent d'ajouter plus facilement un peu plus d'expression au réseau des actes. Aussi, cette centralisation de la logique de travail permet de modifier celle-ci plus facilement, sans avoir à rechercher tous les actes affectés par cette modification.

Parmi les désavantages, celui qui est le plus important est l'augmentation du temps de réponse du système. En effet, avec la proposition d'acte, le nombre de cycles minimaux pour l'exécution d'un acte double ou presque. En effet, un acte, avant de s'exécuter, doit être proposé, ce qui nécessite alors un autre tour de délibération (détails sur ce point un peu plus

loin.) Par contre, nous avons jugé que l'obtention de la fonctionnalité de proposition d'acte et des avantages potentiels de cette fonctionnalité valait la perte de performance du système.

Un autre désavantage est le temps initial accru pour la création du Réseau des Actes. Il faut au début créer en double tous les actes du système. Par contre, la partie la plus longue à créer est la logique de travail derrière l'acte ce qui, à long terme, devient de moins en moins nécessaire à recréer dû à la réutilisation des actes dans le RAD. Aussi, des améliorations à l'éditeur du RA pourraient atténuer l'alourdissement dans la création.

## 5.3 Pilote de Délibération

Le pilote de délibération sert à contrôler la délibération dans CTS. La délibération est l'étape où tous les différents acteurs du système peuvent communiquer et donner leur point de vue sur la situation en cours. Une délibération peut prendre plusieurs cycles à compléter. Bien que la délibération ait été prévue dans la théorie, elle n'a jamais été implémentée.

Le rôle du pilote n'est pas de juger des informations transmises par les différents acteurs, mais plutôt de s'assurer que la délibération a lieu et qu'elle ne s'éternise pas. Ses fonctions sont de faire poursuivre la délibération tant que la coalition est différente, d'indiquer que la délibération est terminée lorsque la coalition cesse d'être modifiée et d'arrêter la publication d'une coalition acceptée si celle-ci n'a pas activé d'actes dans le réseau des actes.

Le pilote est exécuté durant l'étape des modules externes, comme dernier module. Ceci donne le temps à tous les autres acteurs d'agir avant qu'une prise de décision sur l'État de la délibération du tour en cours soit effectuée. Aussi, il est important de savoir qu'une information ajoutée au cycle courant n'est disponible qu'au cycle suivant puisque les acteurs ne considèrent que les informations publiées.

L'algorithme utilisé pour assurer ces fonctions est simple. Tout d'abord, la partie la plus importante est la vérification de la modification de la coalition. Ceci demande l'aide des acteurs externes. Différentes conditions aident à déterminer les différences entre les coalitions :

1. Coalitions différentes :

    a. La clé mémorielle de la coalition est différente de celle du tour précédent. Ici, on ne considère que la clé elle-même et non pas la version de cette clé. La raison s'explique par l'ordre d'exécution des différents acteurs comme la mémoire les acteurs externes et le pilote de délibération. Il est en effet possible que deux coalitions ayant 2 clés identiques et de versions identiques possèdent des mpi de type *Modified* (plus d'information plus bas), ou encore que deux coalitions ayant 2 clés identiques de versions différentes n'aient pas de mpi de type *Modified* auquel cas ils sont identiques. La figure 5.5 plus bas en donne un exemple.

    b. Contient des mpi de type *Modified*. Les mpi de type *Modified* sont ajoutés à la coalition par les différents acteurs externes lorsque ceux-ci effectuent un ajout, un retrait ou une modification de mpi sur la coalition. Les mpi de type *Modified* font donc partie des mpi réservés à la communication intrasystémique.

2. Coalitions identiques : Deux coalitions sont considérées identiques si les clés mémorielles sont identiques ou qu'une des deux clés n'existe pas et qu'elles ne contiennent pas de mpi de type *Modified*. Regardons le cas où une des clés mémorielles est manquante. Dans le cas où elle était manquante au cycle précédent, il est très facile de s'imaginer que la coalition était nouvelle et qu'elle vient de recevoir sa clé mémorielle. Dans le cas où elle n'est pas présente au cycle courant, c'est pour prévenir d'avoir des cycles infinis dans le cas de coalitions qui sont ignorées par la mémoire à court terme comme les coalitions cycliques par exemple ou encore par la coalition de départ « *StartingCodelet* ».

Une fois que le Pilote des délibérations a déterminé si la coalition est identique ou différente, il vient le temps d'effectuer la tâche de pilotage. Dans le cas de coalitions différentes, le pilote remet ses compteurs de cycles délibératifs à zéro et retire tous les mpi de

type *Modified* et de type *Coalition Accepted* (la délibération recommence et par conséquent, il ne faut plus faire de traitement « moteur » par rapport à la coalition.) Dans le cas de coalitions identiques, le pilote incrémente le compteur de cycles délibératifs de 1 et insère dans la coalition un mpi de type *SAME* pour indiquer à la mémoire à court terme que la coalition est la même qu'au tour précédent. Lorsque ce compteur atteint une valeur limite, le pilote de délibération place un mpi de type *Coalition Accepted* dans la coalition et l'on remet les compteurs à zéro. En ce moment, la formule utilisée pour calculer cette valeur limite est le nombre de cycles moyen nécessaire aux acteurs pour réagir aux informations de la coalition (qui est de 1 cycle pour l'instant) plus 1 cycle pour permettre la publication des nouvelles informations. Lorsque la coalition est acceptée, le pilote commence à compter un nombre de cycles pour permettre au Réseau des Actes de réagir à la coalition. Suite à la complexité du calcul d'activation des actes dans le réseau des actes, ce nombre de cycles a été déterminé par essai-erreur. Si aucun acte n'est déclenché avant la fin de ce compte à rebours, la coalition cesse d'être remise sur scène.

Le principal inconvénient engendré par la délibération est le ralentissement général apporté à CTS. En effet, le nombre de cycles minimaux que nécessitera le système avant de prendre une action dans le cas où aucun acteur n'ajoute d'information est égal au nombre de cycles délibératifs (2 dans notre cas) plus un cycle où la coalition est acceptée pour un total de 3. Dans le cas où la coalition ne fait réagir aucun des acteurs du système (aucune information ajoutée et le RA ne réagit pas), la coalition existera pour un nombre de cycles égal au nombre de cycles délibératifs plus le nombre de cycles d'attente de déclenchement du RA (dans le cas actuel : 2 + 3 = 5).

Un autre inconvénient est la demande d'envoi de messages entre les autres acteurs et la délibération lors de modifications de la coalition. Cet inconvénient réduit l'indépendance entre les différents acteurs du système par l'ajout de code n'ayant pas de liens avec le traitement effectué par ces acteurs. Lors de prochaines modifications à CTS, il serait peut-être utile d'étudier différents algorithmes de comparaison de graphe pour déterminer la ressemblance entre deux coalitions.

Malgré l'inconvénient principal qu'est le ralentissement du système, l'ajout de la délibération permet l'ajout et l'utilisation d'acteurs externes au traitement des informations de CTS. Parmi les acteurs externes que la délibération permet désormais d'utiliser, on y retrouve l'expert du domaine et les différents modules composant le modèle de l'apprenant (l'État des connaissances et le profil de l'apprenant.) L'ajout de ces acteurs externes permet à CTS de devenir encore plus près d'un système tutoriel intelligent.

La figure 5.5 ci-dessous donne un exemple de fonctionnement de la délibération sur plusieurs cycles.

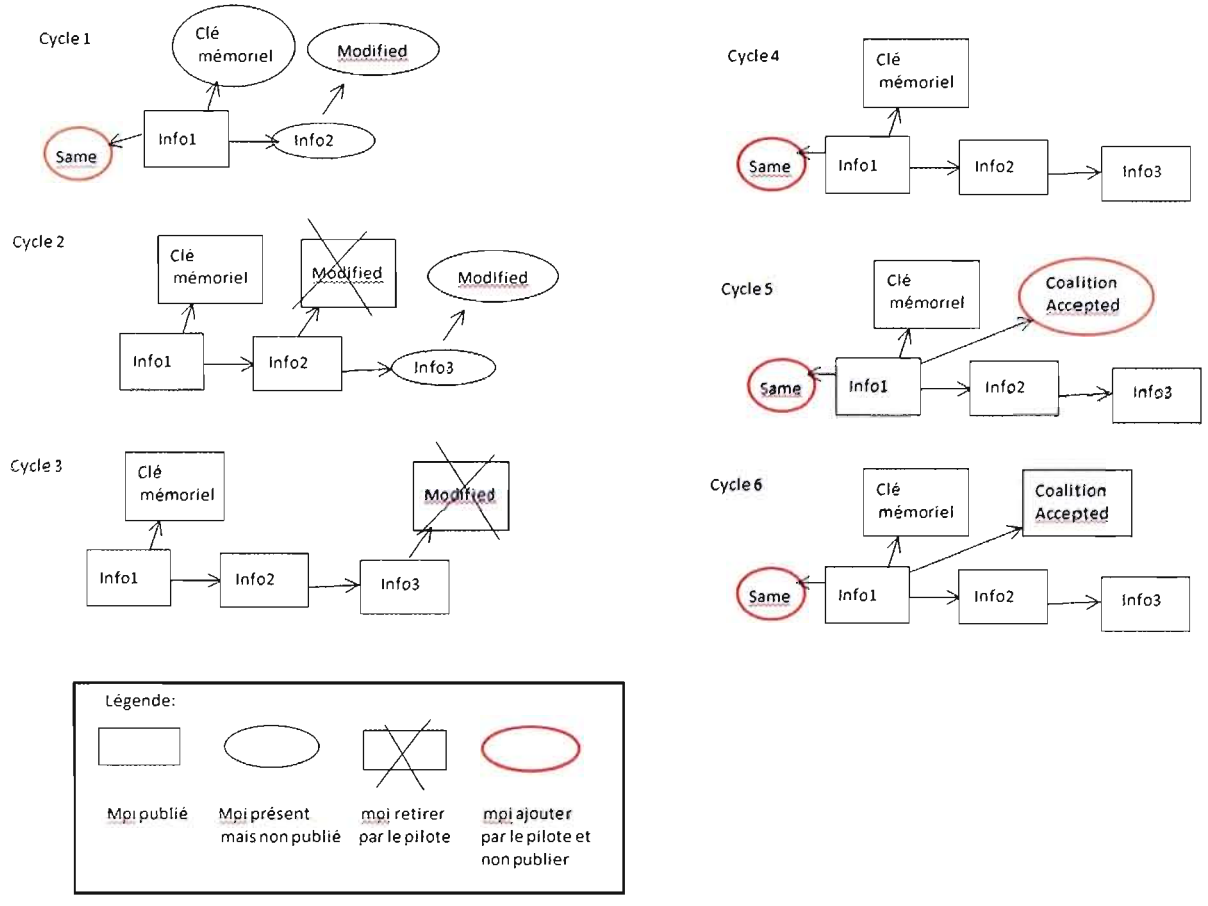



**Figure 5.5** Exemple de fonctionnement de la délibération

Cycle 1 : La coalition publiée contient un seul mpi. La mémoire lui a ajouté une clé mémorielle qui sera visible uniquement au cycle suivant et un acteur a ajouté une information qui sera visible au cycle suivant. Puisque le pilote de délibération n'a pas trouvé de clés mémorielles, elle considère la coalition identique qu'au tour précédent et ajoute le mpi *Same*.

Cycle 2 : Maintenant, les informations ajoutées au cycle précédent sont disponibles. Une troisième information est ajoutée par un acteur externe. Ici, puisqu'il y a une clé mémorielle, on la compare à celle du tour précédent qui n'existait pas et considère que les deux coalitions ont la même clé. Par contre, le pilote détecte le mpi *Modified*, le retire et remet ses compteurs à zéro. Le pilote ne place pas le mpi *Same* puisqu'il a trouvé un mpi *Modified*. Si aucun mpi *Modified* n'avait été détecté ici, le pilote aurait ajouté aussi un mpi *Coalition Accepted*.

Cycle 3 : Toutes les informations ajoutées au cycle précédent sont disponibles. Le pilote compare les clés mémorielles entre le cycle courant et le cycle précédent et les détermine identique. Par contre, le pilote détecte le mpi *Modified*, le retire et remet ses compteurs à zéro. Le pilote ne place pas le mpi *Same* puisqu'il a trouvé un mpi *Modified*.

Cycle 4 : Aucune nouvelle information n'est apparue. Les clés mémorielles sont identiques et aucun mpi *Modified* n'est trouvé. Le pilote ajoute le mpi *Same* et incrémente son compteur.

Cycle 5 : Aucune nouvelle information n'est apparue. Les clés mémorielles sont identiques et aucun mpi *Modified* n'est trouvé. Le pilote ajoute le mpi *Same* et incrémente son compteur qui devient 2 et égal au nombre de cycles délibératifs. Le pilote ajoute maintenant le mpi *Coalition Accepted*.

Cycle 6 : Le mpi *Coalition Accepted* apparait. Les clés mémorielles sont identiques et aucun mpi Modified n'est trouvé. Le pilote ajoute le mpi Same. Il détecte le mpi *Coalition Accepted* et incrémente son compteur d'attente de déclenchement du RA.

À partir du sixième cycle de l'exemple ci-dessus, le RA peut déclencher un acte. Une condition que j'ai ajoutée dans le fonctionnement du Réseau des Actes l'empêche de déclencher un acte sans que le mpi *Coalition Accepted* soit présent. De cette façon, la délibération peut avoir lieu sans être interrompue par le réseau des actes. Aussi, la décision

prise par le Réseau des Actes est beaucoup plus près de la réalité contextuelle puisqu'il est forcé d'attendre d'avoir toutes les informations provenant des différents acteurs « externes ».

## 5.4 Modèle de l'apprenant – Profil de l'apprenant

Le premier module que j'ai simulé est le profil de l'apprenant. Le profil de l'apprenant est la partie du modèle de l'apprenant qui conserve les informations sur les préférences ainsi que les informations personnelles de l'apprenant. Ce module a pour objectif premier de permettre les tests sur la réaction du Réseau des Actes par rapport aux conditions optionnelles et de permettre de tester le fonctionnement de la délibération.

Tout d'abord, ce module ne comporte qu'une seule classe. Cette classe contient le nom, le prénom, le sexe et l'âge de l'étudiant en plus de ses préférences (ce qu'il aime et ce qu'il n'aime pas.) Pour les tests du système, j'ai utilisé quatre préférences qui ont été encapsulées dans une classe énumérative. Chacun des éléments de l'énumération possède un tableau qui contient les couples types valeurs des mpi qui peuvent être affectés par une préférence. Les quatre préférences ont été testées avec l'étape de salutation (plus de détails dans le chapitre sur la planification.)

Une seconde classe énumérative a été créée pour les informations personnelles que ce module peut envoyer. Cette énumération comporte pour chacun de ses éléments un tableau de triplet représentant un couple type valeur plus la valeur d'un mpi *propriété* se rapportant au couple type valeur.

Ces deux énumérations permettent d'ajouter facilement de nouvelles préférences ou de nouvelles informations au système ainsi que d'étendre la liste des mpi affectés par ces informations. En effet, il suffit de créer un nouvel élément de l'énumération et de placer en argument pour le constructeur les couples *type@valeur* séparés par des virgules (« *type1@valeur1* », « *type2@valeur2* ») pour représenter les mpi affectés.

Avec l'aide de ces classes énumératives, l'algorithme utilisé pour ajouter à la coalition les nouvelles informations est devenu très simple. Une boucle existe pour chacun des trois types d'information vérifiée par le profil de l'apprenant soit : ce qu'il aime, ce qu'il n'aime pas et les informations personnelles. Chacune de ces boucles vérifie pour les informations qui font partie de la liste d'informations qui s'y rapporte, s'il existe dans la coalition un mpi correspondant à un couple affecté par la préférence ou par l'information personnelle. Si un tel

mpi existe dans la coalition et que l'information que l'on veut y rattacher n'est pas présente, on rattache à ce mpi un nouveau mpi de valeur représentant la préférence du type représentant s'il aime (*Preference*) ou n'aime pas (*Hates*.) Le fait d'attacher la préférence au mpi qui l'a attiré permet au système de le retirer automatiquement sans en faire la demande explicite lorsque le mpi qui l'a attiré est retiré de la coalition.

L'algorithme pour les informations personnelles est très semblable, seulement il vérifie le mpi propriété attaché au mpi qui attire l'information correspond et lorsque c'est le cas, on y attache un mpi de type *AddInfo* ayant pour valeur l'information personnelle demandée.

Finalement, pour être conforme aux demandes de la délibération, lorsque le module du profil de l'apprenant ajoute une information sur la coalition, il lui ajoute aussi un mpi de type *Modified* ayant comme valeur son nom de module.

## 5.5 Modèle du domaine

Le modèle du domaine est un autre module que j'ai dû simuler. Ce module occupe plusieurs fonctions dans le système. Sa première fonction est de contenir tous les concepts reliés au domaine. Deux classes occupent ce rôle.

La première est *DomainModel* qui contient la logique de peuplement de cette base de connaissance ainsi que de l'extraction d'un concept et possède un *HashTable* qui lie une chaine de caractère à un concept. Cette chaîne de caractère est le nom du concept utilisé à l'interne du système.

La deuxième est *DomainConcept* qui représente les concepts. Cette classe contient une série d'attributs représentant le concept et les liens avec les autres concepts. Ces attributs sont :

1. *Nom* : le nom du concept qui est utilisé à l'interne du système. Ce nom est le même que celui utilisé dans la table du modèle du domaine.

2. *Data1* et *Data2* : Ces attributs représentent les connaissances contenues dans le concept. Souvent, ils contiennent chacun un mot ou une expression qui est synonyme, ou encore une définition.

3. *Type de correspondance* : Cet attribut représente la relation qui lie les *data1* et *data2*. Cette relation peut être associative, c'est-à-dire une relation de synonyme, une définition, auquel cas *data2* est la définition de *data1* ou informationnelle qui représente tout autre lien entre ces données.

4. *Prérequis* et *suite* : Ces deux listes donnent les liens sur l'ordre d'apprentissage des concepts. Les concepts sous *prérequis* doivent être maîtrisés avant ce concept et les concepts sous *suite* représentent les concepts qui peuvent être appris après l'acquisition de ce concept.

5. *Data* et *IsDataOf* : ces deux listes représentent des liens de contenance entre les concepts. Les concepts sous data représentent les sous-concepts du concept père et les concepts sous *IsDataOf* représentent les concepts pères du concept. Un exemple de cette contenance serait le lien entre le concept *Direction* et ses sous-concepts *gauche*, *droite*, *avant*, *arrière*, *haut* et *bas*. Le concept de direction contient une information générale s'y rapportant et les sous-concepts représentent chacune une information spécifique de direction.

6. *Imagepath* : Cet attribut contient le chemin d'accès vers une image aidant à expliquer le concept.

Les fonctions reliées à cette classe sont simplement des fonctions d'accès à l'information. Il n'y a aucune logique de travail reliée à CTS dans ces deux classes. Le lien entre cette base de connaissance et CTS se fait par la classe *DomainModelModule*. Cette classe comporte la logique de traitement pour assurer les fonctionnalités que le modèle de connaissances fournit à CTS. Ces fonctionnalités offertes sont l'évaluation du degré de vérité des réponses fournies par l'étudiant et la capacité de fournir les informations demandées par les autres modules sur des concepts.

La correction des réponses par le modèle du domaine est effectuée de la façon suivante. On vérifie d'abord si la réponse est déjà corrigée, ensuite si la coalition est un texte qui a été compris par le système et finalement si elle contient la clé mémorielle d'une question. Si ce n'est pas corrigé et que la coalition remplit le reste des conditions, on extrait du souvenir de la question la réponse qui était attendue par le système. Ensuite, on compare pour chaque mot le concept fourni par le module de compréhension du langage naturel avec le concept attendu. S'il y a correspondance, on attache à la coalition un mpi indiquant que la réponse est vraie, sinon un mpi indiquant que la réponse est fausse. Dans les deux cas, on ajoute aussi la clé mémorielle de la question répondue.

L'autre fonctionnalité du modèle de domaine est l'ajout d'information. Cet ajout aide surtout pour la sortie texte de CTS vers l'utilisateur. Le fonctionnement est très semblable à l'ajout d'information effectué par le profil de l'apprenant. Une classe énumérative contenant les types d'information pouvant être donnés par le domaine a été créée et tout comme les classes énumératives du profil de l'apprenant, chaque élément contient la liste des mpi sur lesquels le module doit réagir en fournissant l'information en question. Le module recherche dans les coalitions les groupes de mpi reliés aux éléments de l'énumération et y attache lorsqu'il en trouve l'information correspondante.

Dans tous les cas, lorsque le module effectue une modification quelconque sur une coalition, il lui ajoute le mpi *Modified* exigé par la délibération.

Finalement, le modèle du domaine utilisé pour le prototype est disponible à l'annexe A.

## 5.6 Modèle de l'apprenant – Modèle des connaissances de l'apprenant

Le modèle des connaissances de l'apprenant (MCA) est une autre simulation qui est utilisée pour tester d'autres capacités du système. Ce module va servir entre autres à tester la capacité du système à changer de méthode d'enseignement et à choisir les concepts à montrer. Dans cette section, nous allons voir comment la simulation fonctionne, tandis que dans le chapitre sur la planification, nous allons voir comment le système utilise la capacité de ce module pour planifier le dialogue.

Tout d'abord, le MCA possède une structure de données semblable au modèle du domaine. Une table de hachage liant le nom du concept aux statistiques du concept est utilisée. Cette table est placée à l'intérieur de la classe *LearnerKnowledgeModel*. Cette classe contient également une deuxième table qui contient la liste des concepts qui ont été vus par l'étudiant. Bien que l'information puisse être obtenue par vérification de la table principale, cette deuxième table permet de sauver du temps de traitement.

Les statistiques sur chacun des concepts sont conservées dans la classe *LKDomainConcept*. Cette classe possède les mêmes liens entre concepts que pour le modèle du domaine, soit les préalables, les concepts suivants, les concepts contenus (sous-concepts) et les concepts pères. Cette similitude permet au MCA de se promener dans le graphe des connaissances sans avoir à questionner le modèle du domaine pour obtenir ces relations entre les concepts. Les statistiques que ce module recueille sont les suivantes :

1. *Nombre de bonnes réponses* et le *nombre de réponses totales* : Le module recueille le nombre de fois que l'étudiant a répondu à une question portant sur le concept et le nombre de fois que les questions ont reçu une bonne réponse. Ces données servent au calcul de la note sur la maîtrise du concept.

2. *Note* : Cette donnée sert à déterminer le niveau de maîtrise d'un concept. Cette valeur est calculée par la formule suivante :

$$N(c) = \frac{\frac{Nbr}{NrT} + \sum_j^0 N(c_j)}{j+1}$$

Où:

| | |
|---|---|
| N(c) | Note du concept |
| Nbr | Nombre de bonnes réponses |
| NrT | Nombre de réponses Totales |
| j | Nombre de sous-concepts |
| N(cj) | Note du sous-concept j |

La fonction calculant cette note demande ensuite à tous les concepts pères de recalculer leur note.

3. *Acceptabilité du niveau de compréhension* : Ceci indique si le concept est suffisamment bien compris pour permettre de passer à l'apprentissage d'autres concepts. Ceci est déterminé si la note du concept est supérieure à un certain seuil de compréhension. En ce moment, le seuil utilisé est de 80 %.

Le modèle de connaissance de l'apprenant choisit, lorsque ce lui est demandé, la liste des prochains concepts à montrer. Il y a trois façons de choisir les concepts. Les deux premières ont un fonctionnement semblable. La première est d'envoyer uniquement un concept de tête ou concept père. La deuxième est d'envoyer un concept de tête avec la liste de ses concepts constituants. Tout d'abord, on regarde dans la liste des concepts vus si un de ces concepts a un niveau de compréhension non acceptable. Si c'est le cas, on prend ce concept et on le retourne avec ses concepts constituants. Sinon, le système doit choisir un nouveau concept à choisir. On retire d'abord de la liste des concepts tous les concepts qui ont déjà été vus. Puis, pour chaque concept restant, on vérifie si le concept est un concept de tête. Si c'est le cas, on vérifie si ses préalables sont tous compris. Les préalables d'un concept correspondent à ses préalables plus les préalables de tous ses concepts constituants. Lorsqu'un concept répond à ces conditions, il est choisi pour être transmis. La différence entre la première façon et la deuxième est que la première n'envoie que le concept de tête tandis que la deuxième envoie le concept de tête avec ses concepts constituants. La troisième façon est de choisir pour un concept fourni par le système son concept de tête ainsi de ses concepts constituants. Dans tous les cas, les concepts choisis sont ajoutés dans la liste des concepts vus.

La classe faisant le lien entre modèle de connaissance de l'apprenant et CTS tient un compte du nombre de mauvaises réponses consécutives que l'étudiant fait. Lorsque ce compte dépasse un certain seuil (qui est de 2 pour l'instant), un mpi est attaché au concept en question qui indique que l'étudiant comprend mal ce concept.

Il est important de noter ici que les 3 derniers modules décrits ici ne sont que des simulations de ce qui devrait être fait. Il est certain que ces modules en réalité devraient être beaucoup plus complexes et offrir plus de fonctionnalités que ce qu'ils offrent en ce moment. Par contre, les simulations faites ici sont suffisantes pour tester le planificateur et peuvent être

augmentées selon les besoins des autres développeurs de CTS. Le prochain chapitre décrira les modules qui sont plus spécifiquement utilisés dans la planification du dialogue.

# CHAPITRE VI

# LA PLANIFICATION

La planification est un travail coopératif parmi toutes les différentes parties de CTS. Dans cette section, nous allons voir les parties qui traitent plus spécifiquement de la planification du dialogue et qui servent au traitement du langage naturel.

## 6.1 Plan générique

Les plans génériques représentent des séries d'étapes à effectuer dans le cadre d'un dialogue. La description ici représente la liste des catégories d'actions à effectuer dans un dialogue sans pour autant donner la méthode à suivre pour effectuer la tâche. Un exemple de plan générique serait un plan de session qui possède les étapes de début de session, milieu de session et fin de session. Nous avons donc ici la liste des étapes à effectuer pour faire une session de travail, mais pas l'information sur comment chacune de ces étapes doit être exécutée. Une étape peut alors être raffinée soit par un autre plan générique ou par une méthode.

Dans CTS, les plans génériques sont stockés dans le réseau des actes, au niveau décisionnel. Ceci permet au système de choisir une marche à suivre selon le contexte avec les mêmes mécanismes qu'il utilise pour choisir les actions à poser. Ce qui différencie principalement un acte au niveau décisionnel et un acte au niveau des plans génériques est le type de Microprocessus utilisé. Tel qu'indiqué à la section 5.2, un acte au niveau décisionnel utilise un Microprocessus de type *ActProposition*. Un acte contenant un plan générique utilise un Microprocessus de type *GenericPlanCodelet*. Un acte du plan générique contient toutes les étapes consécutives sans alternatives et possédant les mêmes attributs d'une marche à suivre. Lorsque des étapes alternatives se présentent, on doit créer de nouveaux actes

représentant les alternatives dont le système pourra choisir en fonction du contexte. La figure 6.1 montre un exemple d'un plan générique au niveau du réseau des actes.

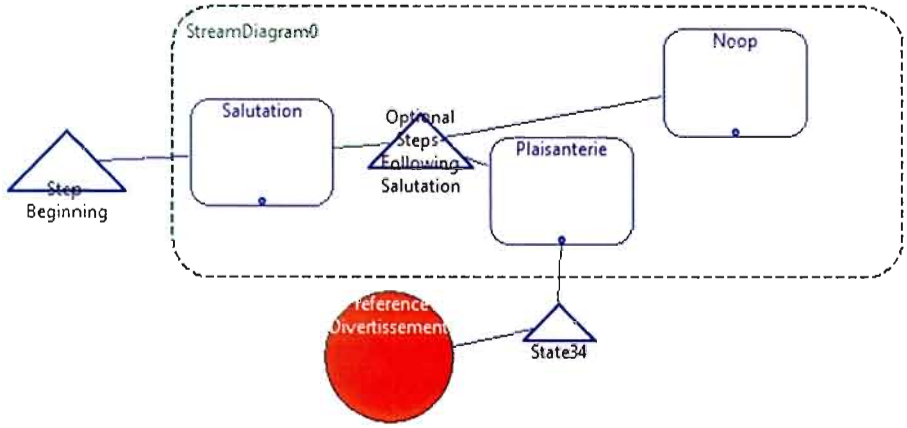



**Figure 6.1** Plan générique d'un début de session

Dans cet exemple, on peut voir qu'un début de session commence tout d'abord par une salutation, puis il peut continuer ou non par une plaisanterie. Selon la préférence de l'utilisateur face aux plaisanteries, le choix de l'étape plaisanterie pourra être favorisé ou défavorisé par l'État optionnel de la préférence sur le divertissement. L'acte *Noop* est un acte spécial qui ne fournit aucune étape supplémentaire au système. Il représente l'optionalité de l'acte de la plaisanterie.

La construction de la marche à suivre est effectuée par des sous-classes spécifiques de la classe abstraite *GenericPlanCodelet*. Cette classe abstraite contient toute la logique nécessaire à la construction des listes de mpi contenant toutes les informations nécessaires à l'instanciation du plan par le planificateur. Cette construction est effectuée en quatre étapes. La première étape consiste à construire une chaîne de mpi commençant par un mpi *New Step-Start* auquel on attache toutes les étapes de la marche à suivre qui sont contenues dans sa liste d'étapes. Les autres étapes de la construction de la marche à suivre sont présentes uniquement lorsque l'élément en question est présent. On attache ensuite à cette chaîne un mpi indiquant que la marche à suivre nécessite des étapes supplémentaires qui sont soit optionnelles ou requises. Ensuite, si des arguments sont disponibles dans la liste des arguments, on attache à la suite de la chaîne un mpi de type *Arg* et de valeur *Start* qui indique le début de la liste des arguments. Le système attache ensuite en chaîne une liste de mpi

contenant ces arguments servant à instancier le plan générique. Le tableau 6.1 contient la liste des mpi réservés aux plans génériques. Finalement, on place à la suite de la chaîne les informations supplémentaires fournies par la coalition à l'intention des plans génériques.

| Type du mpi | Valeur du mpi | Fonction du mpi |
|---|---|---|
| New Step | Start | Indique le début de la chaîne d'étape |
| New Step | Step-*Nom de l'étape* | Représente une étape de la marche à suivre |
| Additionel Step | Optional | Indique que le plan envoyé possède des étapes supplémentaires qui sont optionnelles |
| Additionel Step | Required | Indique que le plan envoyé possède des étapes supplémentaires qui sont requises |
| Arg | Start | Indique le début de la chaîne des arguments |
| Looping | Sequentiel | Indique que ce plan devra être répété plusieurs fois en cyclant une étape à la fois: ex.: s1;s1;s2;s2;s3;s3; |
| Looping | Imbrique | Indique que ce plan devra être répété plusieurs fois en l'insérant complètement avant de cycler: ex.: s1;s2;s3;s1;s2;s3 |
| StepVariable | Concepts | Indique que les étapes se rapportent à un concept en particulier (aussi les concepts en question s'y seront attachés) |
| PGConcepts | Needed | Indique que le plan nécessite de concepts pour s'instancier |
| PGConcepts | HB | Indique qu'il faut envoyer les concepts constituants avec le concept choisi |
| PGConcepts | H | Indique qu'on doit envoyer que le concept de tête |
| PGConcepts | Received | Indique que le plan contient le concept nécessaire (représente l'ajout des concepts par un acteur externe: cet acteur change le PGConcepts-Needed en PGConcepts-Received) |
| OtherPGInfo | *Valeur* | Contient les informations supplémentaires fournies au plan générique |

**Tableau 6.1** Liste des principaux mpi relatifs aux plans génériques

Chaque acte contient un Microprocessus qui sous-classe *GenericPlanCodelet*. Ces Microprocessus ne font que peupler les listes d'étapes et d'arguments de la marche à suivre représentées par l'acte. La figure 6.2 donne un exemple de la chaîne de mpi produit pour le plan générique *Enseigner un Concept*. On donnera plus de détails sur les interactions entre les différentes parties du système dans le chapitre sur l'expérimentation.

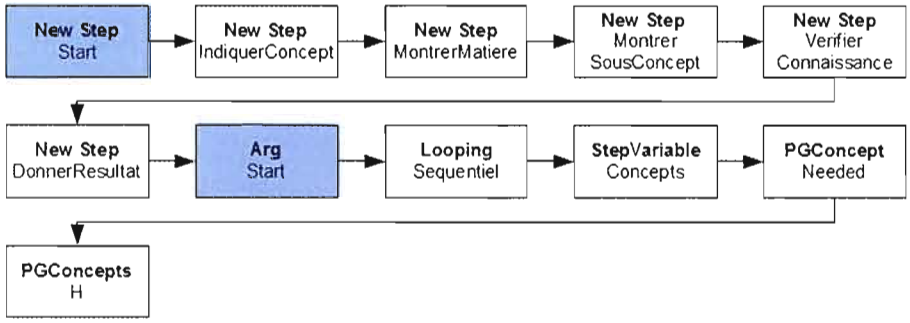



**Figure 6.2** Liste de mpi produits par le plan générique Enseigner un Concept

## 6.2 Le planificateur

Le module de planification utilise une mémoire spécialisée qui permet au système de stocker les plans du dialogue ainsi que la logique de traitement pour l'ajout, la modification et l'abandon d'étapes de plan. La figure 6.3 donne la structure du planificateur.

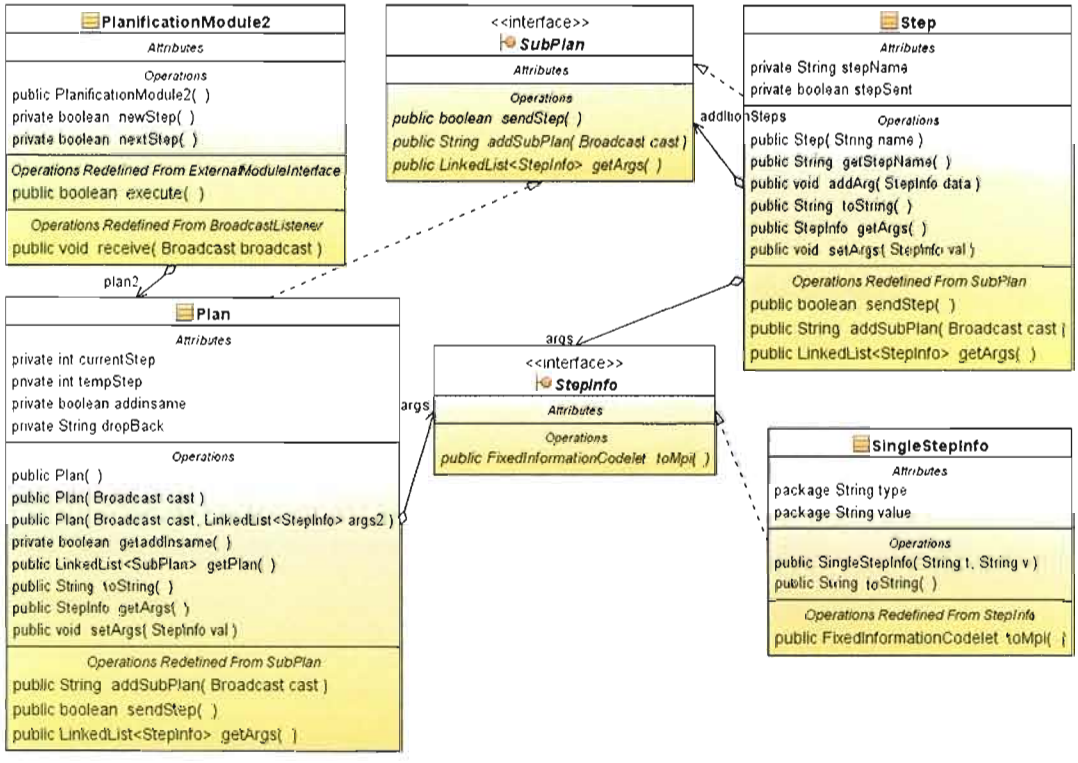



**Figure 6.3** Diagramme de classe du Planificateur

La classe *PlanificationModule2* sert d'interface entre le Planificateur et CTS. Il observe les coalitions publiées à la recherche de deux mpi spécifiques soit un mpi *Next Step* ou un mpi *New Step-Start*. Les plans sont conservés dans une liste de la classe *Plan* qui possède l'interface *SubPlan*. Cette liste contient des éléments du type *SubPlan*. Une étape spécifique est contenue dans la classe Step qui possède elle aussi l'interface *SubPlan* et qui peut contenir une liste d'étapes qui précise la marche à suivre pour effectuer l'étape. La figure 6.4 montre l'aspect de la structure de données obtenu lors de la conservation de plan.

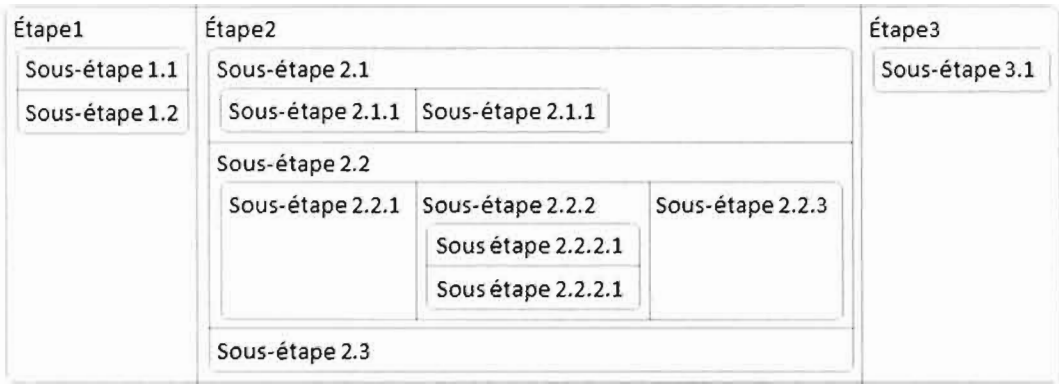



**Figure 6.4** Structure de données d'un plan de dialogue

Dans la figure 6.4, les tables représentent des éléments conservés dans la classe *Plan* et les cellules à des éléments conservés dans la classe *Step*.

Cette mémoire a trois fonctions principales : l'ajout d'un plan en mémoire, l'envoi de l'étape suivante et l'abandon d'un plan

**L'ajout d'un plan en mémoire** se fait en plusieurs étapes. La première étape consiste à instancier le plan à ajouter. Cet ajout est fait suivant les étapes suivantes :

1. Créer la liste des étapes à ajouter. Le module recherche d'abord dans la coalition le mpi *New Step-Start*. Ensuite, il suit la liste des mpi en transformant chacun des mpi de type New Step (excluant le *New Step-Start*) en *Step* qu'il ajoute ensuite dans une liste chaînée. Cette transformation est effectuée en suivant l'ordre des mpi dans la chaîne de mpi.

2. Créer la liste des informations additionnelles. Ici, le module crée une liste chaînée qui contiendra toutes les informations spécifiant les paramètres d'exécution des étapes du plan instancié. Le système recherche tous les mpi dans la coalition qui sont de type *OtherPGInfo*. Ensuite, pour chacun de ces mpi, on encapsule leur valeur dans un élément de classe *StepInfo* que l'on place dans la liste chaînée.

3. Si l'information est disponible, on indique le type d'abandon de plan que l'on effectuera si c'est nécessaire. Si l'information n'est pas là, le type d'abandon est *This*. Plus d'information sur l'abandon des plans un peu plus bas.

4. Maintenant, le système effectue l'instanciation du plan en utilisant la liste des arguments fournie s'il y en a une présente. Tout d'abord, on extrait les variables d'étape de la coalition que l'on place dans une liste. Dans le prototype, il n'y a que des concepts qui sont utilisés comme variable d'étape. Ensuite, le module combine la liste des étapes et la liste de variables suivant le mode de boucle donnée.

    a. Séquentiel : on applique pour chaque étape toutes les valeurs de la variable avant de passer à l'étape suivante. Exemple : pour une liste d'étapes e1, e2 et une liste de valeurs v1, v2. Ce type de boucle va donner le plan instancié suivant : e1v1, e1v2, e2v1, e2v2.

    b. Imbriquer : on applique ici pour chaque valeur de la variable toutes les étapes avant de passer à la valeur suivante. Exemple : pour une liste d'étapes e1, e2 et une liste de valeurs v1, v2. Ce type de boucle va donner le plan instancié suivant : e1v1, e2v1, e1v2, e2v2.

5. Finalement, on combine le plan instancié avec la liste des informations additionnelles. Ici, on ajoute chaque élément de la liste des informations additionnelles à chaque étape du plan instancié.

Maintenant que le plan est instancié, il ne reste plus qu'à l'ajouter au plan en mémoire. Cet ajout peut être effectué dans deux circonstances. Le premier type d'ajout est l'ajout d'un plan comme plan précisant la marche à suivre d'une étape. Ce type de plan s'ajoute à une étape qui n'a pas de plan précisant. Tout d'abord, on va à l'étape en cours à l'intérieur de chacun des plans et on vérifie si cette étape possède un plan précisant sa marche à suivre. Si l'étape en possède un, on répète la recherche. Lorsque le module trouve une étape qui n'a pas de plan qui lui est affecté, il ajoute alors le plan instancié à cette étape.

Le deuxième type d'ajout est pour l'ajout d'une continuation de plan. Ce cas arrive lorsque le plan qui est envoyé est incomplet soit parce qu'il y a un changement d'arguments entre différentes étapes ou parce qu'il y avait un chemin alternatif entre deux types d'étape. Le module recherche alors un plan qui possède la mention *addinsame* (ajouter dans le même plan). Cette mention apparait lorsqu'une coalition d'un plan possède un mpi *Additional Step*. La recherche est effectuée en vérifiant si le plan actuel possède cette mention, sinon on regarde le plan de l'étape en cours pour vérifier l'existence de cette mention.

Ces vérifications pour l'ajout sont effectuées en même temps retirant ainsi la nécessité au système de se rappeler si le plan obtenu appartenait à un plan en cours d'obtention ou à une étape demandant une précision sur la marche à suivre.

**L'envoi de l'étape suivante** se fait lorsque le système le demande. Cette fonction est réalisée par une simple fouille récursive. On obtient d'abord l'étape en cours, on vérifie si elle a été transmise et si elle a été transmise, et on vérifie si elle possède un plan de raffinement. Si elle en possède un, on répète avec le nouveau plan, jusqu'à ce qu'on trouve une étape qui n'a pas été transmise ou qui n'a pas de plan de raffinement. Dans le cas d'une étape qui n'a pas été transmise, on transmet cette étape et on indique qu'elle est transmise. S'il n'y a plus de plan de raffinement, on va à l'étape suivante du plan et on la transmet en indiquant qu'elle est transmise si elle existe. Si elle n'existe pas, on remonte d'un niveau et on répète le processus. Le module envoie un mpi de fin de plan lorsque l'envoi d'une étape échoue en atteignant la fin du plan principal.

**L'abandon d'un plan** s'effectue de la façon suivante. Un plan possède un des deux types d'abandon suivant : *This* ou *SameAsLast*. *SameAsLast* indique que le plan en cours fait

partie intégrante du plan supérieur dans la hiérarchie de plan. Le système trouve d'abord l'étape en cours, puis vérifie le type d'abandon que son plan possède. S'il est de type *SameAsLast*, on remonte d'un niveau, allant dans l'étape qui se retrouve raffinée par ce plan. On répète cette recherche jusqu'à ce qu'on trouve un plan ayant un type d'abandon *This*. Lorsque l'on trouve ce type de plan, on abandonne l'étape en cours et on passe à l'étape suivante. Si on a à ajouter un plan ou une étape de remplacement, on l'insère entre l'étape à abandonner et l'étape qui la suit, puis on envoie l'étape suivante.

## 6.3 Compréhension du langage naturel

La compréhension du langage naturel que j'ai installée dans CTS est très de base. Tout d'abord, il faut noter que les questions posées par CTS sont formulées de façon à demander des réponses à mot (concept) unique. Aussi, le module ne permet pas la correction d'erreurs orthographiques. Le rôle de ce module est de permettre de tester les interactions entre l'utilisateur et CTS. Deux sortes d'interactions sont prévues pour les tests, la première est lorsque l'étudiant répond à une question de CTS et la deuxième est lorsque l'étudiant pose une question de forme précise à CTS.

La compréhension du langage naturel se fait lors de la perception du texte entré par l'utilisateur durant la création des différents mpi résultant de cette perception, mais avant l'envoie des mpi en mémoire de travail.

L'analyse se fait en deux étapes. La première étape est pour déterminer si l'entrée de l'utilisateur est une question. Le système peut reconnaitre pour l'instant qu'une seule forme de question soit la question « Qu'est-ce que *concept*? ». La détection de la question est très simple. La première étape consiste à séparer la phrase en mot. Ensuite, on vérifie si le premier mot est « Qu'est-ce ». Si c'est le cas, on crée une nouvelle chaîne de caractère avec tous les mots suivants le que. Ensuite, on retire le point d'interrogation et les espaces superflus pour obtenir le mot ou expression sur lequel l'utilisateur veut obtenir de l'information. Finalement, on ajoute au mpi contenant la question un mpi de type *User-Text-Form* et de valeur *Question*. Pour exemple, utilisons la question « Qu'est-ce que nombre de direction? ». Le système sépare la phrase en mots ce qui donne les chaînes « Qu'est-ce », « que », « nombre », « de » et « directions? ». Le système reconnait « Qu'est-ce » dans la

première chaîne. Le système crée alors une nouvelle chaîne en ignorant le mot que ce qui donne la chaine « nombre de directions? » Maintenant, on retire le point d'interrogation, ce qui résulte en la chaîne « nombre de directions ». C'est cette chaîne qui sera utilisée pour le reste du traitement de compréhension du langage naturel.

La deuxième étape consiste à déterminer les concepts représentés par l'expression envoyée par l'utilisateur. L'expression utilisée pour cette analyse est soit l'expression intégrale si ce n'était pas la question décrite plus haut, ou le résultat du traitement décrit plus ci-dessus si c'était la question que le système peut traiter.

Premièrement, on vérifie pour chaque concept présent dans le modèle du domaine si l'expression correspond soit à la donnée 1 ou à la donnée 2 du concept. Cette vérification est effectuée par une boucle sur tous les concepts présents dans le modèle du domaine. Une autre solution aurait été de placer dans une table de hachage chacune des expressions présentes dans les concepts du modèle de domaine couplée au concept et à sa donnée correspondante. La première solution a été retenue puisque le modèle de domaine dans le cas présent est relativement petit et que les interactions avec utilisateur dans le prototype sont plutôt infréquentes. Pour chaque correspondance, on place dans une liste le nom du concept et le numéro de la donnée qui a correspondu. Ensuite, pour chaque correspondance qu'il y a eu, on attache au mpi contenant l'expression un mpi de type *Concept* et de valeur le nom du concept couplé à un mpi de type *Concept_Property* et de valeur *Data1* ou *Data2* selon la donnée correspondante. Aussi, on attache à chacun de ces mpi *Concept* qui sont des sous-concepts un mpi de type *Concept_Parent* avec comme valeur le nom du concept parent.

Finalement, si le module a effectué un ajout de mpi, il ajoute aussi un mpi de type *NLU* et de valeur *Understood* sinon la valeur de ce mpi sera *Not Understood.*

Il est important de se rappeler que le but du présent mémoire est l'ajout d'un planificateur de dialogue au système CTS et non l'ajout d'un système de traitement du langage naturel. Il est vrai que le module de compréhension du langage naturel est simpliste dans sa conception, mais il ne faut pas oublier que son rôle se limite à permettre de faire des tests sur les interactions entre l'utilisateur et CTS et de rendre compréhensibles les entrées de l'utilisateur pour CTS. Aussi, CTS est un système coopératif où le résultat final de tout

traitement provient de la coopération des différents modules présents dans CTS. Le traitement des questions et des réponses reçu par CTS sera vu dans le chapitre sur le scénario.

## 6.4 Génération du langage naturel

Le module de génération du langage naturel sert à former le texte qui sera envoyé par CTS à l'utilisateur. La fonction du module que j'ai créée est de vérifier le bon fonctionnement du planificateur en fournissant la capacité d'avoir des sorties en langage naturel qui représentent bien l'action que CTS tente de poser ainsi que le concept à exposer. Ce texte est développé en deux étapes. La première étape consiste à traduire les chaines mpi reçues sous une forme plus propice au traitement à effectuer. La deuxième étape est la traduction de cette nouvelle forme en un texte compréhensible pour l'humain.

Tout d'abord, il faut noter que le module de génération du langage naturel possède deux interfaces, soit l'interface *NLGModuleReceiver* qui sert de porte d'entrée pour les informations provenant de CTS et de l'interface *NLGModuleSender* qui sert de porte de sortie pour les phrases finies vers l'interface utilisateur. Ces deux interfaces permettront de pouvoir changer le module de génération de texte vers des modules utilisant des algorithmes plus puissants. Le livre « Building Natural Language Generation Systems » (Reiter et Dale, 2006) donne de très bonnes bases pour construire de tels modules, mais dont l'implémentation dépasse la portée du présent mémoire.

La classe *VerySimpleBridge* sert de point de sortie pour CTS. Elle possède à la fois les interfaces d'entrée et de sortie pour le module de génération de texte. Elle peut recevoir deux types de données différentes de CTS. Le premier type est une simple chaîne de caractères. Dans ce cas, le module ne fait que transférer la chaîne de caractère vers la fenêtre d'affichage. Ce type de données est surtout utilisé pour afficher des messages d'erreur de la part de CTS et devrait être utilisé uniquement durant la phase de développement du système. Le deuxième type de données est un arbre de mpi sous forme de coalition. Le module de génération de texte est appelé lors de la réception de ce type de données.

Lorsque le module reçoit un arbre mpi, sa première tâche est de décomposer cet arbre sous une forme qui est plus facile à travailler pour le module. Cette extraction a deux fonctions. La première est d'obtenir les informations relatives à la phrase à créer et la deuxième est d'obtenir les concepts et leur ordre d'apparition dans la phrase. Les informations relatives à la phrase correspondent au but communicatif recherché, à la forme de la phrase et au mode de livraison. Le tableau 6.2 donne la liste des différents éléments présents dans chacune de ces catégories. Ceux qui sont utilisés dans le prototype sont en italique dans le tableau.

| **Buts communicatifs** | **Mode de livraison** | **Forme de phrase** |
|---|---|---|
| | | |
| ***Get The Attention*** | ***Monologue*** | ***Fragment*** |
| ***Explanation*** | Directed Line Of Reasoning | ***Declarative*** |
| Conversational_Repair | ***Hinting*** | ***Interrogative*** |
| ***Acknowledgement*** | ***Rephrasing*** | Exclamative |
| ***Probing The Student Inference Process*** | Analogy | ***Imperative*** |
| Instruction In The Rules Of Game | | |
| ***Help In response To Pause*** | | |
| Teaching The Problem Solving Algorithm | | |
| Brushing Off | | |
| ***Teaching The Sublanguage*** | | |

**Tableau 6.2** Informations relatives aux phrases

J'ai tiré ces catégories du système CIRCSIM-Tutor (Kim et al. 2006), car je trouvais que ça reflétait bien les différentes dimensions d'une phrase. Malheureusement, au cours de l'implémentation du scénario, j'ai remarqué que les buts communicatifs décrits dans CIRSCIM-Tutor ne correspondaient pas parfaitement aux buts tutoriels de notre système. Dans le prototype, j'ai conservé les buts communicatifs de CIRCSIM, car j'ai remarqué cette différence tard dans le développement et le changement aurait demandé des modifications trop importantes au Réseau des Actes pour le temps qu'il me restait. L'utilisation de ces buts a entrainé une légère baisse sur le niveau d'expression disponible par le système, mais le niveau d'expression reste acceptable pour le prototype.

Le transfert de ces propriétés de phrase est fait par une simple recherche dans la coalition pour les mpi de type *Concept_Graphe_Property*. On extrait pour chacun de ces mpi le type de propriété et sa valeur que l'on place dans la propriété correspondante.

L'étape qui suit est l'extraction des concepts. Les concepts sont stockés de deux façons dans la structure de données : avec une liste chaînée et dans une table de hachage. La liste chaînée sert à conserver l'ordre des concepts pour la création de phrases, tandis que la table de hachage sert à retrouver un concept précis dans la liste. Chaque concept est conservé dans une instance de la classe *ConceptElement* qui enregistre les informations suivantes :

1. **Concept** : Contient le nom du concept

2. **Propriété** : Liste contenant les propriétés précisant le concept. Exemple : le mot bonjour serait de concept salutation et de propriété début.

3. **Informations additionnelles** : Table de hachage qui sert à contenir des informations provenant de CTS pour aider à la génération du texte. Comme exemples : Le concept *nom* de propriété *utilisateur* possède comme information additionnelle le nom de l'utilisateur.

L'instance possède également un lien vers les concepts suivants et le concept précédent. La figure 6.5 donne un exemple de coalition de mpi et la figure 6.6 donne le résultat de sa transformation vers la structure de données. Les liens dans la structure de données entre les concepts à l'intérieur des concepts sont omis pour une simplification graphique.

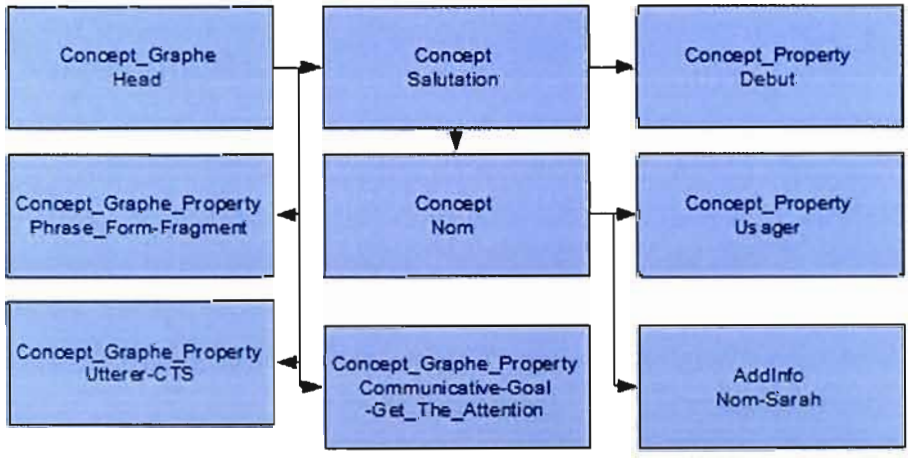



**Figure 6.5** Coalition pour une salutation avec le nom

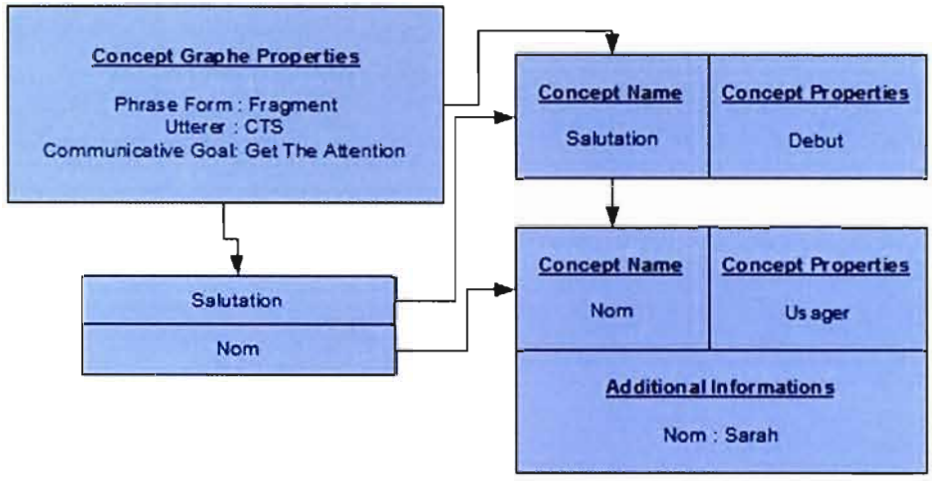



**Figure 6.6** Résultat de la transformation pour le générateur de texte

La transformation de la structure de données en texte est relativement simple. Tout d'abord, à l'aide d'une série de switch, on se rend à la section de code qui traite le graphe de concepts selon ces propriétés, soit le but communicatif, le mode de livraison et la forme de la phrase. Ensuite, le module de génération de texte effectue les traitements propres à la combinaison des propriétés pour générer un texte qui nous est compréhensible. Dans l'exemple de la figure 6.6, le système sait qu'il doit obtenir l'attention en utilisant un fragment. Il doit donc faire un fragment de forme *Salutation-Debut Nom-Usager*. Le système a le choix entre le mot bonjour pour une salutation plus formelle ou les mots bonjour et salut pour une salutation moins formelle. Dans le cas de la salutation, une règle indique qu'elle doit être formelle si le concept titre civil est présent. Dans le cas présent, ce concept est absent. Le système a donc le choix entre bonjour et salut. Le résultat final pourra donc être Bonjour Sarah ou Salut Sarah. Dans le cas présent, le système choisit aléatoirement laquelle de ces deux possibilités sera utilisées.

Le résultat final de la planification avec ce type de générateur de texte donne une machine à États finis. Il serait possible d'augmenter le niveau d'expression du système en enrichissant les différentes bases de connaissance présentes dans CTS.

## 6.5 Module de formes linguistiques

Ce module externe sert de base de connaissance spécialisée. Il reprend tous les concepts présents dans le modèle du domaine plus quelques autres concepts comme les valeurs de vérité et la salutation et leur associent pour chaque but tutoriel présent dans le système les triplets but communicatif-mode de livraison-forme de la phrase qui leur sont possibles. Son existence est due à un désir de conserver dans le modèle du domaine uniquement les informations propres aux connaissances du domaine. À moyen ou long terme, le module de forme linguistique pourrait se voir combiner avec un module pédagogique.

Le module comporte une table de hachage pour permettre un accès rapide à chacun des concepts du système. Chacun des concepts est conservé dans une instance de la classe *LConcept*. Cette classe possède trois tables de hachage imbriquées pour conserver les possibilités. Au premier niveau, la clé utilisée est le but tutoriel recherché. Le résultat donne une deuxième table de hachage qui utilise les buts communicatifs comme clé. La correspondance pour le but communicatif est une troisième table de hachage qui utilise le mode de livraison comme clé. Finalement, la valeur correspondant à une clé dans la troisième table est une liste de forme de phrase. Le tableau 6.3 donne un exemple de la structure.

| | | |
|---|---|---|
| Expliquer | Explanation | Monologue [Declarative] |
| | | Rephrasing [Declarative] |
| | Teaching the sublanguage | Monologue [Declarative] |
| Questionner | Probing The Student Inference Process | Monologue [Interrogative] |
| Aide à une pause | Help In response To Pause | Rephrasing [Interrogative] |
| | | Hinting [Declarative, Fragment, Interrogative] |
| | Get The Attention | Monologue [Declarative, Fragment, Imperative] |

**Tableau 6.3** Exemple de la structure pour un élément de la base de connaissance

Le module fournit de l'information lorsqu'il trouve dans la coalition un mpi de type *InformationGathering* et ayant comme valeur soit *CommunicativeGoal*, *DeliveryMode* ou *PhraseForm*. Si le mpi est présent, il vérifie ensuite si l'information n'a pas déjà été fournie

en regardant l'existence de mpi de type *PossibleBC*, *PossibleDM* ou *PossiblePF*. Si le mpi n'est pas présent, un mpi *PossibleXX* (où *XX* représente le type d'information recherché) est créé pour chacune des possibilités et est attaché au mpi *InformationGathering*. L'obtention des possibilités est simple, si on recherche les buts communicatifs, on obtient la liste des clés de la table sous le but tutoriel. Pour les modes de livraison, on obtient la liste des clés de la table sous le but communicatif choisie du but tutoriel. Les formes de la phrase sont obtenues en allant chercher la liste liée au mode de livraison choisi sous le but communicatif choisi du but tutoriel.

Maintenant que les fonctionnalités des différents modules ajoutés ont été décrites, le prochain chapitre donnera des exemples de fonctionnement et démontrera la collaboration entre ces différents modules.

# CHAPITRE VII

# EXPÉRIMENTATIONS

Nous verrons dans ce chapitre le scénario utilisé pour développer et tester le système de planification. Nous allons voir pour chacun des sous-scénarios utilisés sa description, les éléments qu'il cherche à tester et un exemple de fonctionnement. L'exemple de fonctionnement servira à démontrer les interactions entre les différents composants et modules du système. Ayant déjà décrit plusieurs des modules en détail dans les chapitres précédents, la description du fonctionnement sera généralement limitée au niveau des communications intermodules. Par ailleurs, pour simplifier la compréhension, lorsqu'il y a un résultat ou une action à la suite d'un mpi, ce résultat ou action a lieu après sa publication et après la délibération, sauf sur avis contraire.

## 7.1 Profil des étudiants

Trois profils d'étudiants ont été utilisés pour effectuer les tests dans le scénario. Ils sont principalement utilisés dans le sous-scénario de la salutation. Les différents profils ont servi principalement à tester le fonctionnement des États optionnels et pour fournir les informations additionnelles pour le générateur de texte. Les trois profils sont :

1. Prénom : Pierre-Jean
   Nom : Jacques
   Sexe : Homme
   Age : 35
   Aime : (Aucun choix)
   N'aime pas : Formel

2. Prénom : Claire
   Nom : Brossard
   Sexe : Femme
   Age : 20
   Aime : Plaisanterie, Formel
   N'aime pas : Nommer, Rien

3. Prénom : Juliette
   Nom : Lapinot
   Sexe : Femme
   Age : 49
   Aime : Formel, Nommer
   N'aime pas : Plaisanterie, Rien

Ces profils sont contenus dans le module du profil de l'étudiant et lors de son initialisation, il choisit aléatoirement un de ces trois profils. Une fois le profil choisit, il n'est pas possible de le changer en cours d'exécution.

## 7.2 Salutation

### 7.2.1 Description

Le premier scénario utilisé est celui de la salutation. Ce scénario consiste à choisir en premier lieu un plan de session (il n'y en a qu'un dans le prototype) et ensuite de procéder au raffinement de la première étape, soit effectuer une salutation.

### 7.2.2 Buts recherchés et composants testés

Bien que le scénario ait l'air simple, son but premier était de tester le fonctionnement de plusieurs modules et de leurs interactions. Plus précisément, ce scénario a permis de vérifier le bon fonctionnement de la délibération, du Réseau des Actes à trois tiers, du clignotement des États, du module de simulation du profil de l'apprenant, de la mémoire épisodique à court terme, du module de planification et du module de génération de texte.

### 7.2.3 Exemple de fonctionnement

La première étape lorsque le système démarre est l'initialisation. À ce stade, les seules actions d'intérêts sont le choix du profil de l'apprenant et la création du Microprocessus *Session Started.* Ce Microprocessus allume l'État *Session Started* du réseau des actes. La fonction de clignotement des États éteint l'État *Session Started* du Réseau des Actes et qui est rallumé à chaque cycle où le Microprocessus est publié. Après quelques cycles, un acte de démarrage se déclenche. Au cycle suivant, puisque le Microprocessus *Session Started* n'est pas publié, l'État correspondant reste éteint, prévenant ainsi le redémarrage de l'acte de démarrage. Cet acte de démarrage est le seul acte du réseau qui ne possède pas de couple acte RAD/EAO. La fonction de l'acte est de démarrer des Microprocessus asynchrones qui simulent des traitements, mais qui s'exécutent dans la phase de publications et à lancer aussi des mpi lorsque nécessaire. Pour le prototype et son scénario, l'acte de démarrage ne lance qu'un mpi *Next Step* et aucun acte asynchrone. Le module du planificateur agit à la réception de ce mpi après que la délibération soit terminée et qu'un mpi *Coalition Accepted* ait été ajouté.

À la réception du mpi *Next Step*, le planificateur fait une recherche dans son plan pour trouver l'étape suivante. Lorsqu'il découvre qu'il n'a pas de plan, il envoie un mpi *Session Plan Needed.* Ce mpi active un État correspondant dans le Réseau des Actes dans la section des plans génériques. Un acte se retrouve ensuite activé, soit celui du plan de session. Cet acte envoie une série de mpi correspondant au plan de session, soit les étapes *Début*, *Milieu* et *Fin*. Le planificateur reçoit ces étapes suite à leur publication et les mets dans son plan. Il envoie ensuite la première étape, soit *Step-Debut.* L'État correspondant active l'acte de plan générique de salutation. Le planificateur reçoit ensuite l'étape de salutation qu'il ajoute comme sous-plan de l'étape début. Comme il y a une étape optionnelle de disponible, il demande ensuite l'étape suivante. Durant le processus de délibération, le profil de l'apprenant ajoute s'il y a une préférence pour l'étudiant utilisé. Dans le cas de Claire, il indique qu'elle aime les plaisanteries, dans le cas de Juliette, il indique qu'elle n'aime pas les plaisanteries et dans le cas de Pierre-Jean, le profil de l'apprenant ne fait aucun ajout puisque cet étudiant n'a pas de préférence sur ce sujet. Ces préférences activent à leur tour l'État optionnel *Préférence Plaisanterie*. Cet État optionnel envoie une énergie favorisante lorsque la préférence est de

type aimé (*likes*) pour l'utilisateur et une énergie inhibante lorsque cette préférence est de type pas aimé (*hates*) par l'utilisateur. Dans les tests, dans le cas de Claire, l'acte *Plaisanterie* est presque toujours activé, dans le cas de Juliette, c'est l'acte *Noop* qui est presque toujours activé. Dans le cas de Pierre-Jean, c'est majoritairement l'acte *Noop* qui est activé, mais ceci dépend de l'ordre de compilation des actes du réseau lors de la création de son fichier. L'algorithme du choix de l'acte à exécuter prend le premier acte exécutable trouvé lorsqu'il y a égalité dans leur niveau d'activation. Pour cet exemple, nous allons continuer avec l'hypothèse que c'est l'acte *Noop* qui est choisi. Le planificateur n'ajoute alors aucune étape au plan et envoie l'étape suivante sous forme de mpi, soit l'étape de la salutation.

Ce mpi active l'État correspondant dans le RAD qui à son tour déclenche la proposition d'acte *Salutation*. Le mpi fourni par cet acte active l'État correspondant dans l'EAO qui à son tour déclenche l'acte opérationnel *Salutation* qui lance l'exécution de son Microprocessus d'action. Ce Microprocessus produit un arbre mpi contenant les informations de base pour un graphe de concept linguistique, soit la forme de la phrase (Fragment), le but communicatif (Get the attention), l'orateur (CTS) et les premiers concepts de la phrase (Salutation-Début), ainsi qu'un mpi de type *StateActivator* qui va servir à aider au déroulement de la séquence de création de la phrase. Ce mpi de contrôle indique que l'étape suivante est de choisir s'il y aura un titre civil dans la salutation. Ici, durant la délibération, le profil de l'apprenant ajoute la préférence de l'étudiant sur le sujet. La proportion des choix de l'acte suite à l'activation de l'État optionnel est similaire à celle décrite plus haut selon la préférence. Le Réseau des Actes déclenche alors la proposition d'acte, qui, à son tour, déclenche l'acte opérationnel qui déclenche alors les Microprocessus d'action qui lui sont associés. Il est important de noter que le transfert d'information entre le RAD et le EAO est effectué tel que décrit dans les sections sur le Réseau des Actes à deux tiers et de la mémoire à court terme. Les deux actes effectuent comme tâche le retrait de l'ancien mpi de contrôle et l'ajout d'un nouveau mpi de contrôle pour le nom. Dans le cas où le titre civil a été choisi, l'ajout du concept Titre Civil-Usager est ajouté à l'arbre de mpi. Durant la délibération, le profil de l'apprenant attache la préférence de l'étudiant pour le nom et si le concept du titre civil est présent, il ajoute les mpi d'informations additionnelles du sexe et de l'âge de l'étudiant au mpi du concept titre civil. Un autre choix est effectué de la même façon que le

choix précédent, mais pour le nom cette fois-ci. Dans les deux cas, le mpi de contrôle est retiré et un mpi *Graphe Concept-Terminer* est attaché à l'arbre mpi. Si le nom est sélectionné, le mpi *Concept-Nom* est attaché au mpi *Concept-Titre Civil* s'il est présent, sinon il est attaché au mpi *Concept-Salutation*. Durant la délibération, si le mpi *Concept-Nom* est présent, le profil de l'apprenant attache les informations additionnelles nom et prénom au concept sous forme de mpi.

Le mpi *Graphe Concept-Terminer* déclenche l'État *Prêt à Envoyer* qui déclenche l'acte *CreerEtEnvoyer* (désormais, je vais omettre de citer le déclenchement de l'acte dans le RAD et l'EAO puisque c'est redondant.) Le *mpac* de l'acte envoie l'arbre mpi au générateur de texte, transforme la valeur du mpi *Graphe Concept-Terminer* en mpi *Graphe Concept-Envoyer* et ajoute un mpi *Step_Status-Terminer*. Le générateur de texte transforme l'arbre mpi reçu en sa structure de donnée interne et procède à la création du texte tel que décrit dans la section sur le générateur de texte, puis affiche le texte à l'écran. Le mpi *Step_Status-Terminer* active l'acte *SendNextStep*, qui lui envoie un mpi *Next Step*.

Voici la salutation la plus probante pour chacun des profils étudiants :

1. Pierre-Jean Jacques : CTS : Salut Pierre-Jean
2. Claire Brossard : CTS : Bonjour Mlle
3. Juliette Lapinot : CTS : Bonjour Mme Lapinot

## 7.3 Explication sur la matière à présenter

### 7.3.1 Description

Ce sous-scénario choisit le prochain concept à être présenté. Il le nomme, donne les explications sur le concept choisi et ses sous-concepts puis demande à l'étudiant de les mémoriser. Il est exécuté directement à la suite du sous-scénario précédent.

### 7.3.2 Buts recherchés et composants testés

Ce scénario vérifie le bon fonctionnement de l'instanciation de plan générique à être ajouté dans le planificateur. Aussi il aide à tester les modules de simulation du modèle du domaine et du modèle de connaissance de l'apprenant. Il vérifie aussi le bon fonctionnement des choix de mise en forme des phrases utilisée, plus précisément le module linguistique et un aspect de réutilisabilité de séquence dans le réseau des actes.

### 7.3.3 Exemple de fonctionnement

À la suite de la fin de l'exemple précédent, le planificateur reçoit le mpi *Next Step*. Celui-ci envoie alors l'étape *Middle* sous forme de mpi. Ce mpi active l'étape correspondante qui déclenche l'acte du plan générique (PG) *EnseignerUnConcept*. Cet acte envoie la chaîne mpi du plan générique consistant des étapes : *IndiquerConcept*, *MontrerMatiere*, *MontrerSousConcept* et *VerifierConnaissance*. L'annexe B donne la liste de tous les plans génériques avec la chaîne complète de mpi générée. Durant la délibération, le modèle de connaissance de l'apprenant choisit un concept non maitrisé et l'attache à la chaîne de mpi. Dans cet exemple, le concept *Direction* est choisi. Le planificateur prend les informations du plan générique et les instancie. Durant l'instanciation, le planificateur attache le concept *Direction* à chacune des étapes, puis ajoute le plan sous l'étape *Middle*. Le planificateur envoie ensuite l'étape *IndiquerConcept* avec le concept relié. Ceci déclenche l'acte *Créer Tête de Phrase*.

Si on regarde bien dans la séquence du RAD (disponible dans l'annexe B), cet acte existe de nombreuses fois, chacun relié à un État différent. Cette forme de construction permet d'avoir une relation OU entre ces différents États qui déclenchent en fait tous le même acte dans l'EAO. Cette séquence permet au système de décider du trio but communicatif, mode de livraison et forme de la phrase. L'acte *Créer Tête de Phrase* crée un arbre mpi ayant comme tête le mpi *Concept_Graphe-Head* et comme enfants reliés à cette tête les mpi *Concept_Graphe_Property-Utterer-CTS*, *Concept-Direction* (pour cet exemple), *InformationGathering-CommunicativeGoal* et *TutoringGoal-IndiquerConcept* (ici, la valeur du mpi est le nom de l'étape en cours). Durant la délibération le module de forme linguistique attache au mpi *InformationGathering* des mpi de type *PossibleBC* avec les types de but

communicatif pour ce concept et but tutoriel. Le RAD choisit un des buts communicatifs possibles puis attache un mpi *Concept_Graphe_Property-CommunicativeGoal-but choisi* (à la fin de l'exemple sur cette séquence, je donnerai les choix qui ont été faits). Il retire aussi le mpi I*nformationGathering* qui a aussi pour effet de retirer les mpi de type *PossibleBC*. Il ajoute finalement un mpi *InformationGathering-DeliveryMode*. Durant la délibération, le module de forme linguistique attache à ce mpi des mpi de type *PossibleDM* selon ce qui est permis pour le but communicatif, le concept et le but tutoriel. CTS fait un choix parmi ces possibilités, attache un mpi *Concept_Graphe_Property-DeliveryMode-mode choisi* enlève le mpi *InformationGathering* puis ajoute un mpi *InformationGathering-PhraseForm*. Durant la délibération, le module de forme linguistique attache à ce mpi des mpi de type *PossiblePF* selon ce qui est permis pour le mode de livraison, le but communicatif, le concept et le but tutoriel. CTS fait un choix parmi ces possibilités, attache un mpi *Concept_Graphe_Property-PhraseForm-forme choisi* enlève le mpi *InformationGathering* puis ajoute un mpi *StateActivator-InterventionTypeChoosen*. Presque toutes les interventions écrites du système passent par cette séquence. Pour les prochaines interventions, je ne donnerai que les choix qui ont été faits par le système sans répéter toutes ces étapes précédentes. Il faut tout de même considérer qu'elles ont été effectuées. Dans le cas d'indiquer concept le choix est *Get_The_Attention*, *Monologue* et *Declarative*.

Avec les informations disponibles, le système choisit l'acte correspondant au but tutoriel qui ajoute dans ce cas-ci un mpi *Graphe Concept-Terminer* et retire le mpi *StateActivator*. Ceci active ensuite la séquence qui envoie le texte à l'étudiant, comme on a vu durant l'exemple du scénario de la salutation. Le texte fournit par ce but tutoriel dans cet exemple est : « Nous allons maintenant aborder le concept Direction. »

Le planificateur envoie maintenant l'étape suivante *MontrerMatiere*. Ceci active la séquence du choix de la forme de l'intervention. Le système choisit *Explanation*, *Monologue* et *Declarative*. Ensuite, l'acte relié à ce but tutoriel se déclenche et ajoute la chaine de mpi suivante : *Concept-Direction* (avec un *Concept_Propety-CorrespondanceType*), *Concept-Direction* (avec un *Concept_Propety-Data1*) et *Concept-Direction* (avec un *Concept_Propety-Data2*). Aussi l'acte enlève le mpi *StateActivator*, et ajoute le mpi de terminaison de graphe. Durant la délibération, le module du modèle du domaine ajoute les

informations qu'il possède au graphe mpi. La séquence d'envoi (toujours comme dans les exemples précédents) est enclenchée. Le concept Direction possède sa propre règle au niveau de l'explication et produit la phrase « Il y a 6 directions. » Le planificateur envoie maintenant l'étape suivante qui est *MontrerSousConcept*.

Cette étape déclenche un autre plan générique qui ajoute au plan du planificateur (toujours de la même façon) les étapes *TransitionNiveau* et *MontrerData*. Le planificateur envoie ensuite l'étape *TransitionNiveau* et après avoir passé au travers de toutes les étapes décrites plus haut affiche à l'écran le texte : « Ces Directions sont : ». Le planificateur envoie maintenant l'étape *MontrerData* qui active un nouveau plan générique (PrésenterBlocData) qui envoie le plan à instancier contenant comme seule étape *MontrerMatière*. Durant la délibération, le module du modèle de connaissance de l'apprenant attache les sous-concepts du concept Direction à l'arbre mpi (voir l'annexe A). Le planificateur instancie alors le plan générique, produisant une liste de 6 étapes *MontrerMatière* chacun avec un des sous-concepts fournis. Chacune des étapes *MontrerMatière* fournies par la suite ici par le planificateur effectue le même fonctionnement que celui décrit durant l'étape *MontrerMatiere-Direction*. La règle de production du texte pour ces sous-concepts est : data1 correspond à data2. L'étape *VérifierConnaissance* sera décrite durant le scénario suivant. Les phrases produites pour les sous concepts sont :

1. Avant correspond à Forward.
2. Arrière correspond à Aft.
3. Gauche correspond à Port.
4. Droite correspond à Starboard.
5. Haut correspond à Zenith.
6. Bas correspond à Nadir.

## 7.4 Question sur la matière à présenter

### 7.4.1 Description

Ce scénario sert à vérifier les interactions entre CTS et l'utilisateur. Dans un premier lieu, CTS pose des questions à l'utilisateur sur la matière vue qui se répondent par un seul mot ou expression. Nous allons voir le fonctionnement du système dans le cas d'une réponse bonne et mauvaise. Aussi, nous verrons comment, après la troisième mauvaise réponse, le système abandonne le plan et décide de réexpliquer la matière.

### 7.4.2 Buts recherchés et composants testés

Ce scénario cherche à vérifier le comportement du système lors d'interactions avec l'utilisateur. Entre autres, il permet de vérifier la capacité au système de recevoir une réponse, le bon fonctionnement du module de compréhension de la langue naturelle et de la perception du texte, la capacité au modèle du domaine de déterminer la véracité d'une réponse, les fonctions du modèle de connaissance de l'apprenant sur la note de compréhension des concepts et la capacité du planificateur de laisser tomber un plan et de le remplacer par un nouveau.

### 7.4.3 Exemple de fonctionnement

Suite au scénario suivant, le planificateur envoie l'étape *VérifierConnaissance*. Il reçoit par la suite les étapes *LienQuestionnaire*, *QuestionnerConcept* et *QuestionnerSousConcept*. Il envoie ensuite l'étape *LienQuestionnaire*. Durant l'exécution de cette étape, un mpi *Concept-LienQuestion* est attaché au concept principal. Aussi un mpi *Awaited_Action-LienQuestion* est attaché à l'arbre de mpi indiquant au système qu'il devra attendre une réponse de l'utilisateur en lien avec la question. Le texte fourni par le système pour cette intervention est « Mémoriser ces informations et lorsque vous vous sentirez prêt, dites n'importe quoi, je vous poserai quelques questions. » Le système démarre aussi un Microprocessus de confirmation (mpc). Ce Microprocessus reste actif jusqu'à ce qu'il trouve un mpi du type *Text-Utterer*. Lorsque l'utilisateur envoie une réponse, la perception reçoit le texte et le transforme en arbre mpi contenant un mpi pour le texte entré de type *Text-String*, un mpi *Text-Utterer-User*, et un mpi *Text-Shown-False*. La compréhension du texte est effectuée ici, mais j'en parlerai un peu plus loin lors des réponses aux questions. Lorsque ce mpi est trouvé, le mpc lui attache un mpi *ActionChecked-LienQuestion* puis s'éteint. Ici, deux mpi peuvent faire réagir le système. Le premier est le mpi *Text-Shown*, qui déclenche un acte qui permet d'afficher à l'écran le texte entré par l'utilisateur. Cet acte change à valeur du mpi à vrai puis republie l'arbre. Le deuxième est le mpi *ActionChecked* qui déclenche ici la suite du déroulement et qui ne republit pas l'arbre. Le système déclenche ensuite l'acte qui permet d'effacer le contenu de la fenêtre de clavardage et envoie un *Next Step*. Le planificateur envoie maintenant l'étape *QuestionnerConcept*.

Durant cette étape, le système choisit les paramètres de la phrase et obtient le trio *Probing the student inference process*, *Monologue* et *Interrogative*. Le système construit ensuite l'arbre mpi représentant la question. Durant cette étape, deux mpi sont ajoutés. Le premier est un mpi *Concept-NomConcept* qui sera utilisé pour créer la question. À ce mpi, on ajoutera un mpi de propriété qui indiquera quelle information du concept sera utilisée pour poser la question. Le deuxième mpi ajouté est un mpi *Awaited_Answer-NomConcept* auquel on ajoute un mpi de propriété qui indique quelle information on attend recevoir en réponse. Après la délibération où les différents sous-systèmes ajoutent leurs informations complémentaires, le texte est envoyé au module de génération de texte pour être construit et affiché. Aussi, le système démarre un mpc contenant la clé mémorielle de la question posée.

Ce mpc attend 5 secondes pour recevoir la réponse, puis, s'il n'a pas rien entendu, envoie un mpi *Problem-NoAnswer* couplé au numéro de la question. Le RAPG envoie le plan générique *NoAnswer* qui contient l'étape *NoAnswerTimeOut*. Le planificateur instancie ce plan en ajoutant à l'étape le numéro de la question, puis envoie l'étape *NoAnswerTimeOut* instanciée au système. Le système ici choisit entre 3 solutions. La première consiste simplement à réinitialiser le mpc de la question et de donner 5 secondes supplémentaires à l'étudiant. La seconde consiste à indiquer à l'étudiant qu'on attend une réponse de sa part puis réinitialiser le mpc de la question et la troisième est de reposer la question et de réinitialiser son mpc. Pour l'instant, le système choisit à sa convenance la méthode à utiliser, mais rien n'empêche de placer des États optionnels qui permettraient au système de faire son choix basé sur des préférences de l'utilisateur et d'autres facteurs.

Dans le cas où une réponse est donnée par l'utilisateur, le système commence d'abord par créer l'arbre mpi de la réponse par la perception, puis le module de compréhension du langage naturel effectue ses traitements et ajoute des mpi à l'arbre comme indiqué au chapitre précédent. Le mpc attache à l'arbre, si ce n'est pas fait, un mpi contenant la clé mémorielle de la question qu'il surveille, puis réinitialise son temps d'attente. Lorsque la réponse de l'utilisateur est publiée, durant la délibération, le modèle du domaine analyse la réponse à la question. Il s'assure d'abord que la réponse ait été affichée à l'écran et qu'elle n'a pas été corrigée. Ensuite, il extrait la réponse attendue que la question associée à la clé mémorielle qui a été attachée par le mpc. Il compare finalement les deux réponses et indique en attachant

un mpi de type *Answer_Correctness* indique la valeur de vérité de la réponse. Finalement, il attache aussi un mpi *AnsweredQuestion-NumérodeQuestion* pour indiquer quelle question a été répondue. Bien que plusieurs parties de ce système aient été conçues pour pouvoir accommoder plusieurs questions simultanées ou imbriquées, seule la partie qui détermine la valeur de vérité de la réponse ne permet pas ce genre de traitement et ne peut accommoder qu'une seule question à la fois. Après ce traitement, toujours durant la délibération, le modèle de connaissance de l'apprenant se met à jour en incrémentant le nombre de réponses fournies sur le concept visé. Aussi, si la réponse est bonne, il incrémente le nombre de bonnes réponses du concept, sinon il incrémente le nombre de mauvaises réponses consécutives reçu. Finalement, il met à jour la note de compréhension du concept comme indiqué au chapitre 5. Après la délibération, si la réponse est bonne, le plan générique envoie au planificateur l'étape *GoodAnswer* qui est ensuite retransmise au système. Suite à cette étape, le système envoie une notification à l'utilisateur indiquant qu'il a donné une bonne réponse, puis envoie un mpi *Next Step*. Si la réponse était mauvaise, l'étape *BadAnswer* est envoyée. Une notification indiquant que la réponse est mauvaise est envoyée, puis le mpi *Next Step* est transmis.

L'étape *QuestionnerSousConcept* est envoyée et le planificateur reçoit, après délibération, le plan générique contenant l'étape *QuestionnerConcept* et les sous-concepts du concept *Direction*. Le planificateur instancie le plan puis envoie la prochaine étape. Ici, le fonctionnement est identique à celui décrit plus haut.

Maintenant, supposons que durant l'exécution du questionnaire, l'étudiant donne des mauvaises réponses aux questions 3, 4 et 5. Lorsque le modèle de connaissance de l'apprenant reçoit la troisième mauvaise réponse, il attache un mpi *Knowledge-Unsufficient* à l'arbre mpi de la réponse. Ce nouveau mpi permet de déclencher l'envoi du plan générique *DropAndReshow* qui contient les étapes *MontrerConcept* et *MontrerSousConcepts* avec l'information additionnelle d'utiliser une image avec les explications. Aussi, une commande Drop est présente dans le plan générique. Ceci ordonne au planificateur de laisser tomber les étapes de questionnement en cours et de les remplacer par les nouvelles étapes. Aussi, il replace l'étape abandonnée *VérifierConnaissance* à la suite de ces nouvelles étapes. Le

fonctionnement de ces étapes ajoutées est le même que pour le scénario « Explication sur la matière à présenter » à un détail près, à chaque explication une image est aussi affichée.

## 7.5 Question posée par l'utilisateur

### 7.5.1 Description

Ce scénario peut être exécuté à n'importe quel moment durant le fonctionnement du système. Ici, l'utilisateur pose une question de la forme « Qu'est-ce que *expression*? » et le système lui répond avec l'information correspondant dans le concept relié à l'expression. La réponse tient compte du contexte et des informations fournies passées.

### 7.5.2 Buts recherchés et composants testés

Le scénario a pour but de voir comment le système réagit à une interruption durant une séquence. Aussi, il permet de tester la capacité au système d'obtenir les informations pour donner la bonne réponse et de vérifier la mémoire épisodique associative pour obtenir le bon concept à répondre.

### 7.5.3 Exemple de fonctionnement

L'utilisateur entre une question dans le système comme « Qu'est-ce que avant? ». La perception reçoit cette question, et après transformation et analyse comme décrite au scénario précédent, envoie au système un arbre mpi. Cet arbre est publié lorsque sa valeur d'activation est la plus grande dans le système. L'acte qui affiche le texte possède un Microprocessus d'action qui permet de déterminer le concept visé par la question. Durant le fonctionnement de l'acte, ce mpac crée d'abord pour chaque concept présent dans l'arbre (ce qui signifie pour chaque concept qui peut représenter l'expression) la liste des concepts pouvant s'y rattacher. Pour chaque concept, on obtient donc une liste de concepts correspondant à ses parents, ses enfants et ses sœurs. Ensuite on obtient dans la mémoire sémantique associative, toutes les clés des souvenirs qui possèdent un des concepts de la liste. On garde ainsi pour chaque concept présent dans la publication le numéro de clé s'y rattachant le plus élevé. Finalement, on prend le concept qui possède la clé la plus élevée et on le place dans un mpi *OtherPGInfo-DConcept|concept*. Si aucune correspondance dans la mémoire n'est trouvée, alors le système

choisit au hasard un des concepts possibles. Après la publication de l'arbre avec les nouvelles informations, un plan générique est fourni au planificateur contenant l'étape *MontrerConcept* couplée au concept choisi lors du travail précédent. Le planificateur envoie ensuite cette étape dans le système et elle est exécutée comme nous avons vu lors du scénario sur comment montrer la matière. Après que l'explication ait été donnée, le système continue à exécuter le plan établi.

# CONCLUSION

Dans ce mémoire, nous avons décrit une implémentation d'un planificateur de dialogue basé sur l'architecture à trois tiers de BEETLE en utilisant le système CTS. Nous avons vu comment CTS en utilisant cette méthode combinée à ses caractéristiques propres (Réseau des Actes, délibération, etc.) arrive à assembler un dialogue de haut niveau et comment il est raffiné jusqu'à ce qu'un texte en langage naturel soit produit.

Entre autres, ce système permet de voir comment un système multiagent est capable de communiquer et de collaborer pour permettre la création d'un dialogue tutoriel cohérent, flexible et sensible au contexte. Plusieurs sous-systèmes et modules que nous pouvons qualifier d'agents interagissent entre eux pour pouvoir accomplir le dialogue avec l'étudiant.

Au premier tiers, nous avons vu comment le Réseau des Actes (dans sa partie sur les plans génériques) interagit avec le module ou mémoire de planification pour former les plans à haut niveau du dialogue. Nous avons vu comment la perception et les différents acteurs internes comme le profil de l'apprenant viennent aider à ce choix en allumant des États contextuels au niveau du réseau des actes. Nous avons vu aussi comment les plans génériques sont instanciés avec l'aide des différents acteurs internes.

Au deuxième tiers, le Réseau des Actes décisionnel choisit contextuellement la méthode à utiliser pour permettre la construction de l'intervention du système tutoriel avec la collaboration de ces divers acteurs internes qui fournissent diverses informations nécessaires soit en aidant à spécifier le contexte ou en spécifiant des informations.

La réalisation du texte et l'utilisation de média (comme des images) sont effectuées au troisième tiers. Ici, les tampons de sorties du système reçoivent les informations à transformer de CTS et les rendent compréhensibles à l'usager. Le générateur de texte reçoit de CTS la

coalition d'informations permettant de construire la ou les phrases avant de les envoyer à la fenêtre de clavardage. Aussi, la fenêtre d'affichage des images reçoit l'image à afficher.

Nous avons constaté que la richesse du texte en langage naturel fourni par le système est directement reliée à la qualité des différentes bases de connaissances du système et du générateur de textes. Dans le cas du présent système, les bases de connaissances étant simulées et le générateur de texte fonctionnant à l'aide de scripts, nous avons obtenu une machine à États finis. Considérant que la majorité des systèmes tutoriels intelligents utilisent ce type de planificateur de dialogue, nous pouvons affirmer que notre système saura produire des résultats minimalement équivalents à ceux produits par d'autres systèmes. Compte tenu des mécanismes complémentaires proposés par notre architecture, nous espérons en fait atteindre des interventions ultimement mieux adaptées que celles possibles avec un système essentiellement prédéterminé, notamment avec l'ajout de bases de connaissances plus riches et d'un générateur de texte avancé.

Parmi les autres ajouts que j'ai faits, la mémoire à court terme joue un rôle important dans la compréhension du langage naturel par le contexte. Entre autres, la mémoire à court terme permet de choisir correctement le concept visé par la phrase de l'étudiant en examinant les concepts utilisés récemment dans le passé. Un autre ajout important est la délibération. Bien qu'elle fût prévue dans la théorie, elle n'avait jamais été implémentée. La délibération joue un rôle important dans la collaboration entre les acteurs internes, car c'est elle qui leur donne le temps de réagir aux informations publiées.

La division du Réseau des Actes en tiers décisionnel et opérationnel confère plusieurs avantages au système. Tout d'abord, elle permet au Réseau des Actes de proposer l'acte à entreprendre comme il était prévu dans la théorie. Aussi, un autre avantage important de cette division est la réutilisabilité des actes. En effet, il devient désormais plus facile d'utiliser un même acte dans plusieurs séquences différentes sans avoir à le réécrire au complet et sans que les préconditions et les postconditions associées à ses différentes apparitions ne viennent interférer entre eux. L'inconvénient majeur de tous ces ajouts est lié au ralentissement de la performance résultant de la combinaison de la délibération et de la division en deux tiers du réseau des actes. Ce ralentissement atteint un facteur de 6 et peut même le dépasser si la

délibération est particulièrement longue, mais nous avons trouvé que les bénéfices ajoutés valent bien ce coût supplémentaire dans le prototype.

Dans les prochains travaux sur CTS, plusieurs points devront être examinés et améliorés, notamment :

- Permettre aux coalitions d'être navigables sans nécessairement avoir à le faire dans le réseau mpi sous-jacent;
- Permettre à la mémoire à court terme d'enregistrer autre chose que juste des arbres autrement dit : des graphes complexes et cycliques;
- Créer des mémoires à long terme qui permettront à CTS d'apprendre et de construire des bases de connaissances fondées sur ses propres expériences;
- Trouver des solutions à l'instabilité du Réseau des Actes notamment au niveau de la distribution de l'énergie dans le réseau;
- Explorer les possibilités offertes par le changement de définition d'un but par cette définition : « Un but est un désir de modifier l'État actuel de son environnement en un autre État »;
- Permettre des liens ET-OU-ET dans les préconditions des actes;
- Permettre aux actes de lancer des chaines mpi (et non pas que des mpi dissociés);
- Modifier l'éditeur du Réseau des Actes de façon à accommoder les différents changements structurels du réseau;

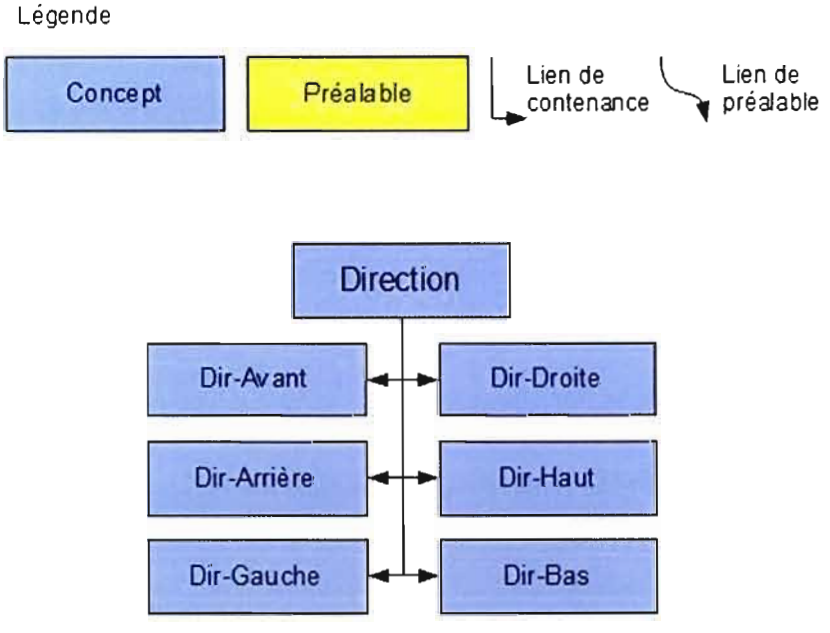
Le système proposé par CTS est très prometteur et pourrait être appliqué à différents domaines comme la robotique. Le présent travail a contribué beaucoup à la stabilisation du système et les futurs travaux aideront à en augmenter sa puissance.

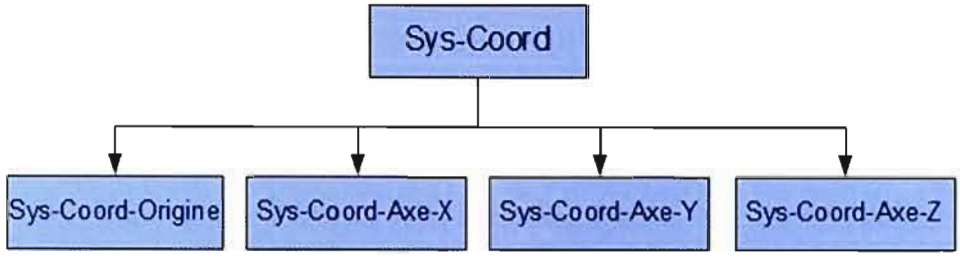
APPENDICE A

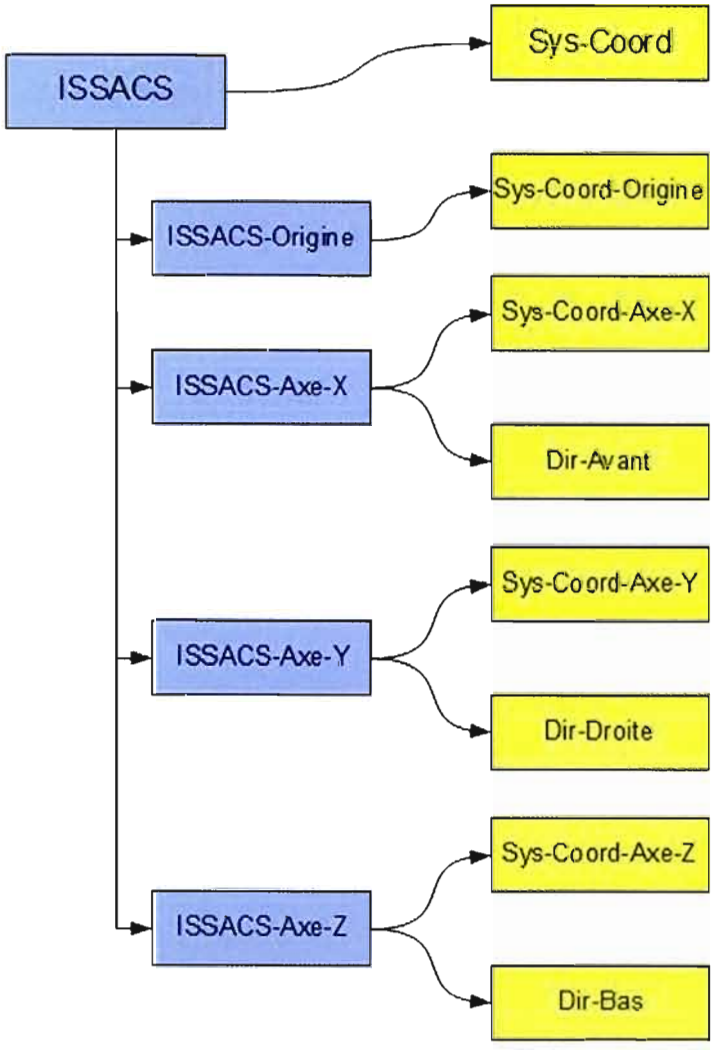
# MODÈLE DU DOMAINE UTILISÉ

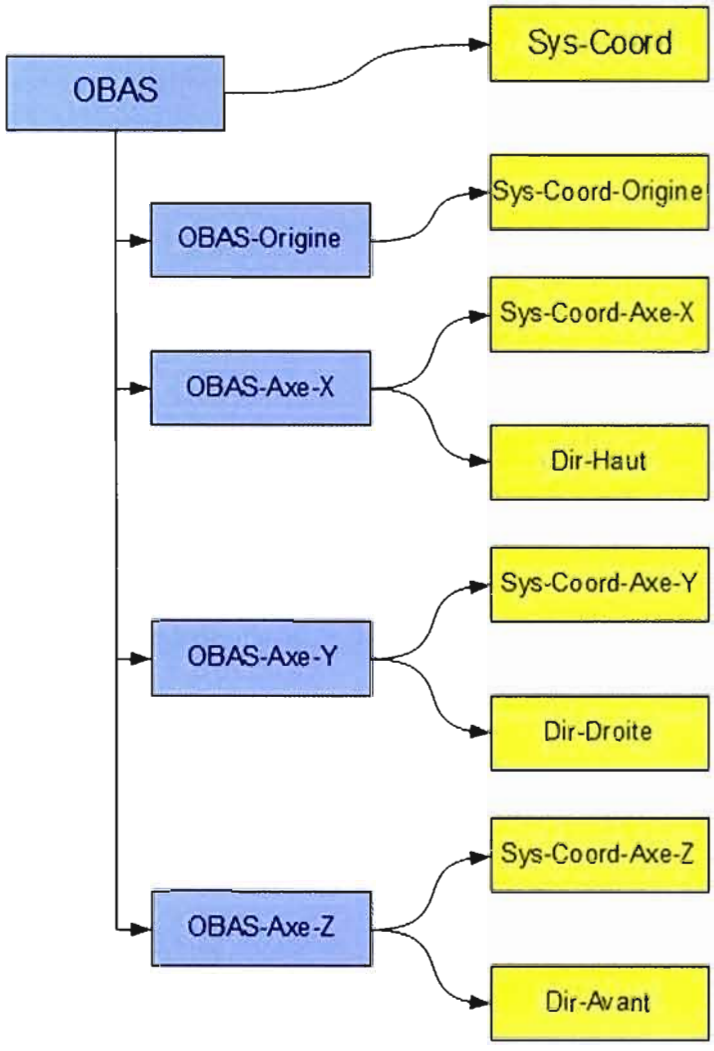


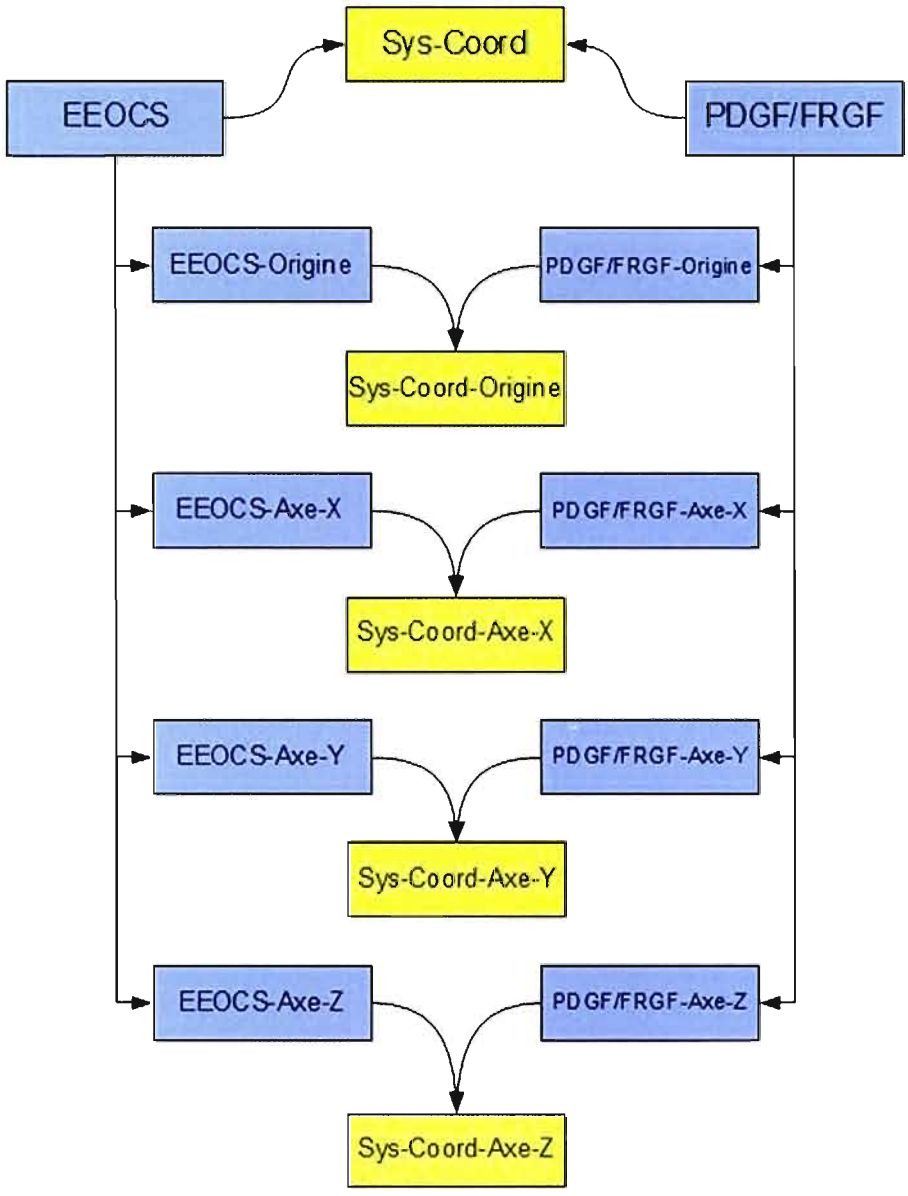
## A.1 Diagramme du modèle de connaissance




Sys-Coord
Sys-Coord-Origine
Sys-Coord-Axe-X
Sys-Coord-Axe-Y
Sys-Coord-Axe-Z

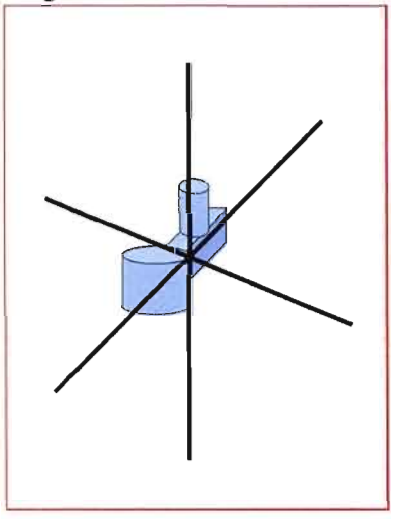

ISSACS
Sys-Coord
ISSACS-Origine
Sys-Coord-Origine
ISSACS-Axe-X
Sys-Coord-Axe-X
Dir-Avant
ISSACS-Axe-Y
Sys-Coord-Axe-Y
Dir-Droite
ISSACS-Axe-Z
Sys-Coord-Axe-Z
Dir-Bas


OBAS
Sys-Coord
OBAS-Origine
Sys-Coord-Origine
OBAS-Axe-X
Sys-Coord-Axe-X
Dir-Haut
OBAS-Axe-Y
Sys-Coord-Axe-Y
Dir-Droite
OBAS-Axe-Z
Sys-Coord-Axe-Z
Dir-Avant

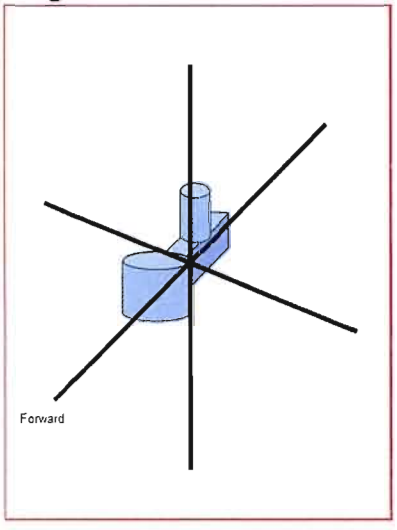

Sys-Coord
EEOCS
PDGF/FRGF
EEOCS-Origine
PDGF/FRGF-Origine
Sys-Coord-Origine
EEOCS-Axe-X
PDGF/FRGF-Axe-X
Sys-Coord-Axe-X
EEOCS-Axe-Y
PDGF/FRGF-Axe-Y
Sys-Coord-Axe-Y
EEOCS-Axe-Z
PDGF/FRGF-Axe-Z
Sys-Coord-Axe-Z

## A.2 Contenu des concepts

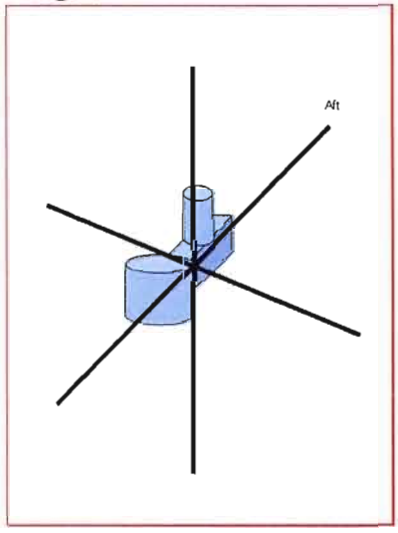
**Nom du concept :** Direction
**Donnée 1 :** Nombre de directions
**Donnée 2 :** 6
**Type de correspondance :** Informationel
**Image :**



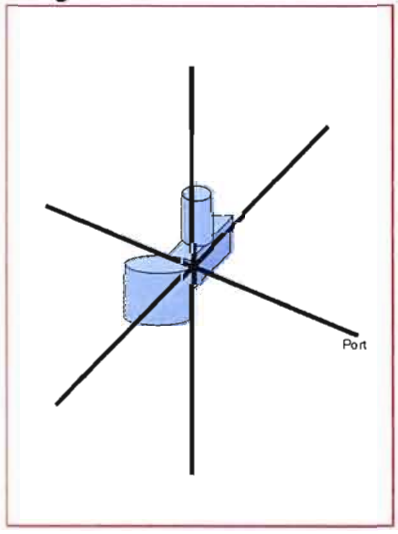
**Nom du Concept :** Dir-Avant
**Donnée 1 :** Avant
**Donnée 2 :** Forward
**Type de correspondance :** Assossiative
**Image :**



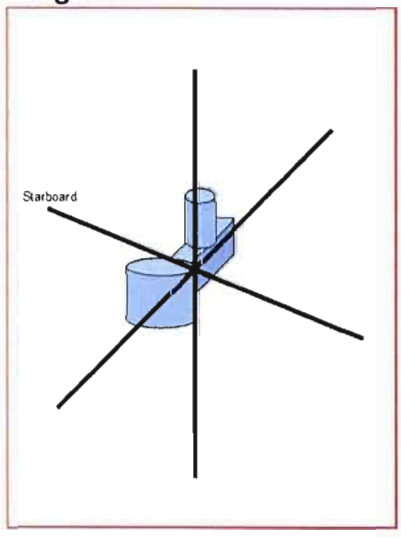
**Nom du Concept :** Dir-Arriere
**Donnée 1 :** Arriere
**Donnée 2 :** Aft
**Type de correspondance :** Assossiative
**Image :**



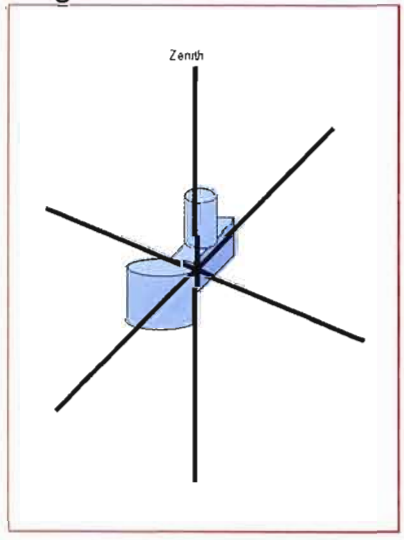

**Nom du Concept :** Dir-Gauche
**Donnée 1 :** Gauche
**Donnée 2 :** Port
**Type de correspondance :** Assossiative
**Image :**



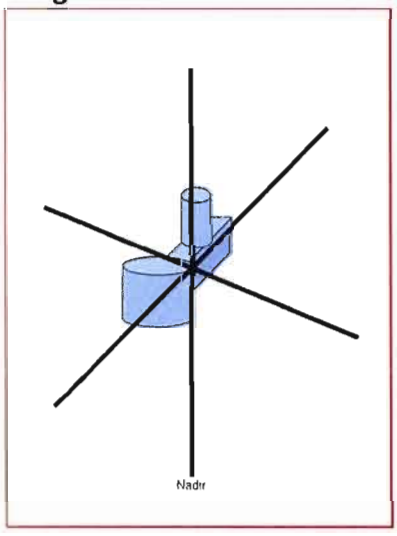
**Nom du Concept :** Dir-Droite
**Donnée 1 :** Droite
**Donnée 2 :** Starboard
**Type de correspondance :** Assossiative
**Image :**




**Nom du Concept :** Dir-Haut
**Donnée 1 :** Haut
**Donnée 2 :** Zenith
**Type de correspondance :** Assossiative
**Image :**



**Nom du Concept :** Dir-Bas
**Donnée 1 :** Bas
**Donnée 2 :** Nadir
**Type de correspondance :** Assossiative
**Image :**




**Nom du Concept :** Sys-Coord
**Donnée 1 :** Système de Coordonnées
**Donnée 2 :** Système de référence permettant de positionner un point dans l'espace
**Type de correspondance :** Definition

**Nom du Concept :** Sys-Coord-Origine
**Donnée 1 :** Origine
**Donnée 2 :** Représente le point (0,0,0) dans un système de coordonnées
**Type de correspondance :** Definition

**Nom du Concept :** Sys-Coord-Axe-X
**Donnée 1 :** X
**Donnée 2 :** Direction de l'axe des X
**Type de correspondance :** Definition

**Nom du Concept :** Sys-Coord-Axe-Y
**Donnée 1 :** Y
**Donnée 2 :** Direction de l'axe des Y
**Type de correspondance :** Definition

**Nom du Concept :** Sys-Coord-Axe-Z
**Donnée 1 :** Z
**Donnée 2 :** Direction de l'axe des Z
**Type de correspondance :** Definition

**Nom du Concept :** ISSACS
**Donnée 1 :** ISSACS
**Donnée 2 :** Définition de ISSACS
**Type de correspondance :** Definition

**Nom du Concept :** ISSACS-Origine
**Donnée 1 :** Origine
**Donnée 2 :** Centre du segment S0 de la station spaciale internationale
**Type de correspondance :** Informationel

**Nom du Concept :** ISSACS-Axe-X
**Donnée 1 :** X
**Donnée 2 :** Forward
**Type de correspondance :** Informationel

**Nom du Concept :** ISSACS-Axe-Y
**Donnée 1 :** Y
**Donnée 2 :** Starboard
**Type de correspondance :** Informationel

**Nom du Concept :** ISSACS-Axe-Z
**Donnée 1 :** Z
**Donnée 2 :** Nadir
**Type de correspondance :** Informationel

**Nom du Concept :** OBAS
**Donnée 1 :** OBAS
**Donnée 2 :** Définition de OBAS
**Type de correspondance :** Definition

**Nom du Concept :** OBAS-Origine
**Donnée 1 :** Origine
**Donnée 2 :** Dessus du réservoir externe
**Type de correspondance :** Informationel

**Nom du Concept :** OBAS-Axe-X
**Donnée 1 :** X
**Donnée 2 :** Zenith
**Type de correspondance :** Informationel

**Nom du Concept :** OBAS-Axe-Y
**Donnée 1 :** Y
**Donnée 2 :** Starboard
**Type de correspondance :** Informationel

**Nom du Concept :** OBAS-Axe-Z
**Donnée 1 :** Z
**Donnée 2 :** Forward
**Type de correspondance :** Informationel

**Nom du Concept :** EEOCS
**Donnée 1 :** EEOCS
**Donnée 2 :** Définition de EEOCS
**Type de correspondance :** Definition

**Nom du Concept :** EEOCS-Origine
**Donnée 1 :** Origine
**Donnée 2 :** Centre de LEE (open end flush)
**Type de correspondance :** Informationel

**Nom du Concept :** EEOCS-Axe-X
**Donnée 1 :** X
**Donnée 2 :** Direction Extérieur à partir du LEE
**Type de correspondance :** Informationel

**Nom du Concept :** EEOCS-Axe-Y
**Donnée 1 :** Y
**Donnée 2 :** Hors de la caméra
**Type de correspondance :** Informationel

**Nom du Concept :** EEOCS-Axe-Z
**Donnée 1 :** Z
**Donnée 2 :** Complétion du système de coordonnées
**Type de correspondance :** Informationel

**Nom du Concept :** PDGF/FRGF
**Donnée 1 :** PDGF/FRGF
**Donnée 2 :** Définition de PDGF/FRGF
**Type de correspondance :** Definition

**Nom du Concept :** PDGF/FRGF-Origine
**Donnée 1 :** Origine
**Donnée 2 :** Intersection entre (grapple pin) et (face plate) (bout du bras grapple)
**Type de correspondance :** Informationel

**Nom du Concept :** PDGF/FRGF-Axe-X
**Donnée 1 :** X
**Donnée 2 :** vers le (face plate) le long du manche
**Type de correspondance :** Informationel

**Nom du Concept :** PDGF/FRGF-Axe-Y
**Donnée 1 :** Y
**Donnée 2 :** s'eloignant de la cible d'alignement
**Type de correspondance :** Informationel

**Nom du Concept :** PDGF/FRGF-Axe-Z
**Donnée 1 :** Z
**Donnée 2 :** Complétion du système de coordonnées
**Type de correspondance :** Informationel

# APPENDICE B

# RÉSEAU DES ACTES



Nous allons voir ici le contenu du Réseau des Actes tel qu'utilisé pour mettre en œuvre le prototype développé. Pour chacun des tiers, nous allons d'abord voir la représentation graphique des actes et États, puis les informations additionnelles relatives à chaque tiers.

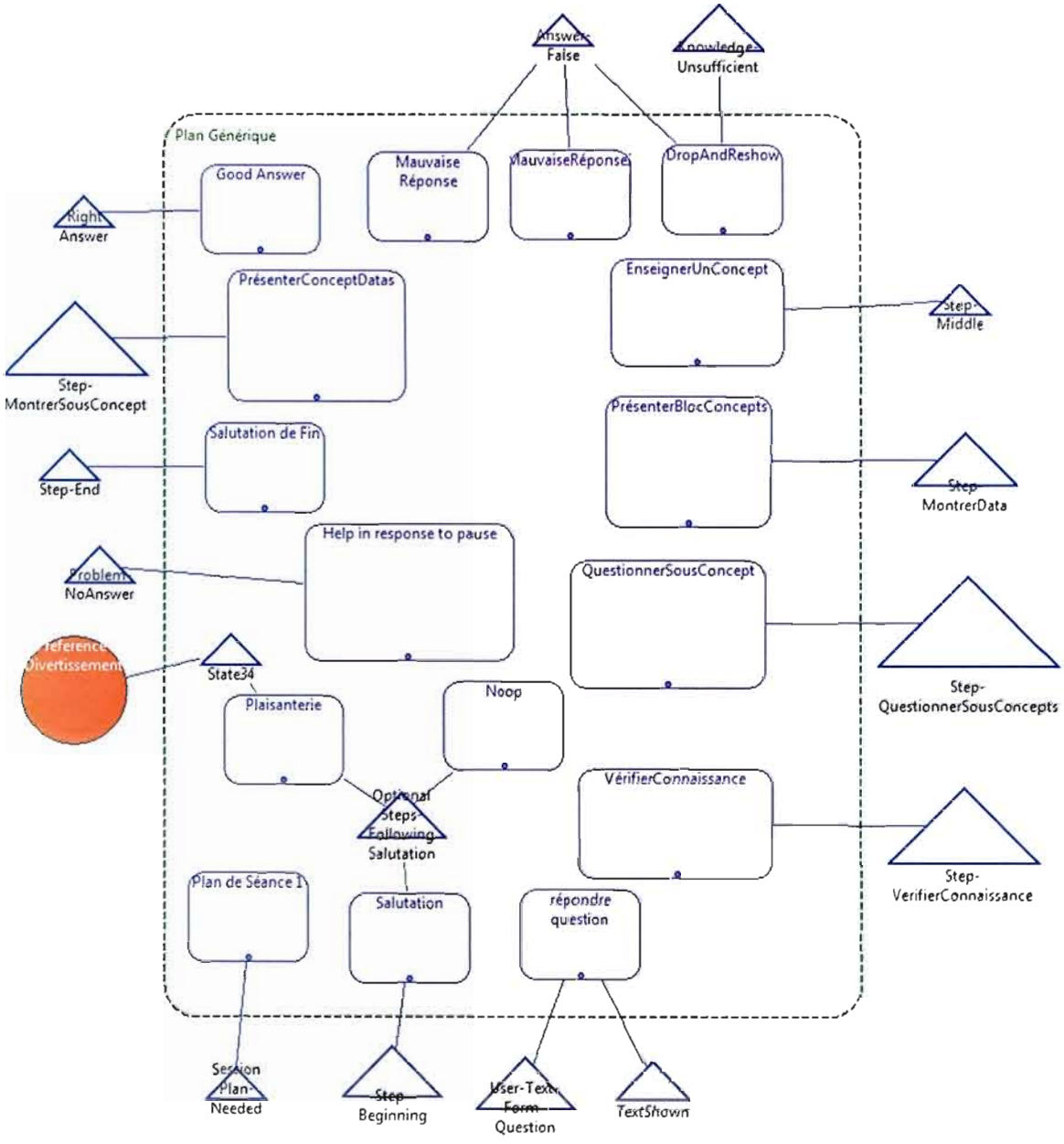
## B.1 Plans Génériques

Dans cette section nous allons voir les réseaux d'acte des plans génériques. Ensuite, je donnerai pour chaque État le mpi l'activant et finalement pour chaque acte la chaîne mpi que le Microprocessus sous-classant le Microprocessus d'action GenericPlanCodelet produit.

États :

| **Nom de l'État** | **Type du MPI déclencheur** | **Valeur du MPI déclencheur** |
|---|---|---|
| Session Plan-Needed | Session Plan | Needed |
| Step-Beginning | Step | Beginning |
| Optional Steps-Following Salution | Optional Steps | Following Salutation |
| User-Text-Form-Question | User-Text-Form | Question |
| TextShown | Text-Shown | True |
| Step-VerifierConnaissance | Step | VerifierConnaissance |
| Step-QuestionnerSousConcepts | Step | QuestionnerSousConcepts |
| Step-MontrerData | Step | MontrerData |

| | | |
|---|---|---|
| Step-Middle | Step | Middle |
| Step-MontrerSousConcept | Step | MontrerSousConcept |
| Step-End | Step | End |
| Knowledge-Unsufficient | Knowledge | Unsufficient |
| Answer-False | Answer_Correctness | False |
| Right Answer | Answer_Correctness | True |
| Problem-NoAnswer | Problem | NoAnswer |



États optionnels :

Preference divertissement :
Mpi en activation positive (favorisant) :
Type : Preference Valeur : Plaisanterie
Mpi en activation négative (contraignant) :
Type : Hates Valeur : Plaisanterie

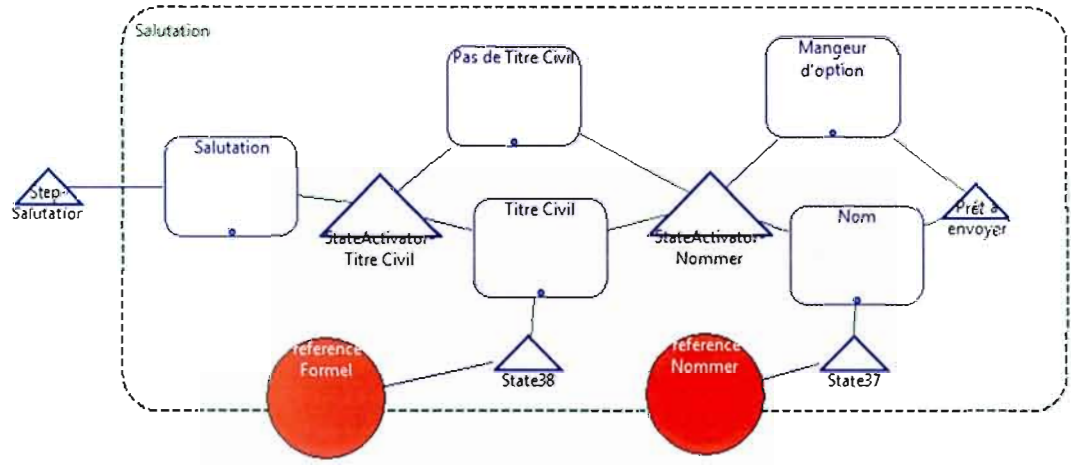
Contenu des actes :

| **Nom de l'acte** | **Chaîne de MPI (Type \| Valeur)** | |
|---|---|---|
| Plan de séance 1 | New Step | Start |
| | New Step | Step-Beginning |
| | New Step | Step-Middle |
| | New Step | Step-End |
| Salutation | New Step | Start |
| | New Step | Salutation |
| | Additional Step | Optional |
| Plaisanterie | New Step | Start |
| | New Step | Plaisanterie |
| Noop | New Step | Start |
| EnseignerUnConcept | New Step | Start |
| | New Step | Step-IndiquerConcept |
| | New Step | Step-MontrerMatiere |
| | New Step | Step-MontrerSousConcept |
| | New Step | Step-VerifierConnaissance |
| | Args | Start |
| | Looping | Sequentiel |
| | StepVariable | Concepts |
| | PGConcepts | Needed |
| | PGConcepts | H |
| PrésenterConceptDatas | New Step | Start |
| | New Step | Step-TransitionNiveau |
| | New Step | Step-MontrerData |
| | Args | Start |
| | DropPlan | SameAsLast |
| PrésenterBlocConcepts | New Step | Start |
| | New Step | Step-MontrerMatiere |
| | Args | Start |
| | Looping | Sequentiel |
| | StepVariable | Concepts |
| | PGConcepts | Needed |
| | PGConcepts | BFromH |
| | DropPlan | SameAsLast |

| Nom de l'acte | Chaîne de MPI (Type \| Valeur) | |
|---|---|---|
| VérifierConnaissance | New Step | Start |
| | New Step | Step-LienQuestionnaire |
| | New Step | Step-QuestionnerConcepts |
| | New Step | Step-QuestionnerSousConcepts |
| QuestionnerSousConcept | New Step | Start |
| | New Step | Step-QuestionnerConcepts |
| | Args | Start |
| | Looping | Sequentiel |
| | StepVariable | Concepts |
| | PGConcepts | Needed |
| | PGConcepts | BFromH |
| | DropPlan | SameAsLast |
| Good Answer | New Step | Start |
| | New Step | GoodAnswer |
| | Args | Start |
| | DropPlan | SameAsLast |
| Mauvaise Réponse | New Step | Start |
| | New Step | Step-BadAnswer |
| | New Step | Step-MontrerMatiere |
| | Args | Start |
| | DropPlan | SameAsLast |
| MauvaiseRéponse2 | New Step | Start |
| | New Step | Step-BadAnswer |
| | Args | Start |
| | DropPlan | SameAsLast |
| DropAndReshow | New Step | Start |
| | New Step | Step-MontrerMatiere |
| | New Step | Step-MontrerSousConcept |
| | Args | Start |
| | OtherPGInfo | With\|Image |
| | Drop | AddNewAndRetry |
| | Note : n'ajoute pas d'argument externe | |
| Help in response to pause | New Step | Start |
| | New Step | Step-NoAnswerTimeOut |
| Répondre Question | New Step | Start |
| | New Step | Step-MontrerMatiere |
| Salutation de Fin | New Step | Start |
| | New Step | Step-EndSalutation |

## B.2 Réseau des Actes décisionnel

Dans cette section nous allons voir les réseaux d'actes décisionnels avec une légère description de leur but. Ensuite, je donnerai pour chaque État le mpi activateur. Tous les actes de cette partie du Réseau des Actes possèdent le Microprocessus d'action ActProposition qui envoie en mémoire de travail le nom de l'acte à exécuter avec la clé mémorielle de la coalition l'ayant déclenché.



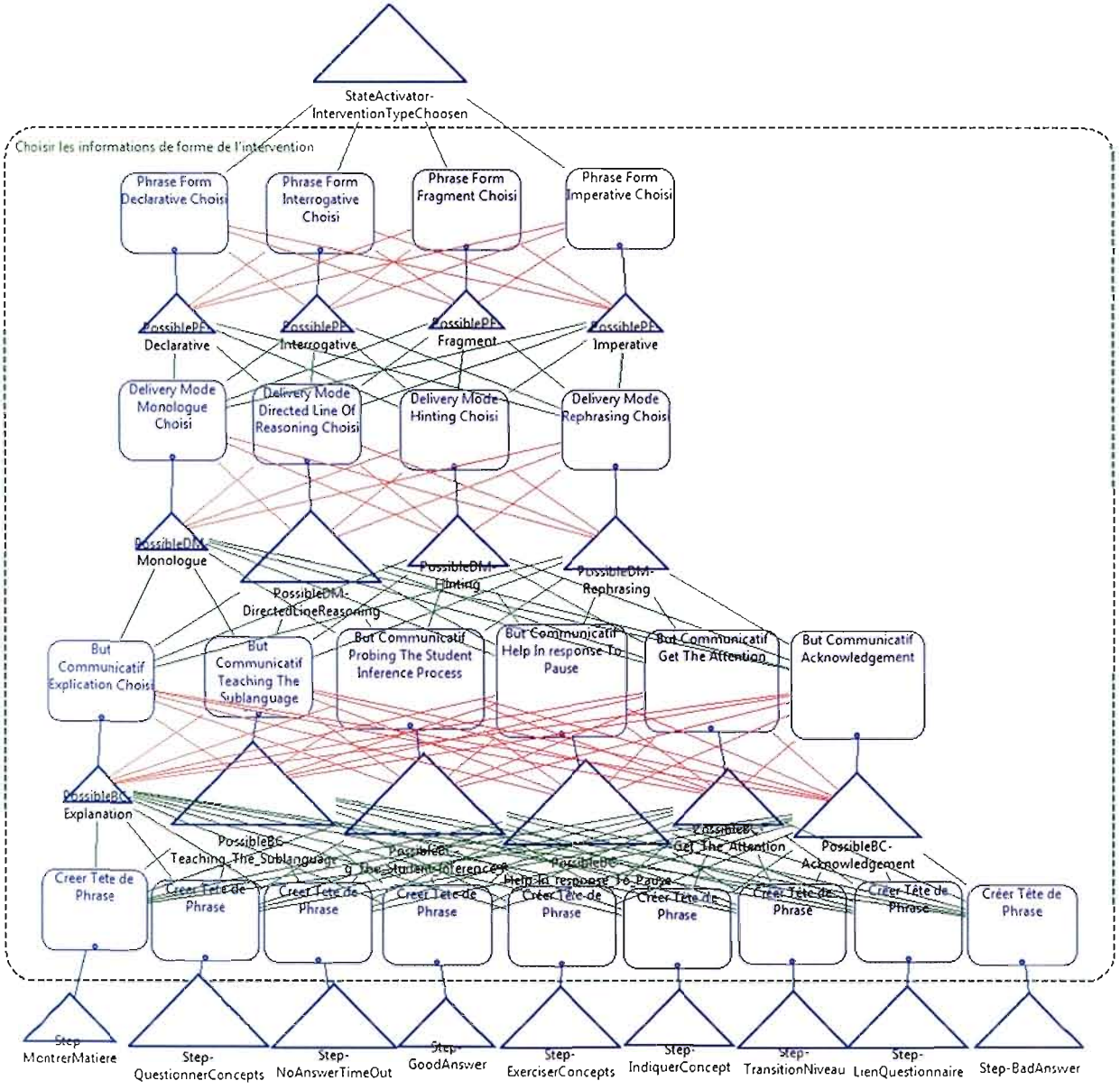

But : Permet de construire l'arbre mpi pour une salutation

États :

| Nom | Type du mpi | Valeur du mpi |
|---|---|---|
| Step-Salutation | Step | Salutation |
| StateActivator-Titre Civil | StateActivator | Titre Civil |
| StateActivator-Nommer | StateActivator | Nommer |
| Prêt à envoyer | Graphe Concept | Terminer |

États Optionnels :

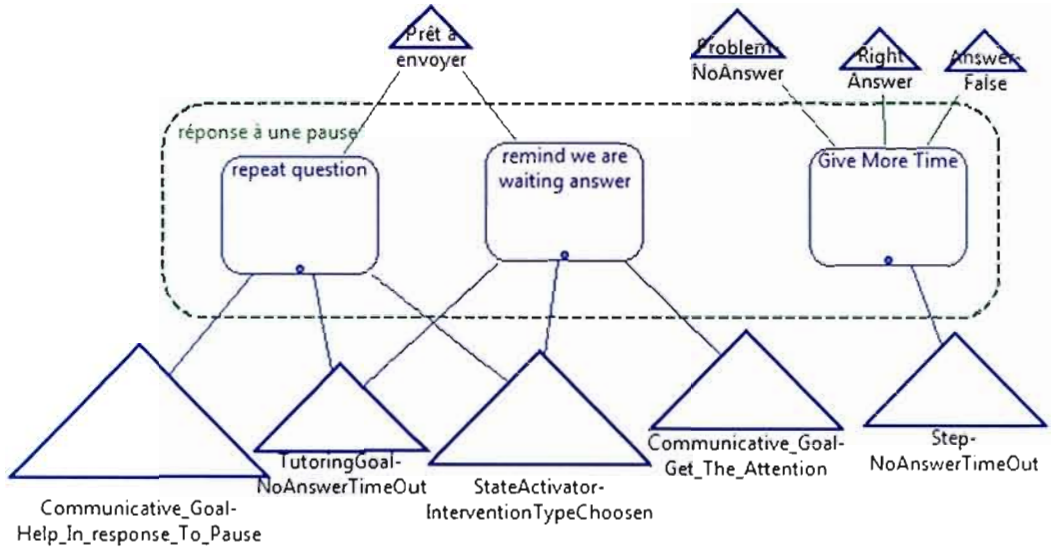
| | |
|---|---|
| **Préférence Formel** | |
| Activation Positive : Type mpi :Preference | Valeur mpi : Titre Civil |
| Activation Négative : Type mpi : Hates | Valeur mpi : Titre Civil |
| **Préférence Nommer** | |
| Activation Positive : Type mpi :Preference | Valeur mpi : Nommer |
| Activation Négative : Type mpi : Hates | Valeur mpi : Nommer |




But : Permet de construire la tête du graphe conceptuel de la phrase et de choisir les buts communicatifs, mode de livraison et forme de phrase. La série d'acte *Créer tête de Phrase* représente tous le même acte permettant de créer des liens ou entre les différentes préconditions.

États :

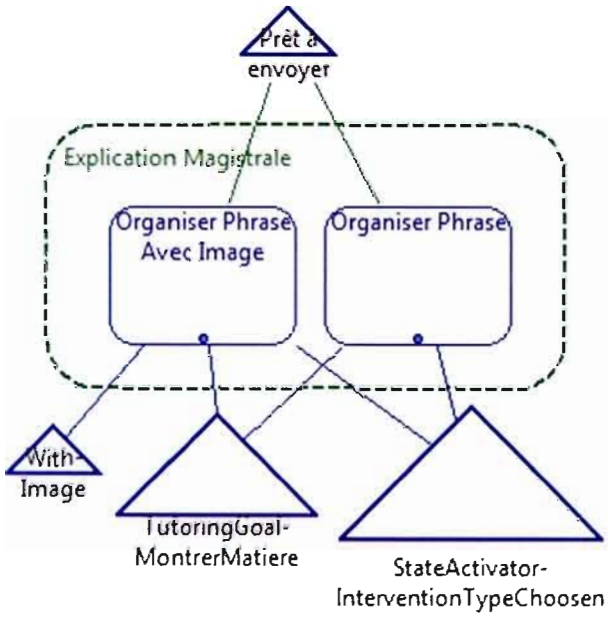
| Nom | Type du mpi | Valeur du mpi |
|---|---|---|
| Step-MontrerMatiere | Step | MontrerMatiere |
| Step-QuestionnerConcepts | Step | QuestionnerConcepts |
| Step-NoAnswerTimeOut | Step | NoAnswerTimeOut |
| Step-GoodAnswer | Step | GoodAnswer |
| Step-ExerciserConcepts | Step | ExerciserConcepts |
| Step-IndiquerConcept | Step | IndiquerConcept |
| Step-TransitionNiveau | Step | TransitionNiveau |
| Step-LienQuestionnaire | Step | LienQuestionnaire |
| Step-BadAnswer | Step | BadAnswer |
| PossibleBC-Explanation | PossibleBC | Explanation |
| PossibleBC-Teaching_The_Sublanguage | PossibleBC | Teaching_The_Sublanguage |
| PossibleBC-Probing_The_Student_Inference_Process | PossibleBC | Probing_The_Student_Inference_Process |
| PossibleBC-Help_In_Response_To_Pause | PossibleBC | Help_In_Response_To_Pause |
| PossibleBC-Get_The_Attention | PossibleBC | Get_The_Attention |
| PossibleBC-Acknoledgement | PossibleBC | Acknoledgement |
| PossibleDM-Monologue | PossibleDM | Monologue |
| PossibleDM-DirectedLineReasoning | PossibleDM | DirectedLineReasoning |
| PossibleDM-Hinting | PossibleDM | Hinting |
| PossibleDM-Rephrasing | PossibleDM | Rephrasing |
| PossiblePF-Declarative | PossiblePF | Declarative |
| PossiblePF-Interrogative | PossiblePF | Interrogative |
| PossiblePF-Fragement | PossiblePF | Fragment |
| PossiblePF-Imperative | PossiblePF | Inperative |
| StateActivator-InterventionTypeChoosen | StateActivator | InterventionTypeChoosen |




But : Cette séquence contient les méthodes qui peuvent être utilisées lorsque l'étudiant ne répond pas à une question.

États :

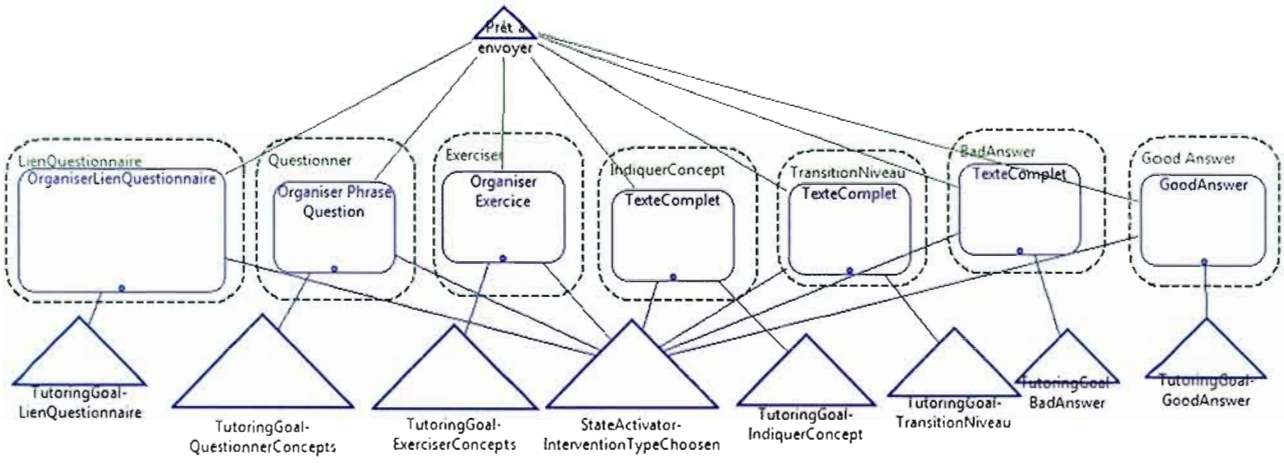
| Nom | Type du mpi | Valeur du mpi |
|---|---|---|
| Communicative_Goal-Help_In_Response_To_Pause | Communicative_Goal | Help_In_Response_To_Pause |
| Communicative_Goal-Get_The_Attention | Communicative_Goal | Get_The_Attention |
| TutoringGoal-NoAnswerTimeOut | TutoringGoal | NoAnswerTimeOut |
| StateActivator-InterventionTypeChoosen | StateActivator | InterventionTypeChoosen |
| Step-NoAnswerTimeOut | Step | NoAnswerTimeOut |
| Prêt à envoyer | Graphe Concept | Terminer |
| Problem-NoAnswer | Problem | NoAnswer |
| Right Answer | Answer_Correctness | True |
| Answer-False | Answer_Correctness | False |




But : Séquence contenant les méthodes pour donner des explications.

États :

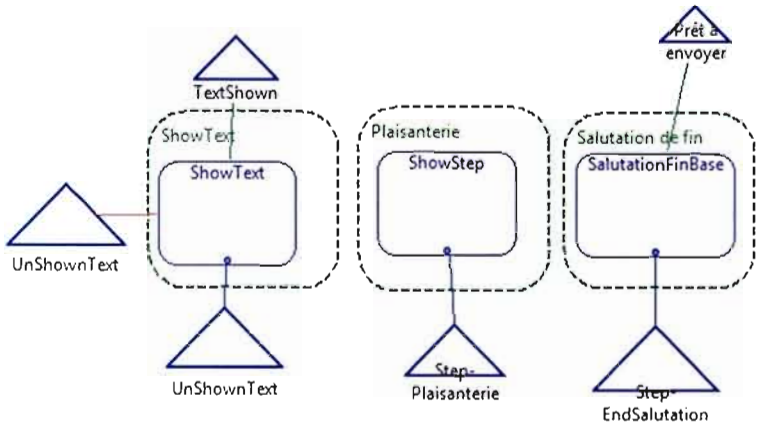
| Nom | Type du mpi | Valeur du mpi |
|---|---|---|
| TutoringGoal-MontrerMatiere | TutoringGoal | MontrerMatiere |
| StateActivator-InterventionTypeChoosen | StateActivator | InterventionTypeChoosen |
| Prêt à envoyer | Graphe Concept | Terminer |
| With-Image | With | Image |




But : Ces séquences contiennent les méthodes servant à exécuter leur but tutoriel respectif.

États :

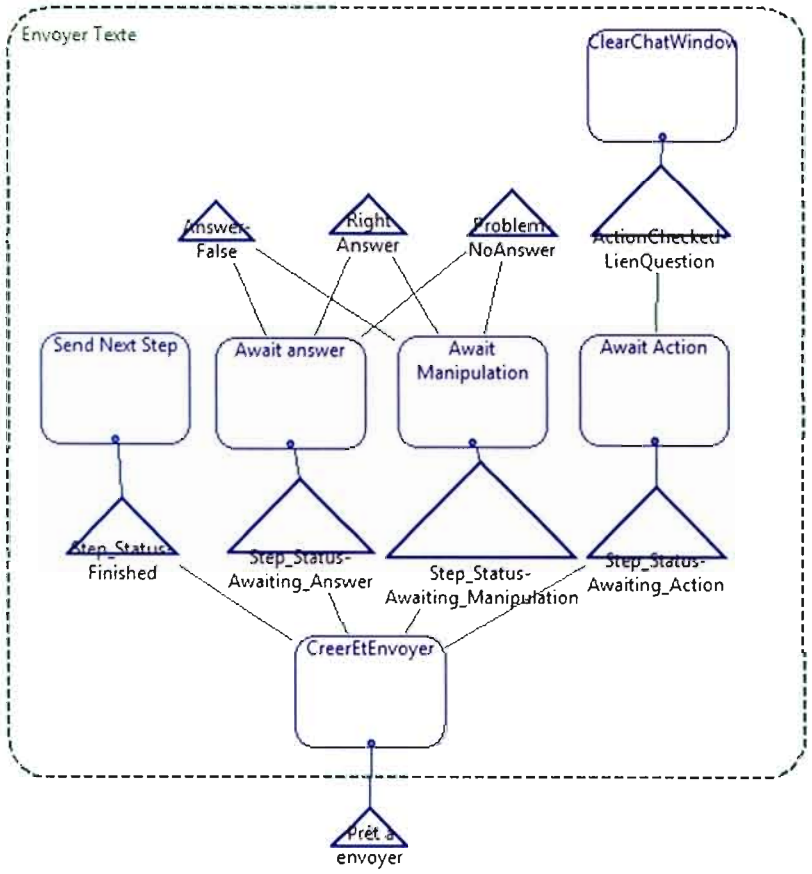
| **Nom** | **Type du mpi** | **Valeur du mpi** |
|---|---|---|
| TutoringGoal-LienQuestionnaire | TutoringGoal | LienQuestionnaire |
| TutoringGoal-QuestionnerConcepts | TutoringGoal | QuestionnerConcepts |
| TutoringGoal-ExerciserConcepts | TutoringGoal | ExerciserConcepts |
| TutoringGoal-IndiquerConcept | TutoringGoal | IndiquerConcept |
| TutoringGoal-TransitionNiveau | TutoringGoal | TransitionNiveau |
| TutoringGoal-BadAnswer | TutoringGoal | BadAnswer |
| TutoringGoal-GoodAnswer | TutoringGoal | GoodAnswer |
| StateActivator-InterventionTypeChoosen | StateActivator | InterventionTypeChoosen |
| Prêt à envoyer | Graphe Concept | Terminer |




But : ShowText sert à afficher dans la fenêtre de clavardage le texte entré par l'utilisateur, Plaisanterie est un représentant pour l'étape de plaisanterie et Salutation de fin est la méthode qui crée l'arbre pour une salutation de fin.

États :

| Nom | Type du mpi | Valeur du mpi |
|---|---|---|
| UnShownText | Text-Shown | false |
| TextShown | Text-Shown | true |
| Step-Plaisanterie | Step | ExerciserConcepts |
| Step-EndSalutation | Step | IndiquerConcept |
| Prêt à envoyer | Graphe Concept | Terminer |




But : Cette séquence contient tous les actes nécessaires à l'envoie d'un texte à l'utilisateur et au déclenchement des processus d'attente de réponses.

États :

| Nom | Type du mpi | Valeur du mpi |
|---|---|---|
| Step_Status-Finished | Step_Status | Finished |
| Step_Status-Awaiting_Answer | Step_Status | Awaiting_Answer |
| Step_Status_Awaiting_Manipulation | Step_Status | Awaiting_Manipulation |
| Step_Status-Awaiting_Action | Step_Status | Awaiting_Action |
| Problem-NoAnswer | Problem | NoAnswer |
| Right Answer | Answer_Correctness | True |
| Answer-False | Answer_Correctness | False |
| ActionChecked-LienQuestion | ActionChecked | LienQuestion |
| Prêt à envoyer | Graphe Concept | Terminer |

## B.3 Actes Opérationnels

Nous allons voir les différents actes opérationnels disponibles dans le système avec une courte description de leur but et la liste des Microprocessus avec une courte description composant chacun des actes. Note : lorsque le nom d'un Microprocessus est suivi d'un tiret, le mot suivant est son argument (exemple : nomMicroprocessus-Argument.)

| Acte : | MethodeSalutationBase |
|---|---|
| But : | Créer la tête de l'arbre mpi pour une salutation |
| Microprocessus d'action : | |
| MethodeSalutationBase | Créer la tête de l'arbre mpi pour une salutation |

| Acte : | MethodeSalutationTitreCivil |
|---|---|
| But : | Ajoute le concept Titre Civil à l'arbre mpi de la salutation |
| Microprocessus d'action : | |
| MethodeSalutationTitreCivil | Ajoute le concept *Titre Civil* à l'arbre mpi de la salutation |
| StateActivatorAdder-Nommer | Ajoute un mpi de type *StateActivator* et de valeur *Nommer* |
| StateActivatorRemover-Titre Civil | Retire le mpi de type *StateActivator* et de valeur *Titre Civil* |

| Acte : | MethodeSalutationNoTitreCivil |
|---|---|
| But : | Sert de transition lorsqu'il n'y a pas de titre civil prévu pour la salutation |
| Microprocessus d'action : | |
| StateActivatorAdder-Nommer | Ajoute un mpi de type *StateActivator* et de valeur *Nommer* |
| StateActivatorRemover-Titre Civil | Retire le mpi de type *StateActivator* et de valeur *Titre Civil* |

| Acte : | MethodeSalutationNom |
|---|---|
| But : | Ajoute le concept Nom à l'arbre mpi de la salutation |
| Microprocessus d'action : | |
| MethodeSalutationNom | Ajoute le concept *Nom* à l'arbre mpi de la salutation |
| TextFinishedAdder | Ajoute un mpi de type *Graphe Concept* et de valeur *Terminer* |
| StateActivatorRemover-Nommer | Retire le mpi de type *StateActivator* et de valeur *Nommer* |

| Acte : | MethodeSalutationMangerOption |
|---|---|
| But : | Sert de transition lorsqu'il n'y a pas de nom prévu pour la salutation |
| Microprocessus d'action : | |
| TextFinishedAdder | Ajoute un mpi de type *Graphe Concept* et de valeur *Terminer* |
| StateActivatorRemover-Nommer | Retire le mpi de type *StateActivator* et de valeur *Nommer* |

| Acte : | Créer Tete de Phrase |
|---|---|
| But : | Crée la tête de l'arbre mpi qui devra ensuite obtenir les buts communicatifs, modes de livraison et formes de phrase |
| Microprocessus d'action : | |
| CreerTetePhrase | Crée la tête d'un arbre mpi pour un phrase en y plaçant les informations reçues du planificateur, incluant le concept, le locuteur CTS et le but tutoriel |

| Acte : | But Communicatif Teaching_The_SubLanguage Choisi |
|---|---|
| But : | Ajoute le but communicatif Teaching_The_SubLanguage à l'arbre mpi |
| Microprocessus d'action : | |
| CommunicativeGoalAdder-Teaching_The_Sublanguage | Ajoute un mpi de type *Concept_Graphe_Property* et de valeur *Communicative_Goal-Teaching_The_Sublanguage* |
| InfoGatAdder-DeliveryMode | Ajoute un mpi de type *InformationGathering* et de valeur *DeliveryMode* |
| InfoGatRemover-CommunicativeGoal | Retire le mpi de type *InformationGathering* et de valeur *CommunicativeGoal* |

| Acte : | But Communicatif Explication Choisi |
|---|---|
| But : | Ajoute le but communicatif Explanation à l'arbre mpi |
| Microprocessus d'action : | |
| CommunicativeGoalAdder-Explanation | Ajoute un mpi de type *Concept_Graphe_Property* et de valeur *Communicative_Goal-Explanation* |
| InfoGatAdder-DeliveryMode | Ajoute un mpi de type *InformationGathering* et de valeur *DeliveryMode* |
| InfoGatRemover-CommunicativeGoal | Retire le mpi de type *InformationGathering* et de valeur *CommunicativeGoal* |

| Acte : | But Communicatif Probing_The_Student_Inference_Process Choisi |
|---|---|
| But : | Ajoute le but communicatif Probing_The_Student_Inference_Process à l'arbre mpi |
| Microprocessus d'action : | |
| CommunicativeGoalAdder-Probing_The_Student_Inference_Process | Ajoute un mpi de type *Concept_Graphe_Property* et de valeur *Communicative_Goal-Probing_The_Student_Inference_Process* |
| InfoGatAdder-DeliveryMode | Ajoute un mpi de type *InformationGathering* et de valeur *DeliveryMode* |
| InfoGatRemover-CommunicativeGoal | Retire le mpi de type *InformationGathering* et de valeur *CommunicativeGoal* |

| Acte : | But Communicatif Help_In_Response_To_Pause Choisi |
|---|---|
| But : | Ajoute le but communicatif Help_In_Response_To_Pause à l'arbre mpi |
| Microprocessus d'action : | |
| CommunicativeGoalAdder-Help_In_Response_To_Pause | Ajoute un mpi de type *Concept_Graphe_Property* et de valeur *Communicative_Goal-Help_In_Response_To_Pause* |
| InfoGatAdder-DeliveryMode | Ajoute un mpi de type *InformationGathering* et de valeur *DeliveryMode* |
| InfoGatRemover-CommunicativeGoal | Retire le mpi de type *InformationGathering* et de valeur *CommunicativeGoal* |

| Acte : | But Communicatif Get_The_Attention Choisi |
|---|---|
| But : | Ajoute le but communicatif Get_The_Attention à l'arbre mpi |
| Microprocessus d'action : | |
| CommunicativeGoalAdder-Get_The_Attention | Ajoute un mpi de type *Concept_Graphe_Property* et de valeur *Communicative_Goal- Get_The_Attention* |
| InfoGatAdder-DeliveryMode | Ajoute un mpi de type *InformationGathering* et de valeur *DeliveryMode* |
| InfoGatRemover-CommunicativeGoal | Retire le mpi de type *InformationGathering* et de valeur *CommunicativeGoal* |

| Acte : | But Communicatif Acknowledgement Choisi |
|---|---|
| But : | Ajoute le but communicatif Acknowledgement à l'arbre mpi |
| Microprocessus d'action : | |
| CommunicativeGoalAdder-Acknowledgement | Ajoute un mpi de type *Concept_Graphe_Property* et de valeur *Communicative_Goal- Acknowledgement* |
| InfoGatAdder-DeliveryMode | Ajoute un mpi de type *InformationGathering* et de valeur *DeliveryMode* |
| InfoGatRemover-CommunicativeGoal | Retire le mpi de type *InformationGathering* et de valeur *CommunicativeGoal* |

| Acte : | Delivery Mode Monologue Choisi |
|---|---|
| But : | Ajoute le mode de livraison Monologue à l'arbre mpi |
| Microprocessus d'action : | |
| DeliveryModeAdder-Monologue | Ajoute un mpi de type *Concept_Graphe_Property* et de valeur *Delivery_Mode-Monologue* |
| InfoGatAdder-PhraseForm | Ajoute un mpi de type *InformationGathering* et de valeur *PhraseForm* |
| InfoGatRemover-DeliveryMode | Retire le mpi de type *InformationGathering* et de valeur *DeliveryMode* |

| Acte : | Delivery Mode Directed_Line_Of_Reasoning Choisi |
|---|---|
| But : | Ajoute le mode de livraison Directed_Line_Of_Reasoning à l'arbre mpi |
| Microprocessus d'action : | |
| DeliveryModeAdder-Directed_Line_Of_Reasoning | Ajoute un mpi de type *Concept_Graphe_Property* et de valeur *Delivery_Mode- Directed_Line_Of_Reasoning* |
| InfoGatAdder-PhraseForm | Ajoute un mpi de type *InformationGathering* et de valeur *PhraseForm* |
| InfoGatRemover-DeliveryMode | Retire le mpi de type *InformationGathering* et de valeur *DeliveryMode* |

| Acte : | Delivery Mode Rephrasing Choisi |
|---|---|
| But : | Ajoute le mode de livraison Rephrasing à l'arbre mpi |

Microprocessus d'action :

| | |
|---|---|
| DeliveryModeAdder-Rephrasing | Ajoute un mpi de type *Concept_Graphe_Property* et de valeur *Delivery_Mode- Rephrasing* |
| InfoGatAdder-PhraseForm | Ajoute un mpi de type *InformationGathering* et de valeur *PhraseForm* |
| InfoGatRemover-DeliveryMode | Retire le mpi de type *InformationGathering* et de valeur *DeliveryMode* |

| Acte : | Delivery Mode Hinting Choisi |
|---|---|
| But : | Ajoute le mode de livraison Hinting à l'arbre mpi |

Microprocessus d'action :

| | |
|---|---|
| DeliveryModeAdder- Hinting | Ajoute un mpi de type *Concept_Graphe_Property* et de valeur *Delivery_Mode- Hinting* |
| InfoGatAdder-PhraseForm | Ajoute un mpi de type *InformationGathering* et de valeur *PhraseForm* |
| InfoGatRemover-DeliveryMode | Retire le mpi de type *InformationGathering* et de valeur *DeliveryMode* |

| Acte : | Phrase Form Interrogative Choisi |
|---|---|
| But : | Ajoute la forme de phrase Interrogative à l'arbre mpi |

Microprocessus d'action :

| | |
|---|---|
| PhraseFormAdder-Interrogative | Ajoute un mpi de type *Concept_Graphe_Property* et de valeur *Phrase_Form-Interrogative* |
| StateActivatorAdder-InterventionTypeChoosen | Ajoute un mpi de type *StateActivator* et de valeur *InterventionTypeChoosen* |
| InfoGatRemover-PhraseForm | Retire le mpi de type *InformationGathering* et de valeur *PhraseForm* |

| Acte : | Phrase Form Declarative Choisi |
|---|---|
| But : | Ajoute la forme de phrase Declarative à l'arbre mpi |

Microprocessus d'action :

| | |
|---|---|
| PhraseFormAdder-Declarative | Ajoute un mpi de type *Concept_Graphe_Property* et de valeur *Phrase_Form- Declarative* |
| StateActivatorAdder-InterventionTypeChoosen | Ajoute un mpi de type *StateActivator* et de valeur *InterventionTypeChoosen* |
| InfoGatRemover-PhraseForm | Retire le mpi de type *InformationGathering* et de valeur *PhraseForm* |

| Acte : | Phrase Form Fragment Choisi |
|---|---|
| But : | Ajoute la forme de phrase Fragment à l'arbre mpi |
| Microprocessus d'action : | |
| PhraseFormAdder- Fragment | Ajoute un mpi de type *Concept_Graphe_Property* et de valeur *Phrase_Form- Fragment* |
| StateActivatorAdder-InterventionTypeChoosen | Ajoute un mpi de type *StateActivator* et de valeur *InterventionTypeChoosen* |
| InfoGatRemover-PhraseForm | Retire le mpi de type *InformationGathering* et de valeur *PhraseForm* |

| Acte : | Phrase Form Imperative Choisi |
|---|---|
| But : | Ajoute la forme de phrase Imperative à l'arbre mpi |
| Microprocessus d'action : | |
| PhraseFormAdder- Imperative | Ajoute un mpi de type *Concept_Graphe_Property* et de valeur *Phrase_Form- Imperative* |
| StateActivatorAdder-InterventionTypeChoosen | Ajoute un mpi de type *StateActivator* et de valeur *InterventionTypeChoosen* |
| InfoGatRemover-PhraseForm | Retire le mpi de type *InformationGathering* et de valeur *PhraseForm* |

| Acte : | GoodAnswer |
|---|---|
| But : | Terminer l'arbre mpi pour une bonne réponse |
| Microprocessus d'action : | |
| StateActivatorRemover-InterventionTypeChoosen | Retire le mpi de type *StateActivator* et de valeur *InterventionTypeChoosen* |
| TextFinishedAdder | Ajoute un mpi de type *Graphe Concept* et de valeur *Terminer* |

| Acte : | TexteComplet |
|---|---|
| But : | Terminer l'arbre mpi pour une phrase de CTS |
| Microprocessus d'action : | |
| StateActivatorRemover-InterventionTypeChoosen | Retire le mpi de type *StateActivator* et de valeur *InterventionTypeChoosen* |
| TextFinishedAdder | Ajoute un mpi de type *Graphe Concept* et de valeur *Terminer* |

| Acte : | RepeatQuestion |
|---|---|
| But : | Permet de répéter une question posée |
| Microprocessus d'action : | |
| RepeatQuestion | Modifie le mpi *Graphe Concept-Envoyer* en *Graphe Concept-Terminer* dans l'arbre mpi de la question à retourner |

| Acte : | Remind we are waiting answer |
|---|---|
| But : | Terminer l'arbre mpi pour rappeler à l'étudiant qu'on attend une réponse |
| Microprocessus d'action : | |
| StateActivatorRemover-InterventionTypeChoosen | Retire le mpi de type *StateActivator* et de valeur *InterventionTypeChoosen* |
| TextFinishedAdder | Ajoute un mpi de type *Graphe Concept* et de valeur *Terminer* |
| RemindToAnswer | Ajoute un mpi *NeedAnswer* à l'arbre mpi |

| Acte : | Organiser Phrase Montrer Concept avec image |
|---|---|
| But : | Terminer l'arbre mpi pour donner une explication avec une image |
| Microprocessus d'action : | |
| StateActivatorRemover-InterventionTypeChoosen | Retire le mpi de type *StateActivator* et de valeur *InterventionTypeChoosen* |
| TextFinishedAdder | Ajoute un mpi de type *Graphe Concept* et de valeur *Terminer* |
| AjoutImage | Ajoute les mpi nécessaires pour fournir une image |
| OrganiserText | Ajoute les mpi nécessaires pour fournir une explication |

| Acte : | Organiser Phrase Montrer Concept |
|---|---|
| But : | Terminer l'arbre mpi pour donner une explication |
| Microprocessus d'action : | |
| StateActivatorRemover-InterventionTypeChoosen | Retire le mpi de type *StateActivator* et de valeur *InterventionTypeChoosen* |
| TextFinishedAdder | Ajoute un mpi de type *Graphe Concept* et de valeur *Terminer* |
| OrganiserText | Ajoute les mpi nécessaires pour fournir une explication |

| Acte : | SalutationFin |
|---|---|
| But : | Créer l'arbre mpi pour une salutation de fin de session |
| Microprocessus d'action : | |
| MethodeSalutationFinBase | Créer l'arbre mpi pour une salutation de fin de session |

| Acte : | Organiser lien Question |
|---|---|
| But : | Terminer l'arbre mpi pour transiger vers une question |
| Microprocessus d'action : | |
| StateActivatorRemover-InterventionTypeChoosen | Retire le mpi de type *StateActivator* et de valeur *InterventionTypeChoosen* |
| TextFinishedAdder | Ajoute un mpi de type *Graphe Concept* et de valeur *Terminer* |
| OrganiserLienQuestionnaire | Ajoute les mpi nécessaires pour créer le texte créant un lien vers les questions |

| Acte : | Organiser Phrase Question |
|---|---|
| But : | Terminer l'arbre mpi pour poser une question |
| Microprocessus d'action : | |
| StateActivatorRemover-InterventionTypeChoosen | Retire le mpi de type *StateActivator* et de valeur *InterventionTypeChoosen* |
| TextFinishedAdder | Ajoute un mpi de type *Graphe Concept* et de valeur *Terminer* |
| OrganiserTextQuestion | Ajoute les mpi nécessaires pour poser une question |

| Acte : | ShowText |
|---|---|
| But : | Afficher le texte entré par l'étudiant |
| Microprocessus d'action : | |
| SendText | Envoie le texte de l'étudiant vers la fenêtre de clavardage |
| DetermineConcept | Détermine le concept visé par le texte de l'étudiant |

| Acte : | CreerEtEnvoyer |
|---|---|
| But : | Terminer le graphe mpi de la phrase et le transmettre au module de génération de texte et l'afficher dans la fenêtre de clavardage |
| Microprocessus d'action : | |
| CreerEtEnvoyer | Terminer le graphe mpi de la phrase et le transmettre au module de génération de texte et l'afficher dans la fenêtre de clavardage |
| AfficherImage | Affiche l'image fournie à l'écran |

| Acte : | ShowStep |
|---|---|
| But : | Afficher le nom de l'étape en cours |
| Microprocessus d'action : | |
| ShowStep | Envoie le nom de l'étape en cours vers la fenêtre de clavardage |

| Acte : | Send Next Step |
|---|---|
| But : | Envoie une demande d'étape suivante |
| Microprocessus d'action : | |
| NextStepMpiLauncher | Crée et envoie un mpi *NextStep* |

| Acte : | ClearChatWindow |
|---|---|
| But : | Efface les fenêtres d'affichage et envoie une demande d'étape suivante |
| Microprocessus d'action : | |
| ClearChatWindow | Efface le contenu de la fenêtre de clavardage |
| HideImage | Retire la fenêtre d'affiche des images |
| NextStepMpiLauncher | Crée et envoie un mpi *NextStep* |

| Acte : | Await Answer |
|---|---|
| But : | Lance l'attente d'une réponse de l'utilisateur |
| Microprocessus de confirmation : | |
| WaitingAnswer | Observe les mpi reçus dans le système pour trouver une réponse de l'utilisateur à la question posée |

| Acte : | Await Action |
|---|---|
| But : | Lance l'attente d'une action de l'utilisateur |
| Microprocessus de confirmation : | |
| AwaitAction | Observe les mpi reçus dans le système pour trouver une action de l'utilisateur |

| Acte : | Await Manipulation |
|---|---|
| But : | Lance l'attente d'une manipulation de l'utilisateur |
| Microprocessus de confirmation : | |
| WaitingManipulation | Observe les mpi reçus dans le système pour trouver une manipulation de l'utilisateur correspondant à la question posée |

# REFERENCES


Aleven, V., Popescu, O. and Koedinger, K.R. (2001). A tutorial dialogue system with knowledge-based understanding and classification of student explanations. In: *Second IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*.. Seattle, WA

Baars, B.J., Franklin S. (2003). How conscious experience and working memory interact » (2003) *Trends in Cognitive Sciences,* 7 (4), pp. 166-172

Callaway, C. B.; Dzikovska, M.; Farrow, E.; Marques-Pita, M.; Matheson, C.; and Moore, J. D. (2007). The Beetle and BeeDiff tutoring systems. In *Proceedings of the SLaTE-2007 Workshop*

Core, M. G., Moore, J. D. and Zinn, C. (2003). The role of initiative in tutorial dialogue. In *Proc. European Chap. Assoc. Computational Linguistics*.

Franklin, S. (1999). Action Selection and Language Generation in "Conscious" Software Agents. In: *Proc. Workshop on Behavior Planning for Life-Like Characters and Avatars*, i3 Spring Days '99, Sitges, Spain

Franklin, S. (2000). Deliberation and voluntary action in 'conscious' software agents. *Neural Network.* 10, pp. 505–521

Franklin, S. (2005b). Perceptual Memory and Learning: Recognizing, Categorizing, and Relating. *Symposium on Developmental Robotics. American Association for Artificial Intelligence (AAAI)*. Stanford University, Palo Alto CA, USA. March 21-23, 2005

Franklin, S., and McCauley, L.. (2003). Interacting with IDA. In: *Agent Autonomy*, H. Hexmoor, C. Castelfranchi, and R. Falcone (eds). Dordrecht: Kluwer

Freedman, R. (2001).An approach to increasing programming efficiency in plan-based dialogue systems. In: *Proceedings of the 10th International Conference on AI in Education (AIED 2001)*, Amsterdam, IOS Press.

Freedman, R. and Evens, M. W. (1996). Generating and Revising Hierarchical Multi-Turn Text Plans in an ITS. In: *Intelligent Tutoring Systems: Third International Conference ITS '96, Montreal, Quebec, pp. 632–640*

Freedman, R., Penstein Rosé, C., Ringenberg, M., & VanLehn, K. (2000). ITS tools for natural language dialogue: A domain-independent parser and planner. In G. Gauthier, C. Frasson, & K. VanLehn (Eds.), *Proceedings of the ITS* 2000, pp. 433–442, Montreal, LNCS 1839. Berlin: Springer.

Graesser, A. C., Lu, S., Jackson, G. T., Mitchell, H., Ventura, M., Olney, A., & Louwerse, M. M. (2004). AutoTutor: A tutor with dialogue in natural language. *Behavioral Research Methods, Instruments, and Computers,* 36, 180-193Woo, C.W., Evens, M.W., Freedman, R., Glass, M., Shim, L.S., Zhang, Y., Zhou, Y. and Michael, J.A. (2006).An intelligent tutoring system that generates a natural language dialogue using dynamic multi-level planning. *Artificial Intelligence in Medicine,* 38(1), 25-46

Graesser, A.C., VanLehn, K., Rose, C., Jordan, P., and Harter, D. (2001). Intelligent tutoring systems with conversational dialogue. *AI Magazine, 22*, 39-51

Jordan, P. W., Hall, B., Ringenberg, M., Cui, Y. and Rose, C.P. (2007). Tools for authoring a dialogue agent that participates in learning studies. In: *Proceedings of AIED 2007*, IOS press.

Jordan, P. W.; Rosé, C. P.; and VanLehn, K (2001). Tools for Authoring Tutorial Dialogue Knowledge. In J. D. Moore, C. L. Redfield, & W. L. Johnson (Eds.), *Artificial Intelligence in Education: AI-ED in the Wired and Wireless Future, Proceedings of AI-ED 2001*. Amsterdam, IOS Press

Jordan, P., Makatchev, M., Pappuswamy, U., VanLehn, K and Albacete, P. (2006). A natural language tutorial dialogue system for physics. In: *Proceedings of the 19th International FLAIRS conference*, AAAI press..

Kim, J.H., Freedman, R., Glass, M. and Evens, M.W. (2006). Annotation of Tutorial Dialogue Goals for Natural Language Generation. *Discourse Processes,* 42 (1), pp. 37-74.

Maes, P. (1989). How to do the right thing. *Connection Science,* 1, 291-323

Rickel, J., Lesh, N., Rich, C., Sidner, C. L., & Gertner, A. S. (2002). Collaborative discourse theory as a foundation for tutorial dialogue . In S. A. Cerri, G. Gouarderes, & F. Paraguacu (Eds.), *Proceedings of the Sixth International Conference on Intelligent Tutoring Systems*, pp. 542-551. Berlin: Springer-Verlag

Tyrrell, T. (1994). An Evaluation of Maes's Bottom-Up Mechanism for Behavior Selection, *Adaptive Behavior, 2 (4)* , pp. 307 348

Woo'C.W., Evens, M.W., Freedman, R., Glass, M., Shim, L.S., Zhang, Y., Zhou, Y. and Michael, J.A. (2006). An intelligent tutoring system that generates a natural language dialogue using dynamic multi-level planning. *Artificial Intelligence in Medicine,* 38(1), 25-46

Zinn, C., Moore, J.D., Core, M.G. (2006). A 3-Tier Planning Architecture for Managing Tutorial Dialogue. *In: Proceedings of the 6th International Conference on Intelligent Tutoring Systems*, p.574-584, June 02-07, Taiwan.