THÈSE

présentée
à l'UFR de Sciences pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DU MAINE

Spécialité : Sciences
Mention : Informatique

par

THIERRY LEMEUNIER

L'intentionnalité communicative dans le dialogue homme-machine en langue naturelle

Soutenue le 1er décembre 2000 devant le jury composé de :

Pierre Tchounikine	LIUM, université du Maine	codirecteur
Daniel Luzzati	LIUM, université du Maine	codirecteur
Anne Nicolle	GREYC CNRS-UPRESA 6072, université de Caen	rapporteur
Violaine Prince	LIRMM CNRS, université de Montpellier II	rapporteur
Christian Brassac	LPI-GRC, université de Nancy	examinateur
Gérard Sabah	LIMSI-CNRS, Orsay	examinateur

Thèse co-dirigée par Martial Vivet.

Thèse préparée au sein du Laboratoire d'Informatique de l'Université du Maine

THÈSE

présentée
à l'UFR de Sciences pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DU MAINE

Spécialité : Sciences
Mention : Informatique

par

THIERRY LEMEUNIER

L'intentionnalité communicative dans le dialogue homme-machine en langue naturelle

Soutenue le 1er décembre 2000 devant le jury composé de :

Pierre Tchounikine	lium, université du maine	codirecteur
Daniel Luzzati	lium, université du maine	codirecteur
Anne Nicolle	greyc cnrs-upresa 6072, université de caen	rapporteur
Violaine Prince	lirmm cnrs, université de montpellier ii	rapporteur
Christian Brassac	lpi-grc, université de nancy	examinateur
Gérard Sabah	limsi-cnrs, orsay	examinateur

Thèse co-dirigée par Martial Vivet.

Thèse préparée au sein du Laboratoire d'Informatique de l'Université du Maine

Remerciements

Ma première pensée va à Martial Vivet. Grâce à lui j'ai pu obtenir une bourse de thèse, mais bien plus encore. Ses cours que j'ai suivi en maîtrise puis en DEA m'ont donné le goût de la recherche.

Comment remercier Daniel Luzzati de m'avoir fait confiance en acceptant de tenter la chance avec moi dans le périlleux chemin de la thèse ? Sa philosophie et sa psychologie m'ont souvent été d'un grand secours.

Je remercie Anne Nicolle pour ses remarques pertinentes et toujours claires. Merci de m'avoir aidé à progresser dans mon travail.

Je remercie Jérôme Lehuen. Compagnon quotidien de travail. Témoin actif. Puissions-nous encore travailler souvent ensemble à l'avenir.

J'adresse mes sincères remerciements à Violaine Prince pour son rapport de pré-soutenance qu'elle a accepté de rédiger. Ses remarques et ses critiques m'ont donné une vision plus claire de mon travail.

Je remercie Gérard Sabah et Christian Brassac pour avoir accepté d'être membre du jury de ma soutenance de thèse. Leurs travaux m'ont été d'une grande inspiration et j'espère ne pas avoir trop déformé leurs pensées.

Je remercie également Pierre Tchounikine d'avoir accepté de prendre "au pied levé" le relais de Martial Vivet comme codirecteur de thèse.

Le LIUM fait partie de ces laboratoires où les thésards sont aidés, suivis, écoutés mais également laissés libre de développer et d'exprimer leurs propres idées. Je remercie toutes les personnes du LIUM qui ont contribué directement ou indirectement à cette thèse.

Thierry Lemeunier
le 17 octobre 2000
Le Mans

À mes parents qui ont tant sacrifié pour leurs enfants.

À Martial Vivet qui m'a ouvert les voies de la recherche.

Table des matières

Introduction

Pourquoi dialoguer en langue naturelle avec une machine

La communication homme-machine regroupe deux thèmes de recherche : les interfaces homme-machine et les dialogues homme-machine. Habituellement, nous associons à chacun des deux modes de communication un spectre d'applications relativement bien défini. Il semble évident en effet qu'utiliser une langue naturelle a peu d'intérêt quand la manipulation directe d'une interface suffit à rendre l'interaction efficace. Cependant, au-delà de ce simple constat, l'usage des langues naturelles nous semble plus intéressant pour ce qui concerne l'adaptation de la machine au fonctionnement cognitif des utilisateurs.

D'un côté les interfaces homme-machine sont généralement des systèmes graphiques qui permettent l'interaction par manipulation du clavier et d'un pointeur. L'utilisateur doit se livrer à un travail d'auto-interprétation avant et pendant l'interaction, pour savoir ce qu'il doit faire. Sa *charge cognitive* peut être réduite en utilisant des métaphores pour accéder aux ressources logicielles et matérielles, elles-mêmes figurées sous formes d'objets informatiques.

De l'autre côté les systèmes de dialogue homme-machine s'appuient sur la langue naturelle pour échanger les informations avec les utilisateurs. Les humains interagissent « naturellement » dans cette modalité. *A priori*, il semble raisonnable de penser qu'en instaurant une interaction en langue naturelle, ils pourront communiquer plus librement avec la machine. En laissant à la machine le travail d'interprétation de leurs énoncés, et, à un niveau supérieur, de reconnaissance de leurs intentions, la charge cognitive des utilisateurs est diminuée d'autant. C'est en cela que réside l'intérêt principal du langage naturel en communication : dialoguer en langue naturelle demande peu d'effort conscient aux utilisateurs humains, la charge cognitive étant en quelque sorte déplacée de l'usager vers la machine.

La langue naturelle est à la fois et souple et puissante. En effet, en général, un homme exprime mieux ses pensées par ses mots que par ses mains[1], la capacité d'expressivité de ce dernier mode d'interaction étant moindre. Sabah affirme ainsi que « les modes de communication classiques mais peu flexibles [...] peuvent être remplacés par un véritable dialogue coopératif utilisant la langue naturelle, seul moyen qui permet des auto-références et une modification dynamique de la situation de communication. » [Sabah *et al.*, 1997].

[1] Le langage des signes est un cas particulier pour pallier une carence auditive et/ou locutoire.

Ainsi, si nous prenons le sens général qui est celui *d'assurer un transfert entre deux systèmes indépendants nécessaires à leur fonctionnement respectif*, l'interface dans le mode de la manipulation directe est figée puisque tout est prévu à l'avance et qu'on ne peut pas en sortir. En revanche, dans un dialogue en langue naturelle, l'interface n'est pas figée, et même plus, elle se construit au cours du dialogue. En combinant les deux modes d'interactions (et même plusieurs autres dans le cas des interfaces *multimodales*), l'interfaçage est bien souvent meilleur (voir le concept d'*hyperdialogue* dans [Rouillard, 2000]).

Cependant, réaliser un système de dialogue homme-machine tirant réellement partie de la souplesse des langues naturelles relève encore pour une grande part de la théorie, car la mise en pratique de ces idées est loin d'être chose faite tant les complexités conjuguées de la langue et du dialogue sont importantes. Des progrès importants ont pourtant été réalisés en ce qui concerne les dialogues finalisés[2], bien différenciées par les chercheurs des conversations spontanées (voir par exemple [Dessalles, 1996]). Pourtant, il faut bien noter que l'utilisation de la langue naturelle ne deviendra réellement intéressante pour l'utilisateur qu'à partir du moment où tous ses avantages seront effectivement garantis. Cela passe, par exemple, par la possibilité de traiter les métaphores [Ferrari, 1997], de prendre en compte la polysémie naturelle de la langue [Prince, 1990] et mieux, de pouvoir générer automatiquement des métaphores explicatives [Prince & Ferrari, 1996].

Les trois types de système de dialogue

Luzzati a proposé une typologie des types d'applications du dialogue homme-machine [Luzzati, 1996]. Cette typologie est fondée sur les types de relations entre l'utilisateur et la machine pour chaque type d'application : elle peut être du type *maître-esclave*, du type *maître-serviteur*, ou du type *maître-maître*. En reprenant cette typologie, nous pouvons préciser les différents types d'interactions qui résultent de chacun des trois types d'applications proposés par Luzzati. Nous avons résumé tout ceci Tab. 1.

Pour une application du type consultation d'une base de données, la relation entre l'utilisateur et le système est du type *maître-esclave*. Les buts des deux interactants sont différents mais compatibles. L'utilisateur veut acquérir un certain nombre d'informations qu'il sait être connues du système. Le but de l'utilisateur, toujours identique, est connu d'avance par le système. Il n'y a donc pas de négociation sur les intentions de l'utilisateur. Le système doit lui fournir ces informations. L'interaction est coopérative — chacun essaie d'aider l'autre pour atteindre son but personnel —, et unidirectionnelle car l'information principale circule uniquement du système vers l'utilisateur. Comme illustration de ce type d'application nous pouvons citer des systèmes développés il y a quelques années : par exemples le système PARTNER

[2] Il faut noter que les principaux progrès en matière de dialogues finalisés proviennent du développement des modèles fondés sur la planification. Ces modèles *semblent* adaptés à ce genre de dialogue. Nous y reviendrons aux cours des chapitres suivants.

Types d'applications	Relation utilisateur-système	Type d'interaction
Consultation d'une base de données Ex. : Renseignements horaires SNCF.	maître-esclave	coopérative unidirectionnelle
Consultation d'une base sémantique Ex. : Pages jaunes de l'annuaire.	maître-serviteur	coopérative unidirectionnelle ou bi-directionnelle avec apprentissage sémantique
Système de conception collaborative Ex. : Programmation en langage naturel.	maître-maître	collaborative bi-directionnelle avec apprentissages pragmatique et sémantique possibles

TAB. 1 – Typologie des applications s'appuyant sur un dialogue.

[Morin & Pierrel, 1987] et le système SUNDIAL [Peckham, 1991].

Pour une application dont le principe équivaut à la consultation d'une base sémantique, la relation est du type *maître-serviteur*. Cette fois-ci encore les buts sont différents mais celui de l'utilisateur n'est pas forcément exprimé clairement : il a des intentions que le système doit essayer de reconnaître pour répondre à sa demande. L'interaction est coopérative — les buts réciproques sont compatibles —, et unidirectionnelle. Elle peut être bi-directionnelle s'il y a un apprentissage lexico-sémantique. Celui-ci peut être automatique si le système est capable de poser des questions sur les mots ou les expressions qui ne font pas partis de son vocabulaire comme dans le système COALA [Lehuen, 1997b], ou simplement manuel comme dans le système HALPIN [Rouillard, 1998].

Enfin, pour l'interaction avec un système de conception collaborative, la relation est plutôt du type *maître-maître*. L'utilisateur se borne à donner des directives et aide le système à faire certains choix. Lui et le système sont d'un même niveau de compétence dans leur domaine respectif et doivent collaborer pour atteindre un but commun. L'interaction est bi-directionnelle : les informations transitent indifféremment de l'un vers l'autre interactant. Le système (comme l'utilisateur) est capable d'apprendre soit par expérience soit à l'aide de l'utilisateur.

Alors que les premiers systèmes de dialogues (essentiellement oraux) étaient orientés vers la reconnaissance et la synthèse de la parole, les systèmes actuels, et le système que nous avons développé en particulier, se situent maintenant plus clairement dans une problématique de compréhension des mécanismes spécifiques du dialogue et de l'intercompréhension. Ils tiennent mieux compte des résultats des sciences humaines (psychologie cognitive, psycholinguistique, pragmatique), et sont soit du second type ou soit à la frontière entre le second et le troisième type (voir [Lehuen, 1997b] [Villaseñor & Caelen, 1998] [Fraczak *et al.*, 1998] [Charnay, 1999] [Quignard, 2000]).

Travail de thèse

Nos travaux ont débuté par la réalisation d'un premier système que nous avons aussitôt testé. Ce qui nous a frappé de prime abord, ce furent ses faibles capacités à réagir aux « non-attendus ». Tout ce que ce système pouvait répondre dans ce cas est la fameuse et fatidique phrase « *je ne comprends pas ce que vous voulez dire.* » Par exemple, dans le dialogue de la Figure 1, le système SOCRATE que nous avons développé durant notre stage de DEA [Lemeunier, 1996], est incapable de répondre à l'utilisateur quand il lui pose des questions sur ses capacités (« *que peux-tu faire pour moi* », « *Quels services proposes-tu exactement ?* »). Le système SOCRATE n'est pas capable de parler de lui-même, et nous avons essayé d'apporter des réponses humoristiques à ce genre de situation (« *Attendez, je dois avoir un train de retard !* », « *Hum, j'ai dû me tromper de voie !* », *etc.*)

M : Service de renseignement bonjour. Quel renseignement désirez-vous obtenir ?
H : que peux-tu pour moi ?
M : Attendez, je dois avois un train de retard !
H : Je ne comprends pas ce que vous voulez dire !
H : Quels services proposes-tu exactement ?
M : Hum, j'au dû me tromper de voie !
M : Je ne comprends pas ce que vous voulez dire !
H : es-tu une borne SNCF ?
M : Ne nous énervons pas et récapitulons.
M : A quelle heure désirez-vous partir ?
H : d'accord maintenant parceque tu ne comprends rien
M : Une minute, j'ai dû dérailler !
M : Je ne comprends pas ce que vous voulez dire !
H : bye ciao a plus j'me casse
M : Le service renseignement vous remercie de votre visite. Au revoir.

FIG. 1 – Exemple de dialogue avec le système SOCRATE en cas de non-compréhension.

Nous avons également observé certaines limitations de ce genre dans les systèmes de dialogue homme-machine plus récents[3]. Par exemple, dans le système HALPIN de Rouillard [Rouillard, 2000], nous avons relevé le dialogue de la Figure 2. Le système n'est plus capable de répondre dès que l'utilisateur ne suit pas ce qui a été prévu à la conception (« *quel temps fait-il à grenoble* », « *les voila ! ne vous l'avais-je point prédit ?* »).

Ces exemples montrent que nos systèmes ont d'importantes lacunes en matière de capacité d'interactions, car une bonne capacité dialogique doit permettre de poursuivre le dialogue même en cas d'imprévus. De manière plus générale, ces exemples illustrent bien le problème de la **rigidité dialogique**. Nous pouvons observer que les systèmes de dialogue homme-

[3] En fait, nous n'avons pas beaucoup trouvé d'exemples de dialogues montrant les limites des systèmes de DHM actuels ou passés, car les auteurs ne présentent pas spontanément les lacunes de leurs systèmes mais plus volontairement leurs points forts. D'autre part, il y a très peu de systèmes de DHM qui ont été ouverts à un large public comme l'a été celui de Rouillard via le Web. Ce qui explique que, dans ce cas précis, les limites du système aient été mises en lumière plus directement.

(...)
M : Voulez-vous effectuer une recherche selon ce(s) critère(s) ?
H : et si je dis non ?
M : D'accord, on ne fait pas de recherche pour l'instant.
M : Dialoguons ensemble encore un peu...
H : quel temps fait-il à grenoble ce soir ?
M : Désolé, je n'arrive pas à saisir vos propos.
H : décidément, vous voulez dialoguer mais vous l'avez pas
de conversation (à part les mots-clefs compulsifs...)
M : Les mots-clefs de ce livre sont : Victime; Victimologie; Victimation;
Indemnisation; Peine; Aide aux victimes; Entourage de la victime;
Préjudice; Médiation
M : Voulez-vous effectuer une recherche selon ce(s) critère(s) ?
R : les voila ! ne vous l'avais-je point prédit ?
M : Décidement, je ne comprends toujours pas ce que vous me dites.
(...)

FIG. 2 – Exemple de dialogue avec le système HALPIN en cas d'imprévus.

machine existants sont trop dépendants de l'application, car en dehors de celle-ci, ils ne sont plus capables de dialoguer.

Avec notre thèse, nous avons tenté d'apporter une réponse à la rigidité dialogique en cherchant à augmenter l'**autonomie de l'activité dialogique** vis-à-vis de l'activité applicative. Nous allons dans ce dessein proposer un modèle des **intentions de communications** pour un agent artificiel dialoguant en langue naturelle.

Présentation du document

Ce document est organisé en deux parties. La première partie présente notre modèle de la génération des intentions de communication. Elles sont calculées dynamiquement à partir de représentations présentes dans un terrain commun, c'est-à-dire dans une mémoire de représentations co-construites par les interactants. Ces dernières proviennent de l'interprétation des énoncés de l'utilisateur en termes d'actes de langages et de connaissances inférées par le système. Nous avons définis un ensemble de 11 configurations remarquables réparties en quatre classes d'équivalences, à partir desquelles une sorte de reconnaissance de forme est effectuée. Ce principe de création des intentions communicatives, à l'origine des actions langagières du système, est général et indépendant de la tâche. Il s'appuie uniquement sur la forme structurelle des éléments mémoriels et sur le statut de ces éléments.

La première partie est organisée comme suit. Dans le premier chapitre, nous approfondissons la discussion initiée dans cette introduction sur notre problématique de recherche. Nous essayons d'y mettre à jour ce qui a motivé notre thèse : comment parvenir à un authentique dialogue homme-machine grâce à une meilleure capacité dialogique. Dans le second chapitre, nous présentons quelques travaux qui ont influencé et nourri notre travail, et sur lesquels nous

nous appuyons pour fonder notre modèle des intentions de communications. Ce dernier est présenté en détail au troisième chapitre. Il débute par le développement des idées théoriques qui fondent ce modèle.

La seconde partie est plus pratique. Nous y décrirons une implémentation informatique de notre modèle dans le système de dialogue AMI (Agent de Messagerie Interactif). Nous présentons la méthode de développement adoptée, puis nous essayons de justifier l'architecture logicielle retenue. Nous finissons par présenter le système AMI2 et par donner quelques exemples de dialogues menés avec cette seconde version de l'application. Nous concluons en présentant les apports et les limites du modèle proposé.

Deux annexes achèvent ce document : l'annexe A présente l'analyse détaillée du corpus AMI1 et l'annexe B est le listing complet des fichiers sources du système AMI2.

Chapitre 1

Problématique

Notre travail de recherche est une étude de l'intentionnalité dans les interactions entre agents humains et agents artificiels, et en particulier lorsque celles-ci se font sous la forme précise de dialogues écrits en langue naturelle. Cette étude est le résultat d'un questionnement initial sur ce que recouvre réellement le dialogue homme-machine.

Dans un dialogue, il faut considérer à la fois la forme qu'il prend, comment il se déroule, mais aussi les participants au dialogue, comment ils se font comprendre, et comment leurs interprétations influent sur le déroulement du dialogue. Comme le fait remarquer Vernant, il existe un aspect externe, dialogal*, et un aspect interne,* dialogique *[Vernant, 1997b]. Notre approche nous a amené à considérer exclusivement l'aspect dialogique, c'est-à-dire la dimension cognitive d'un dialogue, afin de doter un agent artificiel d'une capacité à dialoguer avec un humain.*

1.1 Conditions du dialogue homme-machine

Notre étude débute par l'examen de plusieurs systèmes, un premier en maîtrise d'Informatique, un second en DEA [Lemeunier, 1996], puis un troisième en début de thèse [Lemeunier, 1998a]. Ces trois systèmes étaient simples et limités. Avec ces premiers travaux, nous avons mesuré combien la réalisation d'un système de dialogue homme-machine est complexe si l'on veut aller au-delà des simples suites de question-réponse, et nous avons dû abandonner une vision naïve des problèmes posés. Notre compréhension du domaine s'en est trouvée modifiée, et peu à peu s'est dégagée une question centrale à l'origine de notre thèse : dans quelles conditions peut s'établir un dialogue entre un agent humain et un agent artificiel vraiment satisfaisant ?

Nous allons essayer, dans les paragraphes suivants, de dégager les préalables de la réponse à cette question à partir des travaux antérieurs. Nous allons pour cela d'abord tenter de définir le dialogue homme-machine. Fort de cette définition, nous essaierons de cerner les conditions de

son établissement en faisant intervenir la notion de terrain commun. Cela nous amènera à poser notre hypothèse de recherche relative à la construction d'un terrain commun au cours d'un dialogue. Muni de cette hypothèse, nous essaierons de voir comment garantir la construction d'un terrain commun en évoquant les travaux les plus marquants du domaine. Nous finirons par un résumé des éléments intéressants pour notre propos.

Tout d'abord qu'entendons-nous par « dialogue » ? Il y a une différence entre *dialoguer* et *communiquer*. Le dialogue est une manière particulière de communiquer. La communication peut prendre d'autres formes, langagières ou non langagières. Lors d'un dialogue, les participants interagissent en temps réel les uns sur les autres. La communication est faite s'il y a compréhension mutuelle des interactants. Peut-on dire d'un agent artificiel qu'il *dialogue* avec un agent humain ? Autrement dit, la définition précédente du dialogue est-elle valable lorsque les participants sont un homme et une machine ? Cette question est loin d'être triviale. *A priori*, tel que nous l'avons défini, la condition porte sur l'établissement d'une inter-compréhension entre les participants. Cette condition peut-elle être vérifiée lorsque les participants sont de nature différente ?

Il semble bien que le mot dialogue appliqué à la communication homme-machine ne puisse pas recouvrir la même signification que celle du dialogue naturel tout simplement parce que le dialogue homme-machine met en présence, d'un côté, un être d'origine naturelle, doté d'une capacité spontanée et naturelle à comprendre ce qui l'entoure, et, de l'autre, un être d'origine artificielle dont la capacité de compréhension n'est en rien comparable à celle du premier. Il y a donc dès le départ un hiatus insurmontable et le lecteur gardera bien à l'esprit cette importante différence. Nous devons dès à présent sortir de l'idée que le dialogue homme-machine doit être et peut être identique au dialogue homme-homme car il s'agit-là d'une véritable chimère. Il faut au contraire créer de toute pièce un usage et observer ce qui se passe avec des systèmes réels en laissant de côté les dialogues du type Magicien d'Oz. En conséquence, il ne s'agira pas, pour nous, d'étudier les conversations humaines pour elles-mêmes, mais de nous informer sur le fonctionnement de l'homme en situation de communication. Ces informations nous sont en effet précieuses puisque l'un des partenaires du DHM est précisément un homme, dans toute sa complexité.

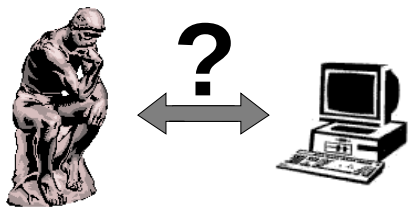


FIG. 1.1 – Une relation à définir.

Cette première distinction étant faite[1], il reste à étudier comment ce type de dialogue inédit peut quand même « fonctionner ». Comme le fait remarquer Vivier [Vivier, 1996, p. 14], « le problème est en quelque sorte de penser des modes de coopération avec un interlocuteur virtuel qui n'a pas l'air d'un humain. »

Examinons les éléments constitutifs d'un dialogue homme-machine, et permettons-nous une tautologie. L'informaticien peut intervenir sur le fonctionnement de la machine. En re-

[1]Nous ne sommes pas seul à faire cette distinction. Par exemple, on pourra lire à ce propos le livre de Luzzati [Luzzati, 1995b], où est évacué dès l'introduction le cliché d'une conversation à bâtons rompus avec un ordinateur omniscient.

vanche, il ne peut pas intervenir directement sur la manière dont « fonctionne » l'utilisateur.

Les études faites en psychologie cognitive ont peu abordé le fonctionnement des interactions conversationnelles [Vivier, 1996]. Il est vrai que l'objet d'étude est complexe tant le nombre de paramètres devant être pris en compte est important. Il faudrait faire intervenir de nombreuses disciplines, non seulement la psycholinguistique, mais aussi la psychologie cognitive, la psychologie sociale et la psychologie développementale. Toutefois, en psychologie cognitive, les résultats disparates dessinent les grands traits caractéristiques des interactions conversationnelles. Ainsi, il semble certain que, lors d'un dialogue naturel, tout homme adulte normal se construit différentes représentations mentales :

- une représentation de son interlocuteur ;
- une représentation de lui-même ;
- et une représentation d'un terrain commun.

Cette dernière notion a été plus particulièrement étudiée. Il s'agit « d'un ensemble de connaissances, hypothèses et croyances partagées par les partenaires, et présumées telles par chacun. » [Caron, 1997].

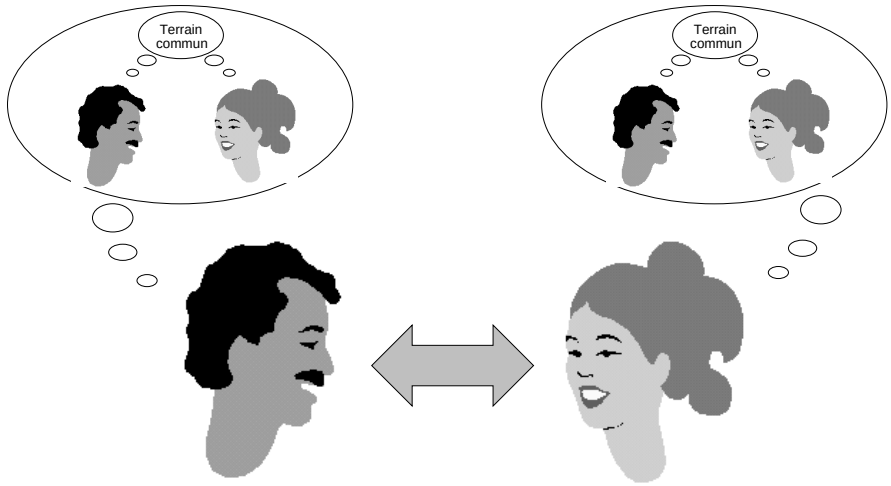


FIG. 1.2 – Les représentations mentales des interactants.

Des expériences menées selon le protocole dit du *Magicien d'Oz* corroborent ces résultats. Vivier (op. cité) relate une simulation de dialogue homme-machine mené par Morel[2] ; à partir de son étude, Luzzati a proposé un modèle dynamique du dialogue homme-machine [Luzzati, 1995b]. Les voyageurs s'informent par téléphone des horaires SNCF. L'hôtesse les prévient qu'ils vont être mis en relation avec un ordinateur, qui est en réalité simulé par l'hôtesse grâce à un filtrage de sa voix. D'après Vivier, les comportements perturbés des voyageurs (problèmes d'adaptation à leur nouvel interlocuteur) proviennent précisément de la représentation qu'ils se font de la machine. Les résultats sont identiques dans une étude menée à l'INRIA [Amal-

[2] M.A. Morel, 1988, *Dialogue homme-machine, premier corpus*, Paris, Publications de la Sorbonne nouvelle.

berti *et al.*, 1989]. Deux groupes de six sujets doivent obtenir des renseignements aériens en dialoguant avec un interlocuteur distant. Les sujets du premier groupe pensent parler à un ordinateur, tandis que les sujets du second groupe pensent parler à un humain. Les auteurs concluent par : « les différences de comportement des deux groupes sont attribuées à des représentations différentes des capacités des interlocuteurs. » Une expérience du même genre menée aujourd'hui donnerait sans doute des comportements moins marqués, car l'ordinateur est maintenant beaucoup plus présent dans la société et la représentation qui en est faite évolue sans cesse.

L'idéal serait de prendre en compte tous ces aspects liés aux représentations mentales des interactants. Certains modèles informatiques et certaines théories philosophiques les intègrent désormais plus ou moins en les adaptant au dialogue homme-machine [Nicolle & Saint-Dizier De Almeida, 1999a] [Coursil, 1993] [Vernant, 1997b]. Ces modèles sont cependant encore très spéculatifs et difficiles à mettre en œuvre. C'est la raison pour laquelle nous nous sommes focalisé sur un seul aspect à la fois, aspect qui nous paraît central. Nous nous sommes intéressé à *l'opérationnalisation* de la notion de terrain commun. Ainsi, pour répondre à notre questionnement de départ portant sur les conditions d'établissement d'un dialogue entre un agent humain et un agent artificiel, nous avons construit notre raisonnement sur l'hypothèse suivante : un dialogue homme-machine authentique ne peut s'établir qu'à la condition[3] qu'un terrain commun puisse se construire entre l'utilisateur et le système. En posant cette hypothèse, nous supposons également que le fonctionnement cognitif d'un utilisateur en situation de dialogue avec un système informatique est analogue à celui d'un homme dialoguant avec un autre homme. C'est généralement le postulat posé mais non formulé pour la plupart des modèles du dialogue homme-machine. Il faudra bien sûr dire ce que signifie la notion de terrain commun dans ce cadre.

Sabah affirme que « pour garantir l'ergonomie des interprétations construites par la machine, c'est-à-dire leurs conformités aux attentes des utilisateurs, le fonctionnement du système mis en œuvre doit être analogue à celui de la cognition humaine. » [Sabah *et al.*, 1997, p. 24]. Sans aller aussi loin que cet auteur, nous ne garantirons pas la plausibilité cognitive du fonctionnement du système. Nous chercherons en priorité à formuler un modèle fondé sur notre hypothèse de recherche.

1.2 Construction d'un terrain commun

Par définition, le terrain commun aux participants est construit au cours du dialogue. L'image que s'en fait chaque interactant intervient dans l'interprétation des énoncés et s'en

[3] Il s'agit d'une condition *nécessaire* et *non suffisante*. Nous avons suggéré que bien d'autres éléments interviennent ; éléments que nous maîtrisons encore moins dans le cadre du DHM. Notamment, il faudrait étudier sérieusement l'intérêt et/ou la mise en œuvre de tout le jeu subtil des relations sociales entre partenaires (cf. travaux de Kerbrat-Orecchioni [Kerbrat-Orecchioni & Plantin, 1995]).

trouve modifiée. Comment garantir la construction d'un terrain commun lors d'un dialogue homme-machine ? Chez les humains, le terrain commun se construit au fur et à mesure que les participants ajustent leur compréhension réciproque en voyant les conséquences de leurs actions. Il s'agit d'un mécanisme itératif : certains éléments préexistent avant même que le dialogue commence. Ces éléments « par défaut » sont ensuite complétés, modifiés, enlevés, etc. La construction du terrain commun n'est donc jamais garantie puisqu'elle est associée à la compréhension des deux participants. De plus, elle est subjective, puisque chaque interactant en a une représentation propre. La question serait donc plutôt : comment faciliter la construction d'un terrain commun lors d'un dialogue homme-machine ? Deux éléments de réponses peuvent nous être apportés par l'étude des conversations humaines : la théorie de l'interprétation des énoncés de Grice et la théorie des actes de langage.

Grice a proposé un principe conversationnel général [Grice, 1975]. Selon l'auteur, toute conversation serait intrinsèquement coopérative[4] : « [speaker must try to] *make his contribution such as is required, at the stage at which it occurs, by the accepted purpose or direction of the talk exchange in which he is engaged.* »[5] Cette coopération est décrite sous la forme de maximes que le locuteur est supposé respecter ou non. Ce respect ou non-respect permettrait à l'auditeur d'interpréter correctement l'énoncé du locuteur et de lui attribuer une *intention*. Ces maximes seraient nécessaires afin d'expliquer la divergence souvent constatée entre le sens littéral de la phrase linguistique énoncé, nommé généralement *signification* par les sémanticiens [Beust, 1998], et le sens de l'énoncé en contexte, qui peut être d'ailleurs porté par d'autres éléments extralinguistiques. Grice nomme *implicature* le mécanisme inférentiel par lequel l'auditeur parvient à interpréter correctement l'énoncé, c'est-à-dire dans le sens où l'entendait le locuteur.

Sans remettre en cause les faits observés, il nous semble qu'il s'agit essentiellement, en ce qui concerne les maximes, d'une *description* des caractéristiques que l'on trouve dans les conversations humaines. Pour l'informaticien, cette explication des faits de surfaces est un peu courte même si elle est juste. Chercher à appliquer le principe de coopération n'est pas possible directement puisque rien n'est dit sur les mécanismes cognitifs par lesquels le locuteur choisirait ou non de respecter telle ou telle maxime et de transgresser telle autre afin d'être compris.

Prenons un exemple de tentative d'adaptation des maximes de Grice. Dans l'environnement Synergic [Gleizes *et al.*, 1994], une plate-forme de développement de sociétés d'agents autonomes, une application des maximes de Grice est proposée afin d'améliorer « l'économie

[4] La notion de coopération de Grice ne recouvre pas le même sens que la notion de coopération entre agents du paradigme de l'IAD (Intelligence Artificielle Distribuée). Dans un SMA (Système Multi-Agents) les agents artificiels peuvent interagir de différentes façons : ils peuvent être en conflit, en compétition, coopérer ou encore collaborer dans leurs comportements [Erceau & Ferber, 1994]. Chez Grice, il s'agit d'une coopération conversationnelle et non comportementale.

[5] Sperber et Wilson propose la traduction du principe de coopération suivante : « *que votre contribution à la conversation soit, au moment où elle intervient, telle que le requiert l'objectif ou la direction acceptée de l'échange verbal dans lequel vous êtes engagé.* ».

du dialogue tout en conservant un haut degré de pertinence » des messages inter-agents. Cependant, l'interprétation qui en est faite n'est pas assez satisfaisante pour gérer un dialogue homme-machine, qui est d'une complexité sans commune mesure avec les « conversations » d'agents logiciels que l'on maîtrise à souhait. Par exemple, pour respecter les maximes de manière, les auteurs affirment que « l'obscurité d'expression et l'ambiguïté sont levées par des règles de réécriture. » Pour nous, un énoncé n'est jamais vraiment ambigu. Simplement, il peut donner lieu à plusieurs interprétations justement négociables par le dialogue. Nous voyons que les auteurs ont une vision statique du dialogue. Au contraire, nous savons depuis les études de Luzzati que le dialogue est « un processus erratique où l'erreur est non seulement admise, mais encore constitutive du processus interactif. » [Luzzati, 1995a] [Luzzati & Taleb, 1995].

Dans la théorie de Grice deux points nous semblent cependant particulièrement intéressants. Premièrement, d'un point de vue épistémologique, Grice a été l'un des tous premiers pragmaticiens à voir dans les conversations autre chose qu'une manifestation de la nature vériconditionnelle de la langue. Même s'il n'a pas remis en cause cette conception de la langue, ce qu'à fait Austin lorsqu'il a débuté ses travaux sur les actes de langage, Grice a changé la manière de voir la pragmatique. Il a en effet limité le travail du sémanticien aux aspects vériconditionnels des énoncés en laissant « le reste » aux pragmaticiens.

Le second point intéressant, qui est central dans le principe de coopération, est l'idée que les conversations humaines sont conformes à une certaine logique, que l'auteur nomme *signification non naturelle* : « dire qu'un locuteur L a voulu signifier quelque chose par X, c'est dire que L a eu *l'intention*, en énonçant X, de produire un effet sur l'auditeur A grâce à la reconnaissance par A de cette intention » [Moeschler & Reboul, 1994]. Avec Grice, le modèle de la communication héritée de la théorie de Shanon et Weaver (appelé *modèle du code*) a fait place au modèle de l'inférence, c'est-à-dire à l'idée que le message transmis n'est pas simplement encodé puis décodé, mais qu'il donne lieu à un raisonnement pour être compris.

La notion d'intentionnalité est devenue par la suite centrale dans la pragmatique contemporaine. Nous pensons qu'elle intervient fortement dans la notion de terrain commun. En effet, nous postulons que les énoncés sont interprétés en fonction de la représentation d'un terrain commun et d'intentions attribuées aux locuteurs, et qu'ils proviennent également d'intentions émanant de la représentation du terrain commun.

Les travaux de Grice ont beaucoup influencé les philosophes du langage tel que Searle dont nous reparlerons, mais également les cogniticiens du langage. Sperber et Wilson ont proposé une théorie à la fois plus ambitieuse et moins descriptive que celle de Grice, puisqu'elle met en scène un modèle complet de la pragmatique [Sperber & Wilson, 1989]. L'impact de cette théorie est aussi important que celle de Grice. Elle est présentée comme une généralisation des maximes de coopération. Sévèrement critiquée par certains ([Caron, 1997] [Bange, 1992] [Dessalles, 1993]), elle a cependant l'intérêt d'avoir mis en lumière l'importance de la notion de pertinence dans les conversations humaines. De plus, le principe de pertinence s'appuie sur

l'hypothèse de l'existence d'un *environnement cognitif partagé* qui se trouve être une notion à la fois très proche mais pourtant différente de celle de terrain commun. La différence entre les deux notions vient du mode de construction : le terrain commun résulte d'une co-construction interactionnelle alors que l'environnement cognitif partagé de Sperber et Wilson résulte d'un processus cognitif uniquement personnel où l'interaction, langagière ou non, ne semble pas intervenir.

Nous pensons que la notion de pertinence doit intervenir dans la modélisation des intentions de communication, et donc indirectement dans la construction du terrain commun, puisque celui-ci intervient dans les processus de la compréhension et les processus intentionnels. Aussi, pour revenir à nos préoccupations concernant le dialogue homme-machine, nous pensons que pour faciliter la construction d'un terrain commun, l'informaticien doit s'assurer de la pertinence du comportement conversationnel du système. La notion de pertinence doit être cependant redéfinie dans le cadre des dialogues entre agents artificiels et agents naturels. Nous esquisserons quelques traits d'un modèle de la pertinence pour une machine.

Outre les travaux de la pragmatique gricéenne, la théorie des actes de langage a marqué fortement les chercheurs du domaine. Elle est désormais relativement bien établie : que ce soient les philosophes du langage ([Vernant, 1997b] [Vanderveken, 1990]), les psychologues de l'interaction ([Brassac & Pesty, 1999] [Trognon, 1995]), les pragmaticiens [Moeschler & Reboul, 1994] ou les informaticiens ([Sabah *et al.*, 1997] [Nicolle & Saint-Dizier De Almeida, 1999a]), tous s'accordent à faire de cette théorie un élément central incontournable de la modélisation des dialogues naturels et artificiels. Par exemple, chez les anglo-saxons principalement, la mise en œuvre de cette théorie a été le moteur d'une branche importante de recherches autour de la notion de plan [Cohen *et al.*, 1992]. La théorie des actes de langage a en effet pour avantage de s'inscrire dans un cadre théorique général qui est celui de la théorie de l'action, telle que la définisse par exemple les ethnométhodologues [Bange, 1992] ou le philosophe du langage Searle [Searle, 1972].

La théorie des actes de langage est en effet particulièrement intéressante pour la problématique générale du DHM. Au même titre que la théorie de Grice, il s'agit d'une théorie de la signification : le but est d'expliquer la manière dont les gens s'expriment pour agir sur leur environnement, mais plus encore, il s'agit d'une théorie de l'intentionnalité car chaque acte de langage est avant tout une *action intentionnelle* [Vanderveken, 1999]. La version contemporaine de la théorie a beaucoup évoluée depuis *How to Do Things with Words* de Austin [Austin, 1962]. Une conversation est le moyen de modifier les *états mentaux* des participants à travers l'accomplissement d'actes de langage. La logique illocutoire de Searle et Vanderveken décrit les conditions de succès et de satisfaction des actes [Searle & Vanderveken, 1985].

Cette dernière version de la théorie souffre cependant d'un important défaut qu'elle partage avec la théorie de la pertinence de Sperber & Wilson. Ce défaut porte sur la manière dont sont expliqués les enchaînements langagiers. Nous considérons que ces théories donnent une image

quelque peu statique et monologique des conversations. Il semble que ces auteurs considèrent en effet que le sens des énoncés est interprétable à tout coup de façon irrévocable en fonction de certains principes (pertinence et logique illocutoire), et qu'il ne peut pas être remis en cause après interprétation puisque ces principes sont sensés garantir *in fine* une interprétation unique (sauf, bien sûr, en cas d'ambiguïté linguistique). Cela se reflète dans les exemples que prennent par exemple Sperber et Wilson : ce ne sont pas des dialogues ou même des extraits de dialogue réels, mais des énoncés artificiels. Dessalles se permet d'ailleurs de qualifier la théorie de la pertinence de « *pragmatique de fauteuil* » [Dessalles, 1993, p. 61].

Il en va tout autrement d'autres modèles plus cognitifs, qui mettent tous en avant le caractère *construit* du sens en interaction. Nous faisons référence en l'occurrence au modèle de Trognon et Brassac [Trognon & Brassac, 1992], aux travaux de Vernant (op. cité), au modèle de Nicolle et Saint-Dizier De Almeida (op. cité), et également au modèle en ethnométhodologie linguistique de Bange (op. cité). Dans ces modèles, le sens des énoncés n'est pas donné d'emblée par une interprétation exhaustive. Il n'existe pas en dehors de la conversation, car il est le fruit d'une *co-construction en actes* au cours de cette conversation. Selon Nicolle, « la singularité [de ces modèles] est de placer l'allocutaire — et non pas le locuteur — au centre de la dynamique interactionnelle. » Bange explicite le mécanisme de l'enchaînement conversationnel de la manière suivante (A et B sont les deux participants et B interprète l'énoncé de A) :

> « Cette interprétation de B, qui reste le plus souvent implicite dans la réponse qu'il donne à A, est en réalité une **offre d'interprétation** adressée au locuteur A qui l'examine pour voir si elle est une interprétation acceptable pour lui et son énonciation initiale. Si A n'a pas de raison massive de refuser, il accepte l'offre d'interprétation de B et cette acceptation est le plus souvent implicite dans la réaction de A à la réponse de B. Cette réaction de A à la réponse de B peut donc être comprise comme comportant une évaluation par A de l'interprétation faite par B sur son énonciation initiale à lui A. [...] C'est dans ce jeu de réciprocité que le sens se constitue. »

Pour expliquer la compréhension des énoncés, le modèle de l'enchaînement conversationnel s'appuie sur *l'alternance des locuteurs* et non sur un seul tour de parole comme dans les exemples analysés par Sperber et Wilson, où l'alternance n'a pas lieu. Ce second type de modèles fait référence à *une logique interlocutoire* par opposition ou en plus de la logique illocutoire de Vanderveken.

C'est ce deuxième point de vue sur les conversations humaines que nous prendrons comme cadre théorique dans notre travail d'opérationnalisation de la notion de terrain commun.

1.3 Résumé

À partir d'un questionnement sur les conditions d'établissement d'un véritable dialogue homme-machine, nous avons fait l'hypothèse que celui-ci était dépendant de la construction d'un terrain commun. En évoquant les principaux modèles et théories des interactions langa-

gières, nous avons ensuite recensé un certain nombre d'éléments intéressants pour répondre à notre interrogation initiale. Nous reprenons ici ces éléments auxquels nous apportons quelques précisions.

- La notion de terrain commun est liée à l'intentionnalité des interactions. En effet, les actions intentionnelles (actes de langage) du locuteur sont générées à partir de la représentation du terrain commun dans le but d'agir sur l'état du monde. Le terrain commun est donc un moyen de cette action est non un but.
- Sans ce but, qui est celui d'agir sur le monde pour satisfaire ses besoins, il n'y aurait jamais d'interaction. Chez l'homme, cela peut provenir d'un besoin quelconque occasionné par ses activités quotidiennes. Par exemple, il a besoin d'interagir avec ses collègues de travail pour effectuer son propre travail. Pour quelles raisons un système informatique devrait-il interagir avec ses utilisateurs ? Il faudra définir un moyen de motiver le système pour répondre aux sollicitations des utilisateurs. Cela pourra prendre la forme de règles ou de principes comportementaux explicitement représentés ou non.
- Une fois le système doté d'une capacité d'interaction, la cohérence des échanges doit être assurée pour qu'un terrain commun puisse être construit. Celle-ci passe nécessairement par un principe garantissant la pertinence des actes de langage générés à partir des intentions communicatives.
- Cependant, garantir la cohérence des échanges ne suffit pas à assurer l'intercompréhension. Une logique interlocutoire est nécessaire afin d'expliquer comment est modifié le terrain commun au cours des interactions. Pour mettre en œuvre une logique interlocutoire nous avons repris à notre compte puis modifié le modèle du DHM développé par Lehuen [Lehuen, 1997b].

Notre travail de thèse propose un modèle informatique de génération des intentions communicatives qui intègre les différentes idées que nous venons de résumer en quatre points.

Chapitre 2

L'enchaînement conversationnel

Ce chapitre présente la modélisation de l'enchaînement conversationnel, qui est le lieu d'échange des intentions communicatives des locuteurs. Il s'agit ici de prendre en compte le caractère construit *du sens en interaction. À cet égard, le modèle des actes de langage de Searle et Vanderveken doit être revu : comme le notent Brassac et Pesty, il faut aboutir à « une dialogisation de l'échange entre agents. » [Brassac & Pesty, 1999, p. 320].*

Pour découvrir les intentions de l'utilisateur d'un système, deux approches sont possibles : soit on considère qu'elles sont planifiées à l'avance, soit on considère qu'elles ne sont pas prévues à l'avance mais s'organisent, s'ajustent au cours de l'interaction. Après une critique des modèles de dialogue fondés sur la planification, nous présentons le modèle psychologique de Trognon & Brassac, avant d'aborder des modèles informatiques tel que celui de Nicolle. Nous finissons par le modèle COALA de Lehuen [Lehuen, 1997b] *que nous avons dans un premier temps repris, afin de l'expérimenter et d'obtenir un premier corpus de dialogue homme-machine. Aux regards des résultats, nous avons ensuite développé notre propre modèle que nous présentons au chapitre suivant.*

2.1 Planification *versus* opportunisme

Les modèles de dialogue proposés en pragmatique informatique sont principalement issus de l'intelligence artificielle. Une grande partie de ces modèles provient des recherches américaines. Ils sont fondés sur la notion de *plan* et s'appuient sur la théorie des actes de langage d'Austin, reprise par Searle, pour représenter le dialogue comme une suite d'actions communicatives visant à modifier l'état mental des agents[1] participants.

[1] La terminologie employée pour désigner les participants à un dialogue que nous prenons dans cette section, correspond à celle utilisée par les auteurs travaillant en planification. Un *agent* désigne aussi bien un participant humain qu'un participant artificiel. Ceci provient du fait que les recherches du domaine sont maintenant tournée vers la réalisation de système multi-agents collaboratifs.

À l'origine de cet axe majeur de la recherche anglo-saxonne se trouvent les travaux d'Allen & Perrault[2], les travaux de Cohen & Levesque[3] et de bien d'autres (voir par exemple [Cohen *et al.*, 1992] pour un panorama de ces modèles). Des chercheurs français ont également repris ce modèle. Citons les travaux de Guyomard, Nerzic et Siroux à l'IRISA [Guyomard *et al.*, 1995], et les travaux de Vilnat et de Balkanski au LIMSI [Vilnat, 1997] [Balkanski & Hurault-Plantet, 1997].

Nous rappelons le principe commun à ces modèles, puis nous discutons de l'adéquation de la notion de plan dans le dialogue en général et dans le dialogue homme-machine en particulier.

2.1.1 Principe des modèles fondés sur la planification

Les modèles fondés sur la planification sont à notre sens moins des modèles de dialogue que des modèles de raisonnement des agents, pour qui communiquer est un moyen parmi d'autres d'atteindre certains buts visés. Le principe commun à tous ces modèles s'appuie à l'origine sur la *rationalité* des agents, c'est-à-dire sur l'hypothèse que les agents établissent des *plans* pour atteindre leurs buts. Parallèlement et conjointement à ce principe commun, certains auteurs ont proposé des extensions mettant en jeu les notions de *méta-planification* et de *plan partagé*. Nous allons présenter succinctement ces trois notions.

La notion de plan

Un plan permet l'anticipation d'une succession d'actions pour atteindre un but, c'est-à-dire un certain état final visé du monde. L'exemple généralement pris est celui du voyage. Si vous souhaitez ou devez faire un voyage (votre but), vous allez établir une succession d'actions (votre plan) dont l'enchaînement séquentiel va vous permettre d'accomplir effectivement ce voyage. Pour cela vous devez successivement : acheter un billet de transport pour la destination choisie, faire vos bagages, vous rendre sur le lieu du départ, effectuer les formalités du départ, prendre le moyen de transport, etc. Dans la théorie des actes de langage les énoncés sont vus comme des actions à part entière. Aussi, les premiers auteurs ont inscrit les actions communicatives dans les plans formulés par les agents au même titre que les actions physiques.

La *planification* consiste à créer, à partir d'une « bibliothèque » de classe d'actions, les instances d'actions spécifiques à la réalisation du but poursuivi. La *reconnaissance de plan* est l'activité inverse. Elle consiste à déduire le but du locuteur à partir d'une ou plusieurs de ses actions. En reprenant l'exemple du voyage, si vous savez que votre voisin a acheté un billet de transport et que vous le voyez partir avec une valise à la main, vous pouvez déduire qu'il part en voyage, et alors, peut-être, lui proposer de l'emmener à la gare. Vous venez de reconnaître le plan que votre voisin a établi et vous avez même essayé de contribuer à la réalisation de

[2] J. Allen et R. Perrault. Analyzing intention in utterances. *Artificial Intelligence,* 15(3):143-178, 1980.

[3] P. Cohen et H. Levesque. *Intentions in C*ommunication, chap. 12 Rational Interaction as the Basis for Communication. Bradford Books, MIT Press, seconde édition, 1992.

son but. La reconnaissance du plan permet à l'interlocuteur de déterminer les intentions et le but du locuteur, de prédire ou d'anticiper les actions possibles à venir, et de détecter les obstacles pouvant entraver la réalisation de ce but. Remarquons tout de suite que dans cet exemple aucun dialogue n'est nécessaire pour que reconnaître le but de son voisin. Il suffit de savoir comment se déroule un voyage et d'avoir vu ses différents gestes.

Ces deux activités que chaque participant à une conversation est censé avoir, sont modélisées en s'appuyant sur le fonctionnement cognitif des participants. Pour établir ou reconnaître un plan, il faut tout d'abord avoir des connaissances sur *l'état du monde* pour le modifier et atteindre l'état final correspondant au but fixé. Il faut aussi avoir des connaissances sur *les moyens d'atteindre ce but.* Les participants ont également des *croyances* sur le monde et sur les connaissances et croyances des autres participants. Ils ont enfin des intentions de faire une action et des intentions d'être dans une certaine situation. À chaque instant, les connaissances, les croyances et les intentions d'un agent forment son « état mental ».

La méta-planification

Des limites innérantes aux modèles initiaux entraînent une ***rigidité dialogique*** des systèmes mettant en œuvre de tels modèles : incapacité à corriger une « erreur » (dans le sens défini par Nerzic [Nerzic, 1993]), incapacité à modifier un plan en cours de reconnaissance, et incapacité à prendre en compte des buts multiples (voir la thèse de Pernel pour un exemple de modélisation [Pernel, 1994]). Une solution est d'ajouter un niveau d'abstraction supplémentaire en établissant des *méta-plans.* Ces derniers ont pour arguments d'autres plans. Ainsi, en dialogue homme-machine finalisé, Litman considère deux catégories de plans : les plans du domaine référant à l'application considérée, et les plans du discours (méta-plans) référant aux dialogues eux-mêmes [Litman & Allen, 1992]. Les plans du discours sont indépendants de l'application. Ils modélisent les relations qu'entretiennent les énoncés et les plans du domaine. Autrement dit, dans un dialogue et même dans un sous-dialogue de clarification (dialogue métadiscursif), les actions communicatives sont censées être planifiées par les locuteurs.

Les plans partagés

La notion de plan partagé a été développée dans le but de permettre la modélisation de situations où la **collaboration** entre agents pour la réalisation d'une tâche commune est essentielle. Elle permet de représenter l'ensemble des croyances et intentions que ces agents doivent avoir pour que leur collaboration puisse réaliser leur but commun. D'après les auteurs, le dialogue finalisé est justement l'une de ces situations. Grosz et Kraus en ont proposé une formalisation complète à partir de l'affirmation suivante :

> « *Because collaborative action comprises actions by different agents, collaborative planning and activity involve the intentions of multiple agents. As a result, collaborative plans cannot be recast simply in terms of the plans of individual agents, but require an integrated*

> *treatment of the beliefs and intentions of the differents agents involved. Furthermore, the collaborative planning process is a refinement process; a partial plan description is modified over the course of planning by the multiple agents involved in the collaboration.* »[4] [Grosz & Kraus, 1996].

Une recette est un ensemble d'actions devant être accomplies pour que l'action principale puisse être accomplie à son tour. C'est le corps de l'action. Ainsi, pour accomplir une action commune un groupe d'agents doit avoir :

1. la connaissance mutuelle d'une recette (partielle au début) ;

2a. des intentions personnelles que l'action soit faite ;

2b. des intentions personnelles que les autres agents réussissent à faire les sous-actions prévues dans le corps de l'action ;

3. et des plans personnels ou partagés pour faire les sous-actions éventuelles.

Il s'agit d'une véritable révision de la notion de plan puisque, outre les plans individuels complets, il faut définir quatre autres métaplans : les plans individuels partiels, les plans partagés complets, les plans partagés partiels et les plans partagés de statut indéfini. À titre d'illustration (et d'exercice) le lecteur pourra observer la définition logique du métaplan individuel complet (noté FIP en anglais) donné en Fig. 2.1.

2.1.2 Adéquation de la notion de plan en dialogue

La notion de plan s'appuie sur l'existence d'une bibliothèque d'actions disponibles aux interlocuteurs. Cela semble confirmé par les études en psychologie cognitive avec la notion de *schéma d'action* très proche de la notion d'action en planification [Richard, 1990]. Des études portent également sur la résolution de problèmes dans laquelle l'activité du sujet est interprétée en termes de planification [Hoc, 1987].

En psychologie cognitive une action peut être vue sous deux aspects : sous son aspect déclaratif, elle exprime un changement d'états car elle exprime l'état résultant ; sous sa composante procédurale elle définit un déroulement et un mode de réalisation. Les actions possèdent des pré-requis. Ce sont des conditions qui doivent être réalisées pour que l'action puisse être exécutée. Ces pré-requis permettent aussi de faire des inférences sur des actions ou sur des états antérieurs. Les informations sur une action sont hiérarchisées, ce qui définit un ordre d'accès. L'information la plus accessible est celle qui concerne le résultat de l'action. Les prérequis sont accessibles en dernier. Un exemple de schéma de l'action « déplacer » est donné à la Fig. 2.2.

[4] « Du fait qu'une action collaborative comprend des actions de différents agents, l'activité de planification collaborative implique de considérer les intentions de plusieurs agents. En conséquence, les plans collaboratifs ne peuvent pas être simplement exprimés dans les termes de plans d'agents individuels, mais nécessitent un traitement intégrant les croyances et les intentions des différents agents concernés. En outre, le processus de planification collaboratif est un processus par raffinement ; un plan partiel est modifié durant le cours de la planification par les différents agents collaborant. »

$$FIP(P, G, \alpha, T_p, T_\alpha, R_\alpha, C_\alpha)$$

(1) Un agent G a un plan P au temps T_p pour faire une action α au temps T_α utilisant la recette $R_\alpha$dans le contexte C_α, c'est-à-dire il « connaît » les sous-actions β_i nécessaires et leurs contraintes ρ_j :
$R_\alpha = \{\beta_i, \rho_j\} \wedge Bel(G, R_\alpha \in Recipes(\alpha), T_p) \wedge$
Pour chaque action β_i, soit l'agent G a l'intention de la faire lui-même (2), soit il croit qu'il peut obtenir de quelqu'un d'autre de la faire (3) :
$(\forall \beta_i \exists T_{\beta_i})[$

(2) L'agent G a l'intention de faire l'action lui-même.

(2a) L'agent G a l'intention de faire la sous-action β_i :
$[Int.To(G, \beta_i, T_p, T_{\beta_i}, C_{\beta_i/\alpha}) \wedge$

(2b) Si la sous-action β_i n'est pas une action élémentaire, G a un plan individuel complet pour faire β_i en utilisant la recette R_{β_i} :
$\neg basic.level(\beta_i) \Rightarrow (\exists P_{\beta_i}, R_{\beta_i})\, FIP(P_{\beta_i}, G, \beta_i, T_p, T_{\beta_i}, R_{\beta_i}, C_{\beta_i/\alpha})] \quad \otimes$

(3) G a l'intention qu'un autre agent G_c fasse la sous-action β_i. Pour cela il doit faire une autre action (γ) en échange :
$[\exists (G_c, \gamma, T_\gamma)$

(3a) G croit en faisant l'action γ qu'il peut obtenir de G_c qu'il fasse la sous-action β_i :
$[Bel(G, GTD(G, \gamma, T_\gamma, G_c, \beta_i, T_{\beta_i}, constr(C_\alpha) \bigcup \{\rho_j\}), T_p) \wedge$

(3b) G a l'intention de faire l'action γ :
$Int.To(G, \gamma, T_p, T_\gamma, C_{\gamma/\beta_i/\alpha}) \wedge$

(3c) G à l'intention que l'agent G_c accomplisse l'action β_i :
$Int.Th(G, (\exists R_{\beta_i}) CBA(G_c, \beta_i, R_{\beta_i}, T_{\beta_i}, constr(C_\alpha) \bigcup \{\rho_j\}), T_p, T_{\beta_i}, C_{cba/\beta_i/\alpha}) \wedge$

(3d) Si l'action γ n'est pas une action élémentaire, G a un plan individuel complet en utilisant la recette R_γ :
$\neg basic.level(\gamma) \Rightarrow (\exists R_\gamma, P_\gamma) FIP(P_\gamma, G, \gamma, T_p, T_\gamma, R_\gamma, C_{\gamma/\beta_i/\alpha})]]]$

Notations :
- Bel : opérateur modal de croyance.
- $Recipes$: fonction donnant les recettes pour faire une action.
- $Int.To$: opérateur modal d'intention (l'intention de faire quelque chose).
- $Int.Th$: opérateur modal d'intention (l'intention d'être dans une certaine situation).
- GTD : méta-prédicat pour désigner qu'un agent peut obtenir d'un autre agent qu'il accomplisse une action (*Get To Do*).
- CBA : méta-prédicat pour désigner qu'un agent peut accomplir une action (*Can Bring About*).
- $constr$: fonction donnant les contraintes induites par un contexte.
- $basic.level$: fonction testant si une action est élémentaire ou non.

FIG. 2.1 – Définition du métaplan individuel complet tiré de [Grosz & Kraus, 1996].

DEPLACER(W, X, Y, Z) c'est
Contraintes sur les arguments
W est l'agent (animé) ou la cause (inanimée) du déplacement
X est un solide
Y est le lieu où se trouve X
Z est un lieu différent de Y
Résultat
X est en Z
X n'est plus en Y
Mode de réalisation :Déplacement à la main
Premier sous-but : Prendre X en Y
Prérequis : 1 - W est à côté de X
2 - Il n'y a rien au-dessus de X
Second sous-but : Poser X en Z
Prérequis : 1 - W est à côté de Z
2 - Il y a de la place en Z où poser X

FIG. 2.2 – Le schéma de l'action « déplacer » tiré de [Richard, 1990].

L'analogie entre schéma d'action en psychologie cognitive et action en planification du dialogue s'arrête cependant là, car elle n'est plus vraie pour les actions langagières[5]. Tout de même, il est bien clair que la notion de plan ne peut être que difficilement remise en cause. Ce que nous critiquons c'est son utilisation inadéquate en matière de modélisation des phénomènes dialogiques.

Établir un plan puis effectuer les actions qui le composent est nécessaire pour atteindre un but, et c'est la meilleure méthode pour y parvenir si le monde est parfaitement connu[6]. Ceci semble valable pour les humains mais encore plus pour les machines. Ces dernières ont besoin d'être *programmées*, c'est-à-dire que leur soit « injecté » un plan pré-établi qu'elles suivront rigoureusement. Pour qu'elles simulent une intelligence « l'astuce » consiste à établir un métaplan minimal capable de construire des plans. Ceci vaut pour des actions non-langagières, mais pour le dialogue, où l'intention communicative devrait être l'objet d'une co-construction, l'astuce ne tient plus, parce qu'il ne s'agit plus de planifier mais d'agir au *coup par coup*. Pour expliquer l'enchaînement conversationnel nous préférons remplacer la planification des actions langagières par ***l'opportunisme*** des actions langagières que l'on trouve dans les modèles que nous décrirons aux sections suivantes.

La distinction que fait Litman entre tâche et discours nous semble au départ intéressante :

> « *However, in task-oriented dialogues an important distinction must also be made between discourse intentions and another nonlinguistic notion relevant to discourse theory: commonsense task knowledge. This distinction is needed to account for subdialogues that do not directly correspond to commonsense tasks, for example, clarification and correction subdialogues.* »[7] [Litman & Allen, 1992].

[5] À notre connaissance très peu d'études ont été menées en psychologie cognitive sur les interactions verbales. En revanche des travaux existent en compréhension de texte [Pynte, 1988].

[6] Une autre méthode pour atteindre un but peut consister à procéder par « essai/erreur » en effectuant un essai, mesurer la distance séparant l'état courant du monde et l'état visé, puis recommencer en fonction de cette différence jusqu'à ce qu'elle soit nulle.

[7] « Cependant, dans les dialogues orientés par la tâche, une importante distinction doit être faite entre les

Cependant, pourquoi y introduire aussitôt des plans du discours ? Dans un tel modèle les intentions (à l'origine des actions communicatives) sont *figées* car elles sont planifiées, prévues à l'avance. Un système fondée sur un tel modèle manque alors *d'autonomie langagière*. Ce qu'il dit et ce qu'il comprend ne peuvent pas être remis en cause. Aussi, de notre point de vue, de tels modèles ne sont pas vraiment des modèles de dialogue comme nous l'entendons, mais des modèles de raisonnement dont l'adéquation reste à prouver en matière de dialogue (cf. Sec. 2.2).

Par ailleurs, avec la notion de plans partagés, les chercheurs s'éloignent des dialogues finalisés pour aller vers la problématique des univers multi-agents [Grosz, 1996]. Le modèle des plans partagés nécessite des outils de formalisation puissants (logique du premier ordre augmentée d'opérateurs modaux et de méta-prédicats, cf. Fig. 2.1). On se trouve encore ici à l'opposé des possibilités de construction du sens offertes par la logique interlocutoire que nous cherchons à prendre en compte. Dans ces modèles, même s'il y a « partage » entre les participants, les actions communicatives (actes de langage) ne sont aucunement objets de négociation.

Nous nous sommes efforcé de montrer l'inadéquation des modèles fondés sur la planification au type d'usage des interactions langagières que nous visons. Nous allons voir dans les prochaines sections plusieurs modèles théoriques puis opératoires qui répondent davantage à nos exigences.

2.2 Logique interlocutoire

2.2.1 La rationalité des conversations

Nous avons vu précédemment que les modèles de la planification sont fondés sur l'hypothèse que les agents sont rationnels, et que cette rationalité apparaît dans la manière dont ils planifient leurs actions. Cela nous semble vrai pour les actions physiques, mais discutable pour les actions langagières. Nous avons dit qu'elles ne devaient pas être planifiées. Le mode de raisonnement pour les actions physiques ne peut pas être le même que celui des actions langagières. Pour Trognon, « la rationalité telle qu'elle s'exerce dans les conversations est une rationalité pragmatique. » :

> « En somme dans le raisonnement mis en œuvre conversationnellement, les interactants raisonnent avec des illocutions (et non avec des propositions) et selon la rationalité communicationnelle qui gouverne leur usage. » [Trognon, 1997b, p. 258].

Reprenons l'analyse de Trognon d'un exemple de conversation portant sur une *situation problématique* (exemple traité également dans [Trognon, 1996]). Il s'agit d'un dialogue dont le

intentions liées au discours et une autre notion non-linguistique ne relevant pas de la théorie du discours : les connaissances de sens commun sur la tâche. Cette distinction est nécessaire pour prendre en compte les sous-dialogues qui ne correspondent pas directement à ceux des tâches de sens commun, tels que, par exemple, les sous-dialogues de clarification et de correction. »

propos est guidé par l'établissement d'une solution à un problème professionnel. Une telle situation peut être comparée au dialogue finalisé en informatique. Deux ouvriers tiennent une conversation lors d'une relève de poste. L'ouvrier descendant A quitte le poste et l'ouvrier montant B le prend. Ces deux ouvriers travaillent sur une machine à faire des feuilles de papier, dont le bon fonctionnement nécessite des réglages fins qu'ils doivent faire par intermittence. Le dialogue en question est retranscrit à la Fig. 2.3.

...
A1 : (...) et les pissettes, ça a l'air d'aller mieux.
B1 : et celle de derrière, elle soulève toujours un peu la feuille,
si t'as remarqué.
A2 : ben oui, peut-être. Mais, j'ai pas eu de pâte par rapport à hier,
j'ai pas eu de pâte après hein.
B2 : j'en ai eu
A3 : t'en as eu ? moi j'en ai pas eu hein
B3 : et j'avais rediminué un peu, parce que je trouvais qu'elle écartait
un peu la feuille et ça faisait euh... gicler.
A4 : ah oui, moi ce que j'ai, j'ai rouvert, c'est celle de devant ce matin,
un tout petit poil, parce que bon, t'as vu aujourd'hui j'ai cassé
(le descendant montre le cahier, matin, partie droite) , bon, j'ai tiré
la pointe euh... trois fois...
B4 : oui
A5 : mais la bande, elle a pas été coupée. J'ai pas nettoyé les pissettes
hein. J'ai même pas enlevé la pâte à la main, rien et y en a pas
après, y'a juste un peu de fibres. c'est tout hein.
B5 : parce qu'elles sont bien réglées.
A6 : c'est aussi bien, hein ?
...

FIG. 2.3 – Rationalité pas à pas dans un dialogue naturel tiré de [Trognon, 1997b].

Au début du dialogue A et B sont en désaccord. Pour B la bordeuse située à l'arrière soulève la feuille tandis que A pense que « ça à l'air d'aller mieux ». À l'issu de l'intervalle {B1,A3}, le contenu du désaccord est bien délimité. Les deux interlocuteurs se sont accordés sur la proposition que la bordeuse arrière soulève la feuille. Mais ils sont en désaccord sur le rôle de cette bordeuse quant aux projections de pâte à papier. Pour B le fait que la bordeuse arrière soulève la feuille est la cause des projections de pâte. Pour A ce fait même n'est pas une condition suffisante puisque dans les mêmes conditions il n'a pas eu de projection. Cette contradiction va être dénouée durant l'intervalle {B3,A5} au cours duquel les interlocuteurs vont s'informer de leurs expérimentations respectives. B a réglé la bordeuse arrière sans résultat apparent. En revanche, A a réglé la bordeuse avant ce qui a arrêté les projections de pâte.

D'un point de vue fonctionnel, ce dialogue peut être résumé de la manière suivante : A explique à B pourquoi il n'y a plus de problème sur la machine, mais cette explication n'est pas donnée et acceptée d'un seul coup. B doit exprimer ce qu'il connaît de la situation pour que A donne des explications adaptées et que B puisse en déduire ce que A voulait lui dire initialement.

Cet exemple montre bien « que les interactants en conversation traitent leurs problèmes, communicationnels, conversationnels et logiques au fur et à mesure qu'ils les rencontrent. » [op. cité, p. 262]. Aussi, dans les modèles de dialogue fondés sur la planification, la rationalité qui permet de planifier les actions n'est pas comparable à celle illustrée par l'exemple de Trognon. Les actions communicatives planifiées ne peuvent pas provenir de la « rationalité dialogique ». Cette dernière s'inscrit justement dans un modèle théorique de l'enchaînement conversationnel : la logique interlocutoire de Trognon & Brassac [Trognon & Brassac, 1992].

2.2.2 Le modèle de la logique interlocutoire

2.2.2.1 Introduction

Le modèle de la logique interlocutoire consiste en une *dialogisation* de la sémantique générale proposée par Vanderveken [Brassac & Pesty, 1999]. Il permet de rendre compte de la construction dynamique conjointe du sens en interaction, ce que ne permet pas le modèle actuel de Vanderveken. Examinons l'exemple reproduit Fig. 2.4 de quatre séquences se déroulant dans un même contexte.

Séquence 1	Séquence 2
E1 Tu as le téléphone ici L1 Oui, c'est moderne E2 Ah je n'aurais pas cru	E1 Tu as le téléphone ici L1 Oui, c'est le numéro 04-83-35-36-09 E2 Ah ben je pourrais t'appeler comme ça
Séquence 3	**Séquence 4**
E1 Tu as le téléphone ici L1 Oui, c'est moderne E2 (rires) Ah ben j'aurais bien aimé que tu me donnes le numéro	E1 Tu as le téléphone ici L1 Oui, c'est le numéro 04-83-35-36-09 E2 Euh, mais je ne te demandais pas le numéro

FIG. 2.4 – Illustration de la logique interlocutoire tirée de [Brassac & Pesty, 1999].

Ces quatre séquences peuvent toutes s'être réellement passées car elles sont toutes interprétables[8]. Par exemple, dans les séquences 1 et 3 l'énoncé E1 peut être une requête d'information paraphrasée par « je te demande si tu as le téléphone ». En revanche dans les séquences 2 et 4 l'énoncé E1 doit plutôt être considéré comme une requête d'action. Il est paraphrasé par « je te demande ton numéro de téléphone ».

Comment expliquer que ces quatre séquences soient différentes ? Ce qui différencie ces séquences est uniquement la manière dont l'interlocuteur L interprète l'énoncé initial E1. S'il l'interprète comme une requête d'information (séquences 1 et 3) son interprétation en actes (énoncé L1) est soit validée (énoncé E2 de la séquence 1) soit invalidée (énoncé E2 de la séquence 3). Il en est de même pour les séquences 2 et 4 : son interprétation de E1 comme

[8] Seule la première séquence s'est réellement déroulée.

requête d'action est validée dans la séquence 2 et invalidée dans la séquence 4. Ce qui revient à dire que dans les quatre cas, l'interprétation de l'énoncé E1 par L n'est qu'hypothétique, et qu'elle nécessite alors d'être évaluée à l'énoncé E2.

Dans d'autres dialogues, le sens d'un énoncé peut nécessiter d'autres négociations ou même ne jamais être totalement fixé. En effet, qu'est-ce qui permet à l'interlocuteur ou même à un analyste, de décider une fois pour toute du sens d'un énoncé ? La logique interlocutoire permet justement aux énoncés de ne pas prendre un statut fixe et invariable. Non seulement un même énoncé peut prendre des sens différents dans des situations différentes, mais, également, le sens co-construit peut différer selon les locuteurs dans des situations similaires.

2.2.2.2 Actes de langage et logique interlocutoire

Le modèle de Trognon & Brassac s'appuie sur une reformulation de la théorie des actes de langage de Vanderveken [Brassac, 1993]. Selon Vanderveken, un acte de langage simple est soumis à des conditions de réussite et des conditions de satisfaction [Vanderveken, 1990]. S'il est satisfait, un acte de langage dont la direction d'ajustement va du monde aux mots est nécessairement réussi. Prenons l'exemple de la Fig. 2.5 avec, d'une part, une séquence entre les locuteurs L1 et L2, et d'autre part une seconde séquence avec le même locuteur L1 et le locuteur L3 (tiré de [Brassac, 1993]).

Séquence 1	
L1.1	Tu m'as pas dit avec qui tu as mangé à midi
L2.1	J'ai mangé avec Michel

Séquence 2	
L1.1	Tu m'as pas dit avec qui tu as mangé à midi
L3.1	C'est vrai on n'a pas eu le temps d'en parler

FIG. 2.5 – Exemple de réussites interlocutoires.

Dans la première séquence L2 *satisfait* une demande en ajustant le monde aux mots de l'énoncé L1.1. Il donne en effet une réponse à la question « Avec qui as-tu mangé à midi ? ». C'est parce que L2 répond à cette question en produisant L2.1 que l'acte initial L1.1 est instancié comme demande à travers l'accomplissement *réussi* qui découle de sa satisfaction. En revanche, dans la seconde séquence l'interlocuteur L3 réagit à une assertion. Dans les deux cas les interlocuteurs L2 et L3 satisfont un acte de langage mais ce n'est pas le même. C'est parce que L2.1 satisfait une requête d'information que L1.1 acquiert une valeur communicative d'acte directif. De même, c'est parce que L3.1 satisfait une assertion que L1.1 conserve son statut d'assertion.

2.2.2.3 Principe de l'enchaînement conversationnel

Le principe de l'enchaînement conversationnel que propose Trognon & Brassac est celui des ethnométhodologues. Il peut être résumé par le mécanisme en trois temps suivant (L1 et L2 sont les participants) :

1. Production de l'énoncé E1 par L1.
2. Production d'un énoncé E2 par L2 qui communique la façon dont L2 interprète l'énoncé E1.
3. Production d'un énoncé E3 par L1 validant, invalidant ou modulant l'interprétation en actes par L2 de l'énoncé E1.

Autrement dit, dans le meilleur des cas, la valeur conversationnelle de l'acte énoncé n'est effective qu'après la réponse de l'interlocuteur car elle est fonction de cette réponse. Dans les autres cas, elle est sujet à négociation si l'interprétation en acte de E1 produit par L2 est fortement hypothétique.

Ce mécanisme débouche sur l'effet d'intercompréhension : les participants se comprennent mutuellement car ils ont construit un sens commun qui n'est d'ailleurs pas vraiment accessible à un observateur extérieur. « Le dialogue résout d'une certaine manière "le problème de l'interprétation", c'est-à-dire le problème pragmatique par excellence [...] de l'inaccessibilité des intentions des communicateurs. » [Trognon, 1997a]. Selon ces auteurs, il faut cependant « disposer d'une théorie des relations entretenues par les actes de langages successifs d'une conversation ». Ils proposent pour cela d'articuler la logique interlocutoire au modèle hiérarchique de l'école genevoise. Un exemple de composition est donné dans [Trognon, 1995].

Sur ce dernier point, nous souhaitons apporter une remarque. Le but que poursuivent ces auteurs n'est pas d'améliorer le modèle de Roulet, en dépit du fait que leur logique permet de résoudre les problèmes de motivation théorique dont souffrent les procédures d'assignation des composants directeurs et subordonnés du modèle genevois. Leur motivation est purement épistémique : « C'est donc à partir [du plan des structures] que devraient d'une manière ou d'une autre pouvoir être engendrés les effets des conversations et les types de conversations. ». [op. cité, p. 82]. De notre point de vue, la structure d'un dialogue ne permet pas d'expliquer comment les participants interprètent les énonciations successives de ce dialogue, en revanche l'inverse est possible *a posteriori* : savoir comment sont interpréter les énoncés devraient aboutir à sa structure globale. Nous continuons ce débat ci-après afin d'introduire le modèle de Nicolle & Saint-Dizier De Almeida.

2.3 Vers un modèle des interactions langagières

2.3.1 Structuration des conversations et effets conversationnels

Selon Trognon les propriétés des séquences interlocutoires peuvent être situées sur trois plans [Trognon, 1995]. Le premier plan est celui des effets conversationnels : effet d'intercompréhension, émergence de l'identité psychosociale et les rapports sociaux en général. Le second plan est celui des types conversationnels. Enfin, le dernier plan est celui des structures conversationnelles :

> « Tout comme une phrase constitue une structure de constituants, une conversation est fondamentalement une structure d'énonciations. Étudier une conversation (ou une séquence conversationnelle), c'est donc lui associer une structure, c'est-à-dire le système des relations qui tissent les énonciations qui lui succèdent. » [Trognon, 1995, p. 81].

Trognon affirme que le troisième plan est le plus fondamental et « tout projet d'investigation des conversations doit donc s'intéresser prioritairement aux structures conversationnelles » [op. cité, p. 82].

En tant qu'informaticien, nous nous sommes posé la question de savoir ce que peut apporter un modèle élaboré des structures conversationnelles tel que celui de Roulet[9] [10]. En ce qui concerne le dialogue homme-machine nous pensons que ce n'est pas aux structures qu'il faut s'intéresser en priorité, mais aux effets conversationnels, le premier plan mentionné par Trognon. Pour argumenter notre point de vue, nous prendrons l'exemple du modèle de Vilnat qui reprend le modèle genevois [Vilnat, 1997].

En réaction à la complexité accrue des systèmes actuels fondés sur la planification, l'idée initiale est « de traiter à part un certain nombre de problèmes propres du dialogue, en les considérant comme des obligations à traiter à part de la planification proprement dite, qui n'a lieu que lorsque ces obligations ont été respectées. » [op. cité, p. 154]. Dans le système Diabolo développé par Vilnat, la connaissance de la structure courante du dialogue est censée aider la gestion du dialogue. Par exemple, des règles permettent de connaître si un énoncé est une intervention initiative, réactive ou évaluative.

Il nous semble difficile de modifier le modèle genevois d'analyse *a posteriori* en un modèle d'analyse dynamique. Avec le modèle de Roulet, l'analyse se fait **globalement**. L'aspect fonctionnel des énoncés est figé selon l'interprétation de celui qui analyse le dialogue en considérant la totalité du dialogue. Convertir le modèle genevois en un modèle d'analyse dynamique **locale** revient encore à **figer** le statut des énoncés (comme l'a fait Pernel [Pernel, 1994]). Comment dire par exemple qu'un énoncé fait partie d'une intervention complexe sans connaître l'échange complémentaire qui va se dérouler ensuite (voir Fig. 2.6) ?

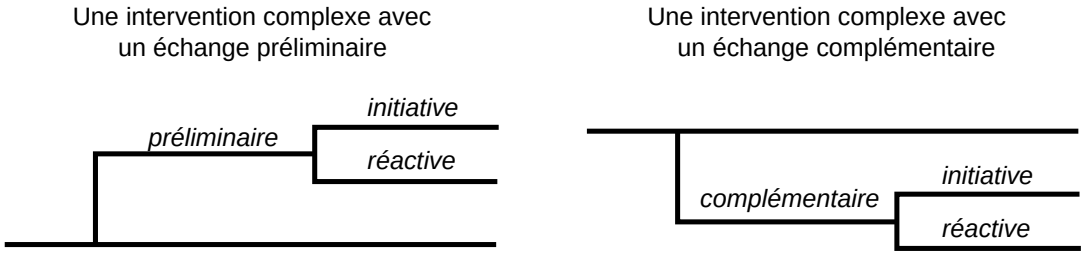


FIG. 2.6 – Les interventions complexes de Roulet d'après [Vilnat, 1997, p. 165].

[9] Nous avons pris cet exemple car c'est le modèle structurel le plus connu et le plus repris par la communauté française de recherche en DHM.

[10] E. Roulet, A. Auschlin, J. Moeschler, C. Rubattel et M. Schelling. *L'articulation du discours en français contemporain*. Peter Lang, Berne, 1985.

Certes, des règles peuvent toujours analyser, *après,* qu'il s'agissait d'une intervention complexe, mais cela a-t-il aidé le système au moment opportun ? Cela aide-t-il la machine de savoir maintenant qu'il s'agit d'une intervention complexe ? La seule justification semble être de « conserver l'information concernant le déroulement du dialogue (de façon très schématique : pour savoir où et comment on a répondu à une demande) [...] » [Vilnat, 1997, p. 164]. Dans Diabolo, la structure du dialogue est en fait nécessaire au gestionnaire de thèmes pour pouvoir revenir sur un thème principal après un sous-dialogue. Sur ce point nous sommes d'accord, mais le modèle genevois n'est pas adéquat. Ce genre d'analyse structurelle poussée n'est pas nécessaire pour gérer un dialogue en temps réel car, selon le modèle de la négociation, le dialogue « se construit interactivement pas-à-pas au fur et à mesure de son déroulement » [Trognon, 1996, p. 32].

Certes, en général un dialogue finalisé n'est pas linéaire. Des transgressions sont souvent nécessaires pour éclaircir certaines choses, pour avoir des confirmations, etc. Des reformulations, des paraphrases sont normales dans un dialogue naturel et encore plus dans le dialogue artificiel où l'utilisateur a souvent besoin d'échanges confirmatifs. Le modèle genevois prend particulièrement en compte l'aspect structurel des dialogues naturels. Cependant, ce que constate Luzzati, c'est qu'il n'est pas adapté au DHM :

> « Ce modèle, qui permet de reproduire une représentation arborescente, est fondamentalement un modèle descriptif, qui permet de rendre compte d'une conversation une fois celle-ci achevée. Le problème en DHM est à l'inverse de rendre compte d'un dialogue au cours de son déroulement, de façon dynamique, afin de pouvoir autant que possible en garder le contrôle. » [Luzzati, 1995b, p. 34].

Selon Luzzati, le dialogue finalisé se structure selon deux axes :

> « Le dialogue peut s'orienter dans deux directions : soit demande d'information et délivrance des renseignements s'enchaînent sans difficulté, et il s'agit d'un **dialogue régissant** ; soit des demandes de précision, d'explication, de confirmation ou de reformulation doivent intervenir pour qu'une question ou une réponse soit acceptée, et il s'agit d'un **dialogue incident**. » [Luzzati, 1995b, p. 36].

Un *écart interactionnel* peut être calculé avec une *fonction interactionnelle* du type $EC = (a \times AR) + (b \times AI) + (c \times AE)$ où $\{a, b, c\}$ sont trois coefficients multiplicateurs à définir selon la souplesse désirée du dialogue et le type d'application, et $\{AR, AI, AE\}$ sont trois variables correspondant respectivement à la position courante sur l'axe régissant, à la position sur l'axe incident et à la position sur un troisième axe mesurant la complexité (en nombre d'imbrications) de l'incidence.

Le modèle de Luzzati est repris par Grau et Vilnat (voir [Grau *et al.*, 1994] et [Vilnat, 1997]). Dans le système Diabolo, la gestion du dialogue s'appuie, entre autres, sur deux fonctions interactionnelles $F1$ et $F2$ permettant d'éviter les possibilités d'impasse : « $F1$ et $F2$ permettent respectivement d'évaluer le dialogue à un instant donné et de décider de la conduite à tenir, à savoir s'il est pertinent d'introduire un nouvel échange préliminaire. » [Grau *et al.*,

1994, p. 18]. $F1$ est indépendant de la tâche. Il indique la difficulté de répondre à une question principale. $F2$ atténue les résultats de $F1$ en tenant compte de la tâche. Ces deux fonctions sont calculées de la manière suivante :

$$F1 = a \times AE + AI$$
$$F2 = F1 \times log(AR)$$

Pour notre part, la validité de cette fonction nous semble assez discutable. Même si les usagers humains sont bien sûr capables de s'adapter, ce qui est acceptable pour certaines personnes ne l'est peut-être pas pour certaines autres et réciproquement. L'apport pratique d'un tel contrôle interactionnel est tout de même d'évaluer quand une demande de reformulation telle que « je n'ai pas compris, voulez-vous répéter s'il vous plaît » ne peut plus être posée, quand une telle conduite n'est plus tenable.

Le modèle des interactions langagières que proposent Nicolle & Saint-Dizier De Almeida ne cherche pas à modéliser la structuration des conversations. Nous présentons maintenant en détail ce modèle avant de revenir sur la structuration des dialogues proposée par Luzzati dans un prochain paragraphe (cf. Sec. 2.4.1.2).

2.3.2 Les interactions langagières

Le modèle de Nicolle & Saint-Dizier De Almeida est un modèle complet des interactions verbales qui s'appuie sur un grand nombre de concepts et de notions repris de l'analyse conversationnel (Goffman, Bange, Levinson) et de l'analyse du discours (Austin, Sperber & Wilson, Searle & Vanderveken, Trognon & Brassac, Vernant, Coursil) [Nicolle & Saint-Dizier De Almeida, 1999a]. C'est un modèle dont le but est de représenter les connaissances et les processus mis en jeu au cours d'un dialogue par les acteurs de ce dialogue. La notion fondamentale défendue est la notion de *valeur interlocutoire*, c'est-à-dire une valeur illocutoire partagée par les interlocuteurs et qui a fait l'objet d'une négociation.

2.3.2.1 Notions et concepts

Les auteurs soutiennent, avec Trognon & Brassac, que le sens des actes produits en interaction est l'objet d'une co-construction. Le sens ne préexiste pas à l'interaction, car l'expression linguistique et le contexte d'énonciation ne suffisent pas à déterminer le sens intentionné. D'une part, le contexte interprétatif est seulement supposé partagé. Les acteurs ne partagent pas forcément le même contexte cognitif. D'autre part, l'expression linguistique n'est pas un code à décoder permettant de délivrer le sens. À chaque énoncé ne correspond pas une valeur illocutoire, même s'il existe des marqueurs linguistiques donnant des indices qui serviront à la construction du sens (par exemple, les connecteurs pragmatiques, les déictiques, etc.) Une production verbale est donc porteuse d'un potentiel d'actes illocutoires et perlocutoires. L'allocutaire va « sélectionner » l'un de ces potentiels.

Selon Nicolle, dans une situation de dialogue, les agents sont situés dans un monde à quatre dimensions :

- Le monde physique fait référence aux choses matérielles et à leurs transformations ; cela inclut le temps et l'espace.
- Le monde social fait référence aux relations inter-agents (organisations de ces relations, statuts institutionnels, rôles, position sociale principalement).
- Le monde du soi fait référence à l'agent en tant qu'individu ayant des sensations, des sentiments ; il est doué de capacité cognitive (raisonnement, mémorisation, catégorisation, intentions principalement).
- Le monde sémiotique fait référence au monde des mots. C'est un monde qui rend compte des trois autres, ce qui permet la communication entre les agents. Il est donc partagé (grâce à des routines interactives, des ethnométhodes, des conventions, des systèmes de signes, des langues), mais également réflexif car il rend compte de lui-même. Ce monde inclut les actes langagiers mais également non-langagiers (geste, non-actions[11]). Pour interagir, les agents ne décrivent pas un monde partagé, mais montrent les éléments pertinents pour ce qu'ils intentionnent communiquer.

Bien que ces mondes indissociables s'auto-déterminent, les différencier est nécessaire car selon que le focus du dialogue porte sur l'un ou l'autre de ces mondes, l'enchaînement conversationnel ne va pas être le même. Par exemple, si on parle pour convaincre, l'enjeu est de modifier le monde mental de son partenaire. Si on parle pour maintenir une relation amicale, l'enjeu est de ne pas détériorer le monde social. Enfin, si on parle pour qu'une porte soit fermée, l'enjeu est la modification du monde physique. Néanmoins, dans chacune de ces interactions, tous les autres mondes interviennent plus ou moins fortement.

Par ailleurs, un élément du monde de l'agent fait souvent partie de plusieurs dimensions à la fois. Les auteurs prennent l'exemple des objets techniques : ce sont des objets matériels construits (monde physique), vendus, achetés et utilisés conjointement (monde social) ; ils appartiennent à un individu (monde du soi) ; ils peuvent être décrits et faire l'objet de conversations (monde sémiotique).

Tout en s'appuyant sur les quatre dimensions du monde, les interactions des agents le construisent.

2.3.2.2 La valeur interlocutoire des énoncés

Dans un énoncé proféré en interaction, il faut distinguer trois aspects. Ces aspects rendent compte des échanges interlocutoires.

La valeur sémantique de l'énoncé est calculée à partir des éléments linguistiques de l'énoncé lui-même et du co-texte (les énoncés précédents). C'est une valeur partagée correspon-

[11] Une non-action est le fait de ne pas faire une action attendue. Par exemple, le fait de ne pas exécuter un ordre donné malgré le fait qu'on en soit capable.

dant à la notion de signification des sémioticiens [Beust, 1998].

La valeur illocutoire est inférée par l'allocutaire à partir de la valeur sémantique, du contexte général et des intentions attribuées au locuteur. C'est une valeur non-préméditée par le locuteur. Cette valeur peut différer de celle du locuteur.

La valeur interlocutoire (« le sens » de l'énoncé) correspond à la signification pragmatique qui résulte de la négociation de la valeur illocutoire. C'est une valeur non-préméditée et partagée.

Les actes illocutoires sont définis par quatre composants. Ces composants sont vus comme des contraintes pour déterminer la valeur des actes illocutoires. Les auteurs redéfinissent la notion d'acte de langage classique : une force illocutoire s'appliquant à un contenu propositionnel. Ces deux éléments sont décomposés de la manière suivante.

- La monstration d'un acte : ce sont les choses évoquées par l'acte (*ce dont on parle*). Cela peut être des objets, des propriétés d'objets, des relations, des états ou des procès, etc.
- La prédication d'un acte : c'est l'état ou le procès portant sur les choses évoquées (*ce qu'on en dit*).
- La force illocutoire d'un acte : c'est la direction d'ajustement (*pour quelle raison on le dit*). Les auteurs distinguent :
  - la force assertive : les mots s'ajustent au monde ;
  - la force déclarative : du fait de l'énonciation, les mots et le monde s'ajustent ;
  - la force directive : le monde doit s'ajuster aux mots par l'action de l'allocutaire ;
  - la force commissive : le monde doit s'ajuster aux mots par l'action du locuteur.
- La modalité : c'est le mode d'accomplissement de l'acte (*comment on le dit*).

À la différence de Searle, les expressifs sont une sous-classe des assertifs. Pour Nicolle, la direction d'ajustement des expressifs n'est pas nulle. Dans les actes illocutoires de force expressive les mots s'ajustent au monde personnel du locuteur. Deux exemples de décomposition sont donnés Tab. 2.1.

2.3.2.3 Un modèle de la gestion conversationnelle

Les auteurs reprennent la logique interlocutoire de Trognon & Brassac et l'englobent dans le cadre général de la théorie du transfert de Coursil[12]. Cette théorie traite de la façon dont les informations sont véhiculées entre les acteurs d'un dialogue. Le transfert se fait sur trois niveaux :

- La physique du transfert traite de la transmission des signaux.
- La logique du transfert traite des tours de parole des acteurs. Elle correspond à l'aspect dialogual défini par Vernant [Vernant, 1997a].

[12] Coursil, J., 1993, Dialog, the semiology of transfert, in *2ème Congrès Européen de Systémique*, AFCET, CSCI, Prague.

Énoncé	*vous préférez les documentations en ligne ou les documentations papier*
Monstration	temps présent "vous" "documentations en ligne" (X) "documentations papier" (Y)
Prédication	Verbe : préférer Sujet : l'allocutaire Complément : OU(X,Y)
Modalité	"vous" de majesté
Force illocutoire	question

Énoncé	*Ben non on n'est pas dans l'bon truc*
Monstration	temps présent "les interlocuteurs" "pas dans l'bon truc"
Prédication	Verbe : être Sujet : "les interlocuteurs" Circonstanciel de lieu : "pas dans l'bon truc"
Modalité	exclamatif
Force illocutoire	assertion

TAB. 2.1 – Exemples d'actes illocutoires d'après [Nicolle & Saint-Dizier De Almeida, 1999a].

- La sémiotique du transfert traite de la représentation mutuelle du dialogue du point de vue des acteurs. Elle correspond à l'aspect dialogique défini par Vernant (op. cité).

Nous commençons par présenter brièvement la sémiotique du transfert. Nous présentons ensuite les mécanismes mis en jeu à partir des représentations définies dans la sémiotique du transfert et de la définition des valeurs illocutoires et interlocutoires vues précédemment.

2.3.2.3.1 Sémiotique du transfert

La sémiotique du transfert concerne les représentations des agents nécessaires aux échanges conversationnels. Si A et B sont les deux acteurs d'un dialogue, alors chacun d'eux se construit une image de l'autre, une image de lui-même et une image du terrain commun. Cette dernière est nécessaire pour rendre compte de la co-référenciation utilisée dans les énoncés afin d'assurer le transfert des informations malgré la polysémie naturelle des langues. Elle s'affine à travers chaque énoncé.

L'image de l'autre est nécessaire ne serait ce que parce que l'interlocuteur attribue des intentions au locuteur pour interpréter les énoncés. Au départ, si on ne connaît pas l'autre acteur, on pose l'hypothèse qu'il est semblable à nous-même. Cette image est ensuite affinée à partir des énoncés émis.

L'image de soi existe de part la réflexivité des agents (l'une des caractéristiques de la conscience humaine). Elle est également modifiée par l'interaction à partir de l'image retournée

par le dialogue que l'autre a de moi-même. C'est ce que Trognon nomme l'émergence de l'identité psychosociale.

Puisque l'image de l'autre inclut ma propre image et ainsi de suite, on pourrait penser que ce modèle est infini (voir Fig. 2.7). Pourtant, les niveaux supérieurs sont déployés uniquement si, à un instant donné, les deux acteurs ne partagent pas le même terrain commun (ou si un acteur ment). Dans les autres cas, le développement d'ordre un est suffisant (voir Fig. 1.2 p. 9).

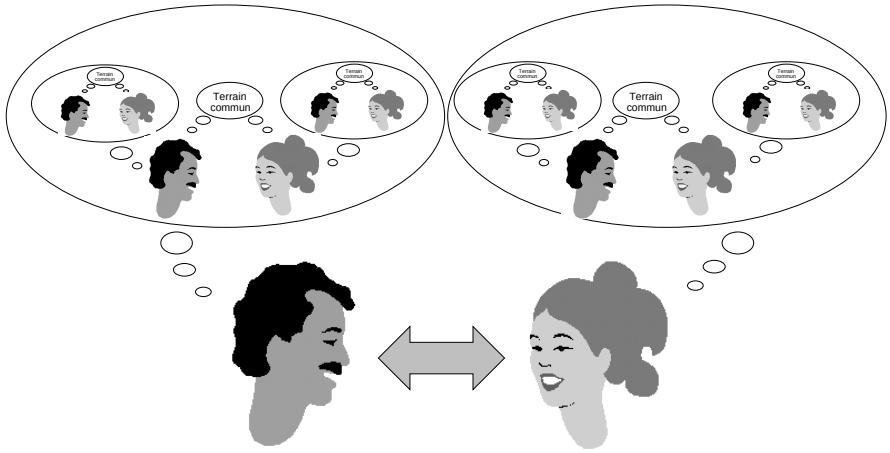


FIG. 2.7 – Représentation mutuelle des acteurs d'un dialogue (de second ordre).

À la suite de Coursil, Nicolle et Saint-Dizier De Almeida affirment que pour construire un analogue machine d'un humain dialoguant, la machine doit elle aussi fonctionner avec ces types de représentations. La différence est qu'ils doivent être à l'image d'elle-même afin qu'un apprentissage puisse ce faire car celui-ci n'est possible que si la machine apprend par elle-même c'est-à-dire lorsque ses représentations sont conformes à elle-même (voir Fig. 2.8). Ceci est résumé par la formule suivante de Coursil reprise par Lehuen : *il faut plonger la machine dans l'activité signifiante de la langue et non la langue dans la machine* [Lehuen, 1997b].

2.3.2.3.2 Les mécanismes d'interprétation

Les mécanismes d'interprétation doivent être valables aussi bien pour les actions verbales que pour les actes gestuels et les non-actions. En reprenant les notions d'actes locutoires, d'actes illocutoires et d'actes perlocutoires, Nicolle explique que l'acte illocutoire est inféré à partir de l'acte locutoire (ensemble d'indices mis dans le terrain commun) et des effets perlocutoires que l'allocutaire imagine être intentionnés par le locuteur. L'allocutaire produit alors une interprétation en actes, résultat de son raisonnement, qui sera évaluée par le locuteur.

Explicitons plus précisément les éléments intervenant dans le travail interprétatif. Tout d'abord l'allocutaire doit avoir connaissance des termes employés par le locuteur. Si ce n'est pas le cas, il peut soit poser une question d'éclaircissement, soit poser une hypothèse de signification. Ensuite, il recherche des éléments d'intentionnalité et de sincérité du locuteur.

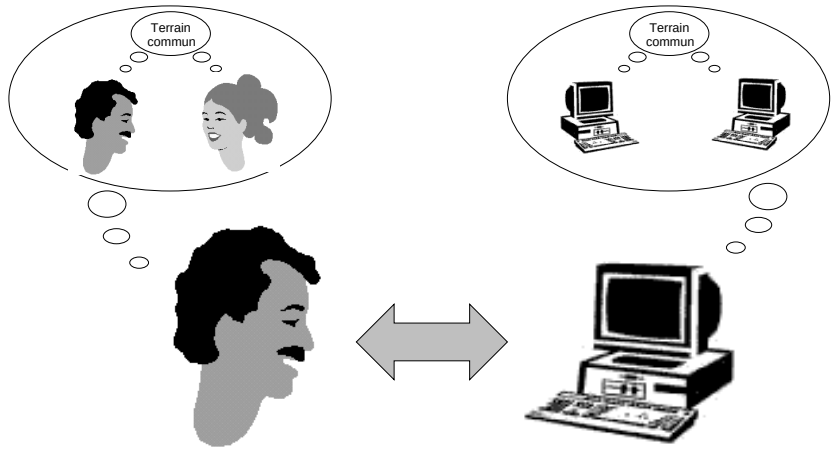


FIG. 2.8 – Représentation mutuelle minimale des acteurs d'un dialogue homme-machine.

Ici, les auteurs mentionnent comme indices : les unités lexicales, le mode et le temps, la structure syntaxique, la prédication, etc. Interviennent également des éléments de la relation sociale (type, objectifs et finalité de l'interaction) et des éléments du terrain commun (ce qui s'est passé avant et ce qui risque de se passer après, notamment les fonctions illocutoires et interactives potentielles de l'acte).

La valeur illocutoire attribuée est transmise par l'allocutaire du fait de ses réactions : si elle est validée par le locuteur, c'est que les effets perlocutoires inférés sont acceptés. Dans le cas contraire, l'interprétation en actes n'est pas ratifiée et une négociation est ouverte explicitement. Dans les deux cas, une intercompréhension doit finalement émerger : c'est la valeur interlocutoire.

En reprenant la notion de réussite d'un acte de Vanderveken, les auteurs affirment qu'un acte illocutoire est **réussi** lorsqu'il lui est attribuée une valeur interlocutoire. Ils définissent également un ensemble de contraintes sur la **satisfaction** d'un acte. Elles portent sur l'acceptabilité et la satisfiabilité nécessaire à la satisfaction d'un acte. Nous nous contenterons ici d'en donner la liste :

Contrainte **1** *l'acceptabilité*
Un acte réussi est acceptable si l'allocutaire accepte la position sociale qu'occupe le locuteur en énonçant l'acte.

Contrainte **2** *la capacité à satisfaire*
Un acte réussi est satisfiable :

Contrainte **2a** : si les capacités du locuteur (pour un commissif, un assertif) ou de l'allocutaire (pour un directif) permettent la satisfaction de l'acte.

Contrainte **2b** : si l'état du monde permet la satisfaction de l'acte.

Contrainte **3** *la volonté de satisfaire*
Un acte réussi est satisfiable si les attitudes mentales de l'allocutaire autorisent la satisfaction

de l'acte :

Contrainte **3a** : si l'allocutaire croit en la sincérité du locuteur ;

Contrainte **3b** : si l'allocutaire croit en la véridicité du contenu communiqué ;

Contrainte **3c** : si l'allocutaire a le désir et l'intention de satisfaire l'acte.

2.3.2.3.3 Les procédures de gestion conversationnelle

Les agents acquièrent en dialoguant une compétence sous forme de procédures partagées. Ce sont des *ethnométhodes* qui facilitent le travail interprétatif. Par exemple, si le locuteur veut donner un ordre, il s'assure tout d'abord que l'allocutaire est capable de le satisfaire. S'il veut convaincre de quelque chose, il essaie d'engager l'allocutaire sur les prémisses menant à conclure de ce quelque chose. L'exemple de dialogue authentique Fig. 2.9 illustre ce propos ainsi que ceux portant sur les mécanismes interprétatifs évoqués précédemment.

PF1 : Donne un bonbon
GM2 : C'est comme ça que tu demandes toi ?!
PF3 : s'il te plaît
GM4 : t'en as assez mangé pour aujourd'hui
GM5 : eh pis j'en ai plus
PF6 : menteuse
GM7 :*GM montre à PF une boîte vide.*
PF8 : Papi il en a mis dans l'autre boîte
GM9 : Bon c'est le dernier

FIG. 2.9 – Exemple de dialogue illustrant les procédures de gestion conversationnelle tiré de [Nicolle & Saint-Dizier De Almeida, 1999a].

La scène se passe entre une grand-mère (GM) et son petit-fils (PF). Ce dialogue se découpe en trois parties. La première partie porte sur la réussite de l'acte PF1. Les deux autres parties sont des enchaînements sur, premièrement, l'acceptabilité de PF1, et, deuxièmement, la satisfiabilité de PF1.

Réussite de l'acte PF1 En énonçant GM2, GM communique qu'elle affecte à PF1 la valeur d'une requête d'action. Avec PF3, PF valide cette interprétation en actes, puisqu'il ne le contredit pas. La valeur illocutoire de PF1 est donc clairement fixée.

Acceptabilité de PF1 Cependant, si l'acte PF1 est réussi, il n'en est pas pour autant satisfait. L'énoncé GM2 vise également à signifier que cet acte est inacceptable tel qu'il est énoncé par PF. La réponse PF3 vise à rendre la requête d'action acceptable.

Satisfiabilité de PF1 La requête d'action semble maintenant accepté puisqu'en énonçant GM4, GM enchaîne sur la troisième contrainte (3c) de la satisfaction de l'acte initial : elle ne veut pas le satisfaire. Même si elle le voulait, elle ne pourrait pas puisqu'elle affirme en GM5 qu'il n'y a plus de bonbon (contrainte 2b).

En PF6, par son énoncé PF6, PF réfute l'acte transparaissant dans MG5. PF ne peut pas

le satisfaire puisque, en mentant, la contrainte 3a n'est pas levée. Pour rendre satisfiable l'acte énoncé en PF6, MG montre alors une boîte vide.
Cependant, PF ne satisfait toujours pas l'acte GM5. Même si la contrainte 2b est levée (l'état du monde autorise la satisfaction de l'acte GM5), la contrainte 3a ne l'est pas encore. En énonçant PF8, il communique un argument justifiant sa non-satisfaction de l'acte GM5. Ce faisant, GM n'a plus d'argument, et en énoncé GM9 elle satisfait la requête d'action initiale PF1, ce qui satisfait également l'acte PF8.

2.3.2.4 Critique du modèle de Nicolle

Comme nous le disions au départ, il s'agit d'un modèle complet dégageant un cadre conceptuel clairement posé. Nous ferons tout de même deux remarques.

La première est que de nombreux aspects des processus cognitifs des interactions sont encore à préciser en vue d'une implémentation informatique. Le problème est la mise en œuvre concrète du modèle : par exemple, quels sont les éléments contenus dans les différentes représentations des interactants, en particulier les éléments du terrain commun ? Ce modèle apparaît donc encore trop théorique et pas assez opérationnel.

La seconde remarque concerne les aspects sociaux du modèle. Il est dit que des éléments de la relation sociale entre le locuteur et l'interlocuteur interviennent dans le travail interprétatif de ces deux participants. Il est clair que, *chez les humains*, le rôle et le statut social interviennent dans les interactions verbales. Je ne dialogue pas de la même manière quand je me trouve avec un ami ou quand je suis avec mon banquier. Le point de divergence provient de l'intrusion de la dimension sociale dans le dialogue homme-machine. Dans l'absolu, idéalement, nous devrions la prendre en compte, mais, dans la réalité, c'est sans doute la dimension que l'on maîtrise encore le moins. Nous pensons donc que, dans une finalité opérative, cet aspect doit être dans un premier temps délibérément négligé. Nous y reviendrons sans aucun doute lorsque les autres aspects, plus pressants et plus accessibles, seront mieux maîtrisés.

2.4 Le modèle COALA

2.4.1 Présentation du modèle COALA

Le modèle COALA a été développé par Lehuen [Lehuen, 1997b]. Il s'agit d'un modèle calculatoire et dynamique du dialogue homme-machine finalisé indépendant du champ d'application.

La conception de ce modèle a tout d'abord été guidée par la volonté de prendre en compte la *non-normativité* du langage dans les interactions langagières. En effet, le locuteur est aussi un « créateur linguistique », surtout dans les situations d'interaction : « on aura beau s'efforcer de répertorier mille possibilités pour exprimer quelque chose, l'utilisateur s'exprimera de façon inattendue et souvent à l'aide d'une syntaxe originale. » Un premier problème auquel a

voulu répondre l'auteur a donc été celui de la représentation des connaissances linguistiques en machine. Au lieu de s'appuyer sur des modèles descriptifs du langage qui représentent « une langue théorisée, normalisée », l'auteur a souhaité intégrer dès le départ une capacité d'apprentissage linguistique, de sorte que le système apprenne par le biais du dialogue les expressions et les termes réellement employés dans une situation de dialogue finalisé.

La seconde préoccupation de l'auteur a été de prendre en compte la *coadaptation* des locuteurs, c'est-à-dire le fait que dans tout dialogue les locuteurs adaptent leurs comportements à ceux des interlocuteurs. Lehuen met ici en avant *l'intersubjectivité* du dialogue : les représentations du locuteur ne sont pas forcément les mêmes que celles de son interlocuteur. Lehuen préconise alors la mise en œuvre d'un système ayant une réelle capacité conversationnelle, c'est-à-dire une capacité de gérer le dialogue en *temps réel*. Avec le modèle COALA, la structure d'un dialogue se met en place avec et au cours du dialogue lui-même et non *a posteriori*. Il s'agit d'une structure subjective qui peut être différente selon le point de vue pris, celui du locuteur ou celui de l'interlocuteur. Ce n'est donc pas un modèle descriptif comme par exemple le modèle de l'école genevoise[13]. C'est surtout sur ce dernier point que l'auteur rejoint nos préoccupations : le dialogue homme-machine n'est pas un dialogue homme-homme. Il faut définir un modèle spécifique, et c'est ce que propose Lehuen. Ces deux préoccupations ont été abordées à travers un modèle de dialogue *hypothético-déductif*.

2.4.1.1 Le modèle hypothético-déductif

Le principe du modèle hypothético-déductif est de pouvoir faire des hypothèses et des déductions sur la compréhension des énoncés de l'utilisateur [Lehuen *et al.*, 1996]. Le système est alors capable de poser des questions sur les mots ou les expressions qu'il n'a pas compris par rapport à des *attentes*.

Dans COALA, un dialogue se compose d'unités minimales d'interactions (notée **UMI**) complètes ou hypothétiques, instanciées dans le cours du dialogue (cf. Fig. 2.10). Une UMI est une structure indicée qui représente une paire adjacente non-symétrique, c'est-à-dire un énoncé du système et un énoncé de l'utilisateur. Elle comprend aussi des éléments du contexte d'énonciation permettant de confronter, d'une part, les *attentes* nécessaires au déroulement du dialogue, et d'autre part les *offres* provenant de l'interprétation des énoncés de l'utilisateur. Plus précisément, une UMI est constituée de plusieurs champs instanciés au cours d'un tour de parole. Le tableau 2.2 donne une description des différents champs de l'UMI du point de vue de la machine : $\{E_i, M_i, Att_{Mi}, H_i, Rep_{Hi}, Hyp_i\}$. Du point de vue de l'usager les champs de l'UMI sont en partie inversés : $\{E_i, H_i, Att_{Hi}, M_i, Rep_{Mi}, Hyp_i\}$.

La manière dont s'enchaînent les UMI dépend du résultat de la confrontation entre les attentes et les offres. Une attente est définie par l'auteur comme étant ce qui est attendu par le système au regard de l'état courant de la tâche : « les attentes de la machine peuvent être

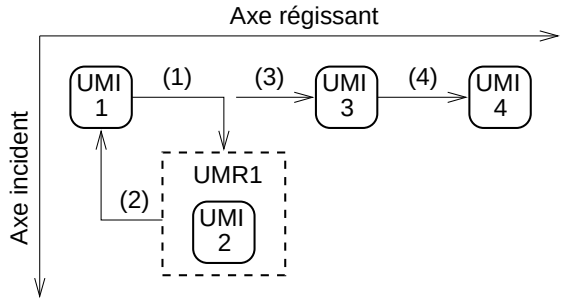
[13] E. Roulet, A. Auchlin, J. Mœschler, C. Rubattel et M. Schelling. *L'articulation du discours en français contemporain*. Peter Lang, Berne, 1985.

CHAMP	INTITULÉ	DESCRIPTION
E_i	État courant de la tâche	Pointeur sur une description de la tâche
M_i	Intervention de la machine	Suite de symboles produite par la machine
Att_{Mi}	Attentes de la machine	Contexte pour interpréter l'intervention H_i
H_i	Intervention de l'usager	Suite de symboles perçue par la machine
Rep_{Hi}	Granules de compréhension	Éléments d'interprétation de l'intervention H_i
Hyp_i	Hypothèses de la machine	Catégorisations, implicites, anaphores, etc.

TAB. 2.2 – La structure d'une UMI du point de vue du système.

exprimées en termes d'objets du domaine d'application et d'actes de dialogue orientés par la tâche. ». Les offres sont des « granules de compréhension » produits par l'analyse sémantique des énoncés H_i de l'usager : « Un granule de compréhension peut être, selon le formalisme choisi, un acte de langage, un objet du domaine d'application, un schéma thématique ou une structure complexe désignant un concept. » Leur formalisme doit être compatible avec celui des attentes. Les limites de cette approche sont discutées à la fin de ce chapitre.

M1 : Médiathèque Louis Aragon bonjour
H1 : Avez-vous Hernani de Victor Hugo ?
M2 : Est-ce que "Avez-vous Hernani" est le titre ? *
H2 : non
M3 : Je recherche avec Hernani conne titre et Victor Hugo comme auteur ...
Nous avons 'Hernani" de Victor Hugo.
Ce document vous intéresse-t-il ?
H3 : oui
M4 : Vous trouverez "Hernani" dans Théâtre Romantique No 4.
Vous avez jusqu'au mardi 8 avril.
H4 : au revoir



** Le système Coala est ici en phase d'apprentissage*

FIG. 2.10 – Un exemple de dialogue et sa structuration en UMI.

2.4.1.2 Structuration du dialogue

Le modèle COALA reprend la structuration du dialogue selon les deux axes définis par Luzzati (cf. Sect. 2.3.1). Il en présente cependant une explication plus formelle, car la struc-

turation du dialogue n'est plus un fait abstrait observé après coup, mais un élément géré dynamiquement par le locuteur lui-même. Les locuteurs deviennent acteurs comme dans le modèle de Nicolle & Saint-Dizier De Almeida.

Le principe a déjà été évoqué : si à un instant donné du dialogue les attentes liées à la situation interactionnelle ne sont pas satisfaites par les offres provenant de l'interprétation de l'énoncé de l'utilisateur, une séquence incidente est ouverte pour essayer de réduire cette différence. Une séquence incidente peut être constituée d'une ou plusieurs UMI ou même d'une autre séquence incidente. On la nomme **UMR** pour Unité Minimale de Résolution. Le cycle de base récursif est le suivant :

1. À l'instant i le système se trouve dans un certain état E_i engendrant un certain nombre d'attentes Att_{Mi}.
2. Une question M_i est posée à l'instant t pour faire en sorte de satisfaire ces attentes.
3. Le système analyse l'énoncé H_i donnant pour résultat les granules représentés par Rep_{Hi}.
4. Le système évalue alors l'UMI$_i$. Il compare les attentes Att_{Mi} et les offres Rep_{Hi}.
   (a) Si l'évaluation est positive, le dialogue continue sur l'axe régissant. Une UMI$_{i+1}$ est créée, de nouvelles attentes Att_{Mi+1} prennent place puisque l'état est maintenant E_{i+1}.
   (b) Si l'évaluation est négative, le dialogue continue sur l'axe incident. Une UMR$_j$ est ouverte pour résoudre le problème de compréhension de l'énoncé H_i. Il y aura clôture de l'UMR$_j$ puis retour automatique à l'UMI$_i$ lorsque toutes ses attentes Att_{Mi} seront satisfaites au cours du dialogue incident.

Les UMI peuvent servir également à poser des hypothèses sur les mots ou les expressions que le système n'a pas analysés, ou à forcer le retour à l'axe régissant si l'incidence devient trop forte. En effet, lorsque le système ne comprend pas du tout ou seulement partiellement l'énoncé de l'utilisateur, le fait qu'il ait certaines attentes à cet instant donné permet de poser l'hypothèse que ce qui n'est pas compris correspond cependant bien à ces attentes. Des questions sont alors posées à l'utilisateur pour valider ou invalider les hypothèses qui ont été faites pour essayer d'interpréter tout de même l'énoncé (cf. exemple Fig. 2.10). Cette capacité expérimentale est la base d'un processus d'apprentissage d'unités lexicales et de patterns syntaxiques [Lehuen, 1997a] [Lehuen & Luzzati, 1999].

2.4.2 Apports et limites du modèle COALA

Ce modèle nous a particulièrement intéressé pour deux raisons. Premièrement, il a l'avantage d'être un modèle ouvert à toute modification et extension. On peut par exemple étendre le nombre de champs d'une UMI ou modifier la signification de ceux proposés par l'auteur.

Deuxièmement, ce modèle est à notre sens proche de celui de Trognon et Brassac. Il en constitue en quelque sorte une opérationnalisation (il s'agit d'un modèle informatique contrairement à celui de Trognon et Brassac). En effet, avec COALA l'enchaînement conversationnel est contraint par le résultat de l'évaluation de l'UMI courante. Le sens d'un énoncé initial est soumis à évaluation au travers de la réponse qui en est faite. On considère donc, comme dans le modèle de Trognon et Brassac, qu'un énoncé peut être sujet à négociation. Le sens en interaction n'est pas donné mais acquis. Évidemment, dans le cas du DHM finalisé, cette négociation est la plupart du temps inexistante car le cadre strictement applicatif des dialogues ne le permet pas. Il y a moins de jeu entre l'usage de la langue, c'est-à-dire *comment* on l'utilise, et son utilisation, c'est-à-dire *pourquoi* on l'utilise.

Afin de tester les potentialités du modèle de Lehuen, notre premier travail de thèse a consisté à reprendre le modèle COALA pour développer un premier système de dialogue (AMI1) en vue de l'expérimenter, d'obtenir un corpus et de l'analyser. Nous commençons par décrire l'expérimentation, le corpus obtenu et la méthode d'analyse. Nous donnons ensuite une synthèse des résultats obtenus qui montrent les limites du modèle COALA. L'analyse du corpus est donnée dans son intégralité en Annexe A.

2.4.2.1 Expérimentation de AMI1

Le système AMI1 (Agent de Messagerie Interactif) est un système de dialogue en langue naturelle écrite simulant un standardiste téléphonique. Il est la première version d'un projet informatique à long terme dont le développement fait l'objet du Chap. 4. Entièrement codé en Common Lisp, l'interaction avec le système s'effectue via une fenêtre de saisie sous MS-Windows (voir Sect. 4.1 p. 89 pour une présentation complète).

2.4.2.1.1 Protocole d'expérimentation

L'expérimentation s'est déroulée sur cinq demi-journées à raison d'une séance par demi-journée et par personne. Les séances se sont effectuées sur un terminal graphique. Celles-ci ont duré de trente à cinquante minutes. Les utilisateurs disposaient d'une notice présentant très brièvement le système (ses différentes fonctionnalités) ainsi qu'un annuaire téléphonique des personnes censées être connues du système. Ils avaient également la possibilité de noter sur une feuille leurs remarques sur le système et son fonctionnement. Les informations concernant chaque séance sont résumées Tab. 2.3.

Les utilisateurs étaient des personnes ayant une bonne ou une très bonne connaissance de l'informatique (quatre personnes sur cinq étaient des informaticiens), et ne connaissaient pas du tout ou un petit peu le principe du système (deux personnes sur cinq connaissaient un peu son principe de fonctionnement). Le choix délibéré de ce type de public a été concluant car, contrairement à des utilisateurs néophytes, ils se sont révélés assez peu coopératif, essayant de piéger ou de tester le système, ce qui nous a aidé à cerner les lacunes et faiblesses de cette

Séance	Date	Durée	Utilisateur	Dialogues
1	9 juin 1998	45 mn	Informaticien et ne connaît pas le système	D1-1 à D1-13
2	9 juin 1998	52 mn	Non-informaticien et ne connaît pas le système	D2-1 à D2-15
3	10 juin 1998	50 mn	Informaticien et ne connaît pas le système	D3-1 à D3-23
4	10 juin 1998	50 mn	Informaticien et connaît le système	D4-1 à D4-21
5	12 juin 1998	30 mn	Informaticien et connaît le système	D5-1 à D5-12

TAB. 2.3 – Informations sur les séances d'expérimentation de AMI1.

première version du projet AMI.

2.4.2.1.2 Corpus obtenu

Le corpus obtenu forme un ensemble de 84 dialogues [Lemeunier, 1998b]. Ces dialogues ont un nombre d'échanges allant d'un échange à 25 échanges. Sur les 84 dialogues, 61 sont *complets*, soit 72 %, et 23 sont *incomplets*, soit 28 %.

Un dialogue complet est un dialogue qui a été normalement clos soit sur l'initiative du système soit sur l'initiative de l'utilisateur. À l'opposé, un dialogue incomplet est un dialogue qui a été achevé brutalement sans clôture, soit à cause d'une erreur logicielle, soit pour mettre fin à un dialogue « tournant en rond » ou trop long.

Un dialogue complet ne signifie pas que le dialogue s'est bien déroulé et que la demande de l'utilisateur a bien été comprise. Un dialogue complet peut avoir été clos par l'utilisateur soit par lassitude (due à la longueur du dialogue ou sa répétitivité), soit par l'incapacité du système à répondre à ses requêtes. Le nombre de dialogues ayant réellement bien fonctionnés, c'est-à-dire dans lesquels la requête de l'utilisateur a été correctement interprétée, est de 29 soit 34,5 %. Ainsi, nous obtenons trois types de dialogue : dialogue complet et correct, dialogue complet et incorrect, dialogue incomplet (voir Tab. 2.4). Ces résultats montrent les faibles capacités du système AMI1 puisque seulement environ un tiers des dialogues sont corrects.

Dialogues complets et corrects	34,5 %
Dialogues complets et incorrects	37,5 %
Dialogues incomplets	28 %

TAB. 2.4 – Résultats de l'expérimentation de AMI1.

2.4.2.1.3 Méthode d'analyse

Le corpus n'a pas été analysé entièrement. Seules les deux premières séances ont été analysées en détail, les autres dialogues n'ont fait l'objet que d'une analyse superficielle. La raison de cette limitation de l'étude est que les phénomènes observés dans les dernières séances sont

identiques à ceux observés dans les deux premières. Celles-ci nous ont donc semblé suffisamment représentatives de la globalité du corpus obtenu.

La méthode d'analyse est la suivante : pour chaque dialogue sont pris en compte aussi bien la structure du dialogue que les traces du fonctionnement du système correspondant à ce dialogue. La structure est représentée en UMI selon le modèle COALA. En plus de la suite d'énoncés formant le dialogue à proprement parler, la représentation de sa structure permet de rendre compte du fonctionnement global de l'interaction : s'agit-il d'un dialogue linéaire sans incidence ou bien d'un dialogue fortement incident, etc. Les traces permettent de suivre et d'apprécier la compréhension des énoncés de l'utilisateur par le système. L'étude des traces du fonctionnement est essentiel pour notre propos : elle permet de voir la manière dont le système traite les énoncés de l'utilisateur. L'analyse se fait donc à la fois sous un aspect externe et sous un aspect interne.

2.4.2.2 Limites du modèle COALA

L'analyse du corpus de AMI1 a tout d'abord révélé la présence de trois types de problèmes : des problèmes purement informatiques et des problèmes de conception qui pourraient être résolus si le logiciel était durable, et enfin des problèmes conceptuels liés au dialogue et à la gestion de l'interaction (voir l'Annexe A pour une présentation complète).

Parmi les problèmes conceptuels, qui nous intéressent ici, nous avons relevé deux problèmes principaux. Le premier est celui de la prise en compte des *non-attendus*. Le second est la lourdeur de l'interaction dans les cas des incompréhensions (totales et partielles) ou dans le cas des non-attendus.

2.4.2.2.1 Les *non-attendus*

Avec le modèle COALA, les attentes sont fortement liées à l'activité applicative du système, c'est-à-dire à la tâche qu'il est capable d'accomplir : « les attentes de la machine peuvent être exprimées en termes d'objets du domaine d'application et d'actes de dialogue orientés par la tâche. » [Lehuen, 1997b, p. 108]. Elles sont principalement créées par l'expertise de l'application et contraignent fortement le déroulement du dialogue comme l'illustre l'exemple de la Fig. 2.11 tiré de l'expérimentation de AMI1. Dans ce dialogue, la partie « [...] je voudrais parler à Pascal G » de l'énoncé H2 n'est pas prise en compte par le système car il ne s'y attend pas à ce moment-là du dialogue. La solution qui consiste à ajouter des attentes dans le modèle de la tâche applicative n'est pas satisfaisante puisqu'elle n'est pas généralisable : d'une part, les attentes sont liées à l'application et donc non-portables à une autre application et, d'autre part, ajouter des attentes et un processus dont on ne connaît pas la fin.

2.4.2.2.2 Les lourdeurs de l'interaction

Lehuen indique lui-même que, durant les phases d'apprentissages, « à l'interlocuteur d'être

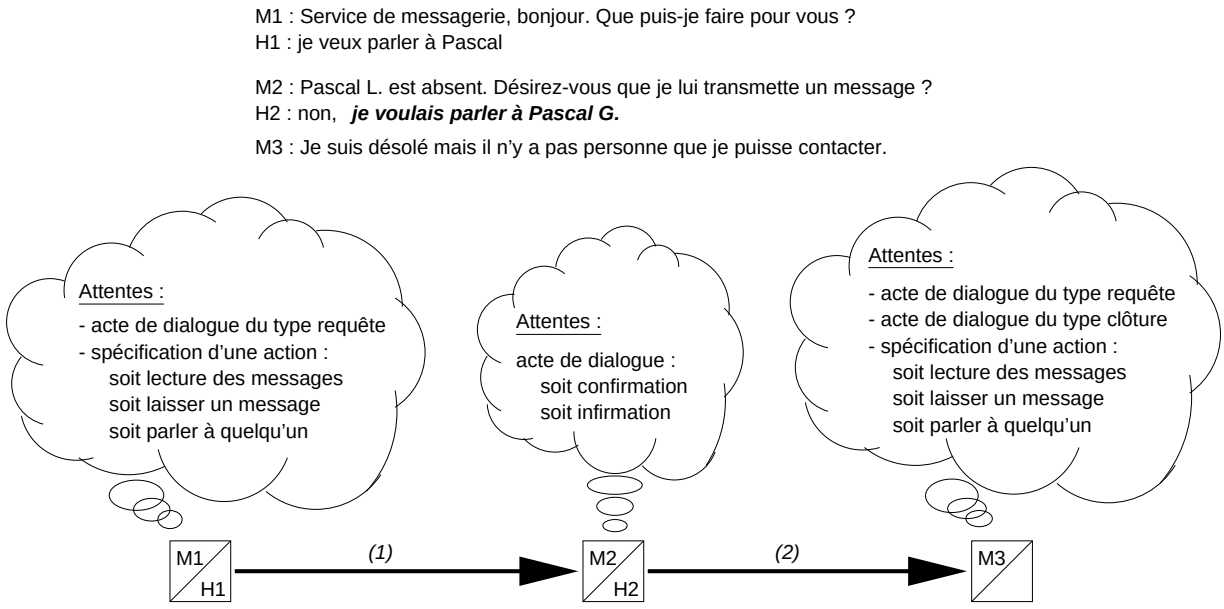


FIG. 2.11 – Exemple du problème des non-attendus dans AMI1.

patient et coopératif. » [Lehuen, 1997b, p. 139]. Cependant, même lorsque le système possède beaucoup de connaissances comme dans le cas de AMI1, l'exemple du dialogue Fig. 2.12 montre bien que ce type de situation est difficilement acceptable.

Dans cet exemple le système connaît parfaitement le sens de « bonjour » en H1, mais le problème provient du fait qu'il s'attend surtout à d'autres types d'actes de discours, notamment d'une requête[14]. La capacité à poser des questions sur des hypothèses de compréhension, sans être mise en question, doit donc être utilisée plus finement pour éviter ce genre de lourdeur qui peut également apparaître en cas d'incompréhension.

2.4.2.2.3 Une ébauche de solution

Dans [Lemeunier, 1999] nous avons étendu et généralisé plus avant la notion d'attente en améliorant le formalisme de représentation.

Premièrement, nous avons proposé de structurer les attentes hiérarchiquement par fréquence d'apparitions[15]. Ceci limite le premier défaut dont il est question ci-dessus. Sans enlever la capacité à poser des hypothèses, nous faisons simplement en sorte qu'il y en ait moins de posées (cf. Fig. 2.13).

Les attentes les plus fréquentes sont en haut à gauche. Ainsi, certaines attentes sont des sous-attentes d'attentes plus importantes. Par exemple, l'attente $< PARLER >$ est à gauche de l'attente $< LIRE - MESSAGE >$ car nous avons observé qu'il est plus probable que l'utilisateur utilise AMI1 pour entrer en contact avec une tierce personne que pour lire ses

[14] L'autre problème est évidemment que l'usager est ici peu coopératif. Ce type de dialogue existe cependant même si l'usager fait montre d'une grande patience (voir le corpus obtenu [Lemeunier, 1998b]).

[15] L'existence d'un corpus de l'application est nécessaire pour déterminer ces fréquences d'apparition.

M1 : Service de messagerie, bonjour. Que puis-je faire pour vous ?
H1 : bonjour.
M2 : Venez-vous de me demander de lire les messages qui vous sont adressés ?
H2 : non.
M3 : Venez-vous de me demander de laisser un message pour quelqu'un ?
H3 : non.
M4 : Venez-vous de me deamnder de parler à une personne du laboratoire ?
H4 : non.
M5 : Venez-vous de vous présenter ?
H5 : non.
M6 : Venez-vous de clore le dialogue ?
H6 : non.
M7 : Venez-vous de désigner la personne que vous voulez contacter ?
H7 : non.
M8 : Pourrais-je connaître l'identité de votre correspondant ?
H8 : oui.
M9 : Pourrais-je connaitre l'identité de votre correspondant ?
H9 : Zinedine Zidane
...

FIG. 2.12 – Exemple d'un dialogue lourd avec AMI1.

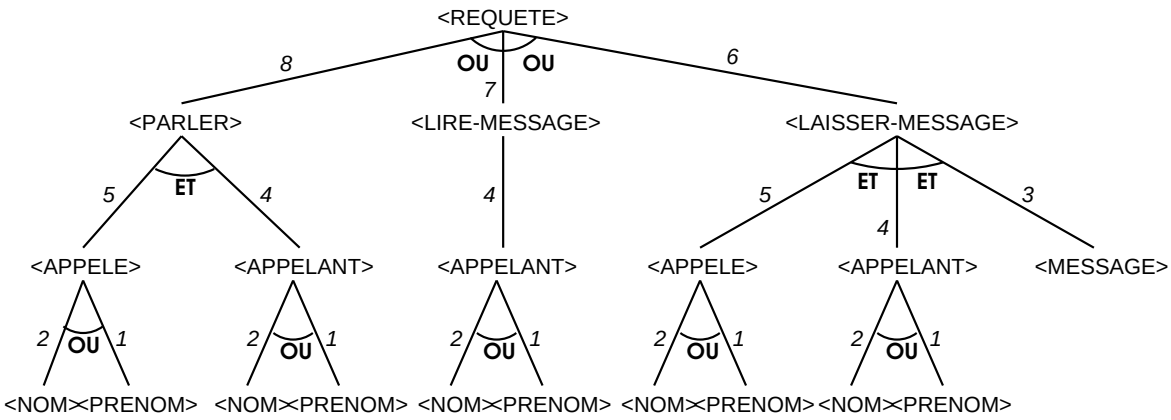


FIG. 2.13 – La hiérarchie d'attentes applicatives de AMI1.

messages. En cas d'incompréhension, les questions portent d'abord sur les attentes les plus importantes des hiérarchies et le dialogue ne se poursuit que si l'énoncé de l'utilisateur correspond bien à ces attentes. Le dialogue précédent est maintenant celui de la Fig. 2.14. Il est rapidement clos car l'attente d'un acte de dialogue du type requête, qui est la plus importante, n'est pas satisfaite.

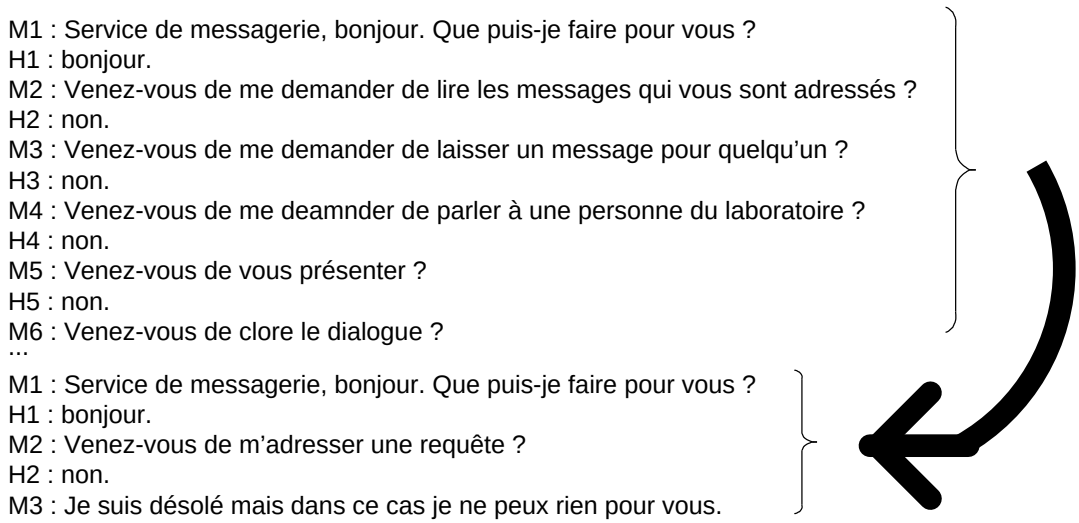


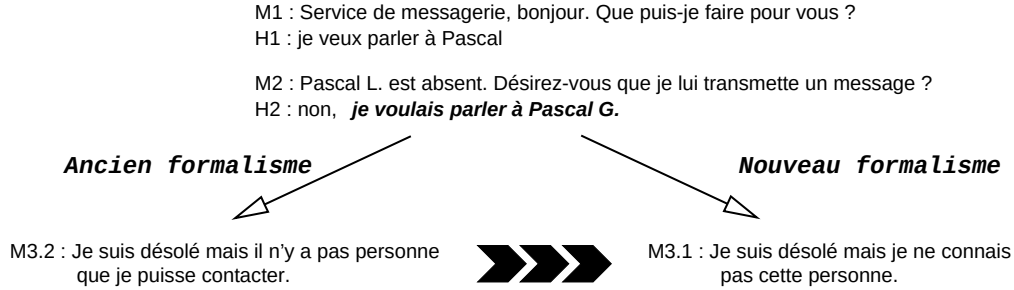
FIG. 2.14 – Amélioration de la lourdeur du dialogue.

Deuxièmement, grâce à ce formalisme, le système a la possibilité de créer automatiquement des attentes non-prévues lors de l'interprétation des énoncés de l'utilisateur et ceci à n'importe quel moment de la progression du dialogue. Dans l'exemple Fig. 2.15, le système ne s'attend pas à ce que l'utilisateur fasse une objection en H2. Avec le nouveau formalise de représentation, celle-ci sera quand même prise en compte car le système va créer les attentes présentes dans l'analyse de l'énoncé mais non-présentes dans l'UMI courante. Cela n'était pas possible dans le modèle initial de Lehuen puisqu'on ne savait pas comment lier entre elles les attentes créées ou bien même comment les lier avec celles déjà existantes.

Pour autant, il est évident que cela ne permet pas au système de réagir aux énoncés qu'il ne peut pas analyser : pour cela, il faut utiliser des stratégies dialogiques (ou *tactiques dialogiques* dans la terminologie de Lehuen) menant le système à vérifier des hypothèses qu'il pose sur le sens des termes ou des expressions inconnus.

Ce premier développement n'est cependant pas suffisant car deux problèmes subsistent. Le premier est lié à la stricte structuration en UMI : le dialogue est géré par à-coups, sans prendre en compte le dialogue antérieur. Autrement dit, il n'y a pas de *mémoire de l'interaction.* Pourtant, un dialogue se déroule en fonction de l'instant présent et également en fonction des instants passés. Aussi, rien n'empêche le système de se répéter plusieurs fois au cours d'un même dialogue.

Le second problème est que les intentions communicatives sont directement générées par



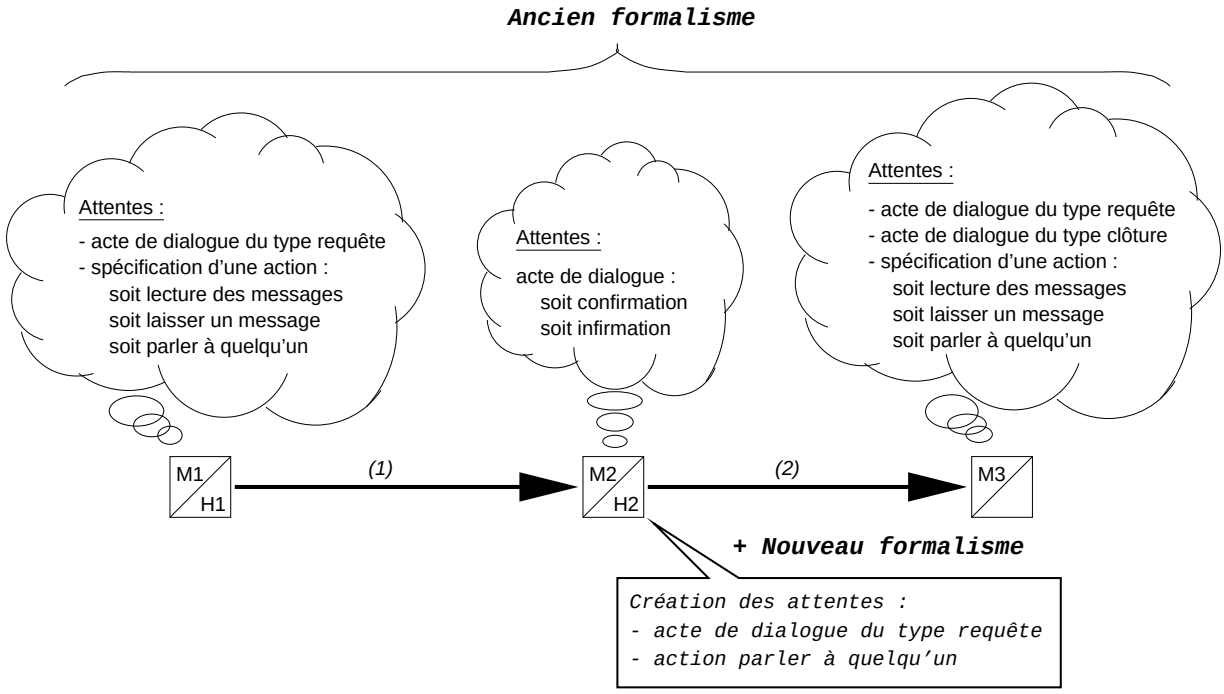


FIG. 2.15 – Prise en compte des non-attendus.

l'activité applicative alors qu'au contraire, dans le modèle des interactions langagières que nous avons adopté, les intentions ne sont pas liées à la tâche mais aux énonciations à travers lesquelles l'application est menée.

La prise en compte du terrain commun que nous proposons dans notre modèle apporte une réponse à ces deux problèmes. Ce modèle fait l'objet du prochain chapitre.

Chapitre 3

La génération des intentions de communication

Le chapitre précédent nous a permis de mettre en lumière les éléments fondamentaux à considérer pour construire des machines avec lesquelles l'utilisateur humain pourrait dialoguer. Nous allons dans ce chapitre en synthétiser tout d'abord les éléments essentiels sous la forme de propositions de modélisation des dialogues homme-machine. Ces propositions portent sur la compréhension en interaction, sur la mémoire interactionnelle et sur les intentions de communication.

Ces propositions forment la base théorique du modèle Génédic, un modèle de génération des intentions de communication pour les machines. Le modèle Génédic prend en compte le modèle des interactions langagières selon lequel le sens que les locuteurs ont l'intention de communiquer est l'objet de négociations. Le modèle Génédic peut contribuer à améliorer la compétence conversationnelle des systèmes de dialogue en les rendant plus apte à suivre une conversation réelle.

3.1 Propositions pour modéliser le dialogue homme-machine

3.1.1 La compréhension en interaction

Après nous être distingué des modèles fondés sur la planification, nous avons présenté des travaux qui prennent mieux en compte les processus d'interactions langagières. L'essentiel de cette approche réside dans la manière de voir comment deux « individus » — une machine et un homme — parviennent à communiquer. Résumons les principaux modèles existants.

Dans les modèles de Searle & Vanderveken et de Sperber & Wilson, les énoncés sont interprétés en fonction de leurs valeurs sémantiques et du contexte d'interaction. L'interprétation qui en résulte n'est jamais remise en cause, puisque celle-ci est censée pouvoir être inférée à tous les coups. En revanche, dans les modèles proposés par Trognon & Brassac (cf Sec. 2.2) et

par Nicolle & Saint-Dizier De Almeida (cf Sec. 2.3.2), l'interprétation de l'énoncé du locuteur n'est plus un processus individuel, mais un processus commun au locuteur et à l'interlocuteur. Une valeur est associée à l'énoncé et son statut n'est plus posé en termes de fixation interprétative mais en termes de *fixation interprétative partagée*, car la valeur de l'énoncé est l'objet d'une négociation. Pour résumer les deux points de vue, nous dirons que dans la première approche l'interlocuteur comprend le locuteur et vice-versa, tandis que dans la seconde approche le locuteur et l'interlocuteur *se* comprennent (et un observateur extérieur ne comprend pas forcément exactement ce qu'ils se sont dit s'ils font des références à des objets inconnus de l'observateur). Nous suivons la seconde approche pour les deux raisons suivantes.

Premièrement, l'observation des conversations humaines nous pousse à penser que cette approche leur correspond le mieux. L'exemple de dialogue entre les deux ouvriers du chapitre précédent (cf. Fig. 2.3 p. 24), montre bien tout le jeu subtil « joué » par les locuteurs. Ce que ces ouvriers peuvent se dire, ils pourraient le résumer en un seul énoncé chacun. Mais cela ne se passe pas comme cela ! Ils ont besoin de plusieurs échanges pour ce qu'ils veulent dire, parce qu'ils doivent co-construire un espace de référenciation pour parler des mêmes choses afin de se comprendre. La logique que suivent les locuteurs ne s'apparente pas à une démonstration mathématique dans le sens où les énoncés ne suivent pas un enchaînement **calculable *a priori***, car ce que dit l'interlocuteur détermine comment le locuteur répondra. Le dialogue est une fonction *indéterminée*. Il ne s'agit donc pas d'une logique illocutoire, **individuelle**, mais d'une logique interlocutoire **participative**. Les phénomènes langagiers qui en découlent, dont ce dialogue est un exemple parmi d'autres, en est la trace observable.

Deuxièmement, pour construire de meilleurs systèmes de dialogue, c'est-à-dire des systèmes qui seront *in fine* préférés par les utilisateurs, nous devons parvenir à des dialogues plus **véritables**, c'est-à-dire dans lesquels le système est un partenaire avec lequel l'utilisateur interagit plus réellement. Il ne fait aucun doute que les dialogues finalisés actuels « *semblent* » fonctionner : le système est capable de fournir les informations demandées par l'utilisateur. Mais ce bon fonctionnement n'est qu'apparent. Ce qui sépare l'agent humain de l'agent artificiel c'est que le premier est muni d'une formidable capacité d'interprétation au contraire du second. L'utilisateur humain, de par sa nature, cherche toujours à donner une interprétation à ce qu'il lui est dit. Il est capable de suppléer aux lacunes dialogiques du système en imaginant une intercompréhension même si elle n'existe pas. Pourquoi chercher alors à en faire plus ? Parce que ce déséquilibre apparaît tôt ou tard lorsque l'utilisateur « en demande un peu plus » à la machine. Notre souhait (à longs termes) est de parvenir à un équilibre entre la capacité interprétative de l'homme et celle de la machine. Comme le rappelle Vilnat : « il semble très dangereux de refuser de s'intéresser à un phénomène, pour la seule raison qu'il n'apparaît pas dans les corpus. Il est en particulier impossible de ne donner à un utilisateur que la possibilité de se servir d'une "partie de la langue" » [Vilnat, 1997, p. 136]. Cela s'applique non seulement aux ellipses, anaphores et autres tropes, mais également aux comportements discursifs

eux-mêmes.

En conséquence, nous partirons de la proposition suivante pour modéliser la compréhension en interaction dans le modèle Génédic :

> Proposition 1
> La compréhension des interactants est une intercompréhension résultant d'un processus partagé de constitution mutuelle d'un espace de référenciation.

Le modèle des interactions langagières de Nicolle et Saint-Dizier De Almeida, que nous reprenons à notre compte dans cette thèse, s'appuie sur une logique interlocutoire dans laquelle sont redéfinies les notions de succès et de satisfaction de la théorie des actes de langage vue par Vanderveken. Avec ces définitions (cf. Sec. 2.3.2.3.2 p. 34), nous nous sommes concentré sur l'explicitation de l'élaboration du terrain commun qui est le processus cognitif sous-jacent à l'intercompréhension. D'une part, la constitution du terrain commun n'est pas expliquée en profondeur dans le modèle de Nicolle, et, d'autre part, il s'agit de la condition essentielle (mais non-suffisante) à l'établissement d'un authentique dialogue.

3.1.2 La mémoire interactionnelle

D'un point de vue strictement logique, un espace de partage est nécessaire dès lors que l'on considère le sens en interaction comme le résultat d'un travail commun de constitution. La négociation du sens, qu'elle soit implicite ou explicite, s'appuie en effet obligatoirement sur l'existence d'un **espace commun** de référenciation. Pour négocier quelque chose, il faut d'abord représenter cette chose : on part pour cela de sa réification[1]. Ce faisant, l'agent prend mécaniquement un recul par rapport à la chose représentée. Il faut également être en accord sur ce que l'on va négocier. Un terrain commun est donc constitué.

La notion de terrain commun a été étudiée en particulier par Clark [Clark, 1992]. Il s'agit « d'un ensemble de connaissances, hypothèses et croyances partagées par les partenaires, et présumées telles par chacun. » [Caron, 1997]. Chaque interactant se fait une image propre du terrain commun construit en cours du dialogue, qui fait l'objet d'un processus permanent d'ajustement. Ce terrain commun permet d'éviter ou de lever les ambiguïtés rencontrées. Il est une condition nécessaire au fonctionnement du dialogue et il dépend de ce dialogue pour être construit [Vivier, 1997]. Ainsi, l'intercompréhension et la construction du terrain commun se conditionnent réciproquement. Clark distingue différentes connaissances susceptibles d'être partagées dans le terrain commun :

[1] Pour simplifier, nous dirons que la réification est l'opération élémentaire de catégorisation par laquelle on catégorise par apprentissage (par différenciation avec les autres objets) un objet quelconque physique ou mental. Les autres opérations sont, d'une part, l'abstraction et l'instanciation, et d'autre part, la généralisation et la spécialisation.

- les connaissances liées à la co-présence physique de la situation de communication ;
- les connaissances liées à la co-présence linguistique : connaissances propres à la mémoire de ce qui a été dit ;
- les connaissances communes aux membres d'une même communauté.

Cette recherche porte sur le second point : nous ne prenons en compte ni la dimension physique ni la dimension sociale des dialogues homme-machine, le type de l'application choisie le permet.

Le lieu de la représentation des éléments négociés est une *mémoire de travail*. Lors d'un dialogue, la mémoire de travail d'un agent (artificiel ou naturel) est avant tout une *mémoire interactionnelle*. Cette dernière peut être vue comme une spécialisation temporaire de la mémoire à court terme [Tiberghien, 1997]. La définition de la mémoire interactionnelle ne fait pas directement référence au concept d'histoire interactionnelle [Kerbrat-Orecchioni & Plantin, 1995]. Dans cette dernière notion intervient tout ce que les locuteurs ont partagé au cours des interactions précédentes, et, en cela, elle fait notamment référence aux aspects sociaux des interactions humaines. Ici, pour l'instant, la mémoire interactionnelle correspond aux aspects purement cognitifs de l'interaction courante, sans remémoration des dialogues antérieurs éventuels, et sans prendre en considération les aspects sociaux dont rend par exemple compte la théorie des faces de Goffman [Goffman, 1974]. D'autres travaux, surtout dans le domaine des systèmes multi-agents, s'intéressent à ces aspects [Chicoisne & Pesty, 1999].

Cette mémoire interactionnelle fixe les éléments provenant de l'interprétation des énoncés du locuteur et de l'interlocuteur[2], et des inférences faites à partir de ces premiers éléments. Ils forment le résultat de l'interprétation des énoncés en termes d'actes de langage. Nous retrouverons ainsi des forces illocutoires appliquées à des contenus propositionnels, mais également des concepts instanciés, des références résolues, etc. Ces éléments mémoriels ne sont pas indépendants les uns des autres mais groupés selon des structures d'action.

En conséquence, nous partirons de la proposition suivante pour modéliser le terrain commun dans le modèle Génédic :

> Proposition 2
> Une représentation du terrain commun est co-construite dans la mémoire interactionnelle de chacun des participants à un dialogue.

[2] Si l'on retient l'idée que la parole n'est pas préméditée, un modèle complet devrait alors tenir compte du fait que le locuteur interprète son propre énoncé [Coursil, 1992]. Ce point de vue est entièrement justifié pour les dialogues oraux où l'analyse des transcriptions fait apparaître de nombreux phénomènes de recouvrement, des hésitations, des reprises, des corrections, etc. La définition que donne Luzzati d'un énoncé oral est d'ailleurs celle d'un « énoncé conçu et perçu dans le fil de son énonciation ». Pour ce qui concerne le dialogue écrit, cela est moins justifié, puisque les locuteurs sont moins spontanés, bien qu'ils produisent souvent, comme à l'oral, des énoncés agrammaticaux. Le fait d'écrire un énoncé implique que celui qui écrit interprète en même temps ce qu'il écrit, ne serait-ce que parce que le passage du code oral au code écrit l'impose. Pour prendre en compte la non-préméditation de la parole, il faut un système capable de faire une analyse en temps réel, comme le suggère Gérard [Gérard, 1999].

Le modèle de Nicolle distingue la représentation de l'autre, la représentation de soi-même et la représentation du terrain commun. Nous n'avons pas encore cherché à différencier clairement les éléments de l'image de l'autre des éléments de l'image de soi. Nous avons surtout travaillé sur les éléments de l'image du terrain commun que se fait chaque agent. À ce propos, nous pensons que les différentes images construites par chacun des acteurs d'un dialogue ne doivent pas être considérées comme étant isolées les unes des autres. Notre hypothèse est au contraire que les éléments de ces images sont liés « pragmatiquement », et qu'ils sont répartis à travers différents groupes mémoriels. Par exemple, le système peut avoir des croyances sur les connaissances de l'utilisateur — faisant donc partie de l'image de l'autre — et que ces croyances soient liées à des connaissances qu'il pense être partagées (éléments du terrain commun).

Dans la plupart des systèmes de dialogue homme-machine, il est nécessaire de maintenir un historique du dialogue. Par exemples, dans le système Diabolo de Vilnat et Grau [Vilnat, 1997], et dans le modèle ADex de Charnay [Charnay, 1999], l'historique du dialogue stocke, dans une représentation structurée adaptée du modèle genevois[3], des pages de l'historique. Ces dernières sont composées d'une suite d'informations concernant chaque énoncé : identité du locuteur, l'énoncé, l'interprétation de l'énoncé, le thème principal, etc. Dans notre approche, la notion d'historique du dialogue disparaît au profit de la notion de mémoire du dialogue courant. D'autres problèmes apparaissent alors, qui portent sur l'oubli, la remémoration des éléments en mémoire de travail, et plus généralement sur leur cycle d'activation. Généralement, les éléments mémoriels suivent un cycle précis : après leurs activations, ils subissent certains traitements (modification de leurs états, structuration), puis doivent disparaître d'eux-mêmes ou laisser la place, être « écrasés » par d'autres éléments, ou tout simplement être supprimés. Le problème concernant l'oubli est délicat à traiter. La grande capacité de mémorisation à court terme de la machine par rapport aux capacités humaines peut poser des problèmes : un utilisateur ne se rappelle pas forcément tout ce qu'il a dit depuis le début de l'interaction, surtout si le dialogue dure depuis plusieurs minutes, contrairement au système. Ce dernier peut faire référence à un objet évoqué au début du dialogue mais l'utilisateur ne sera plus capable de résoudre cette référence car le terrain commun ne sera plus le même[4]. Ce problème complexe n'a été traité que très partiellement dans ce travail (cf. Sect. 3.2).

3.1.3 Les intentions de communications

La manière dont est perçu un système de dialogue repose sur les énoncés qu'il produit. Ses énoncés reflètent son interprétation de la situation : ils reflètent sa compréhension des énoncés

[3] E. Roulet, A. Auchlin, J. Mœschler, C. Rubattel et M. Schelling. *L'articulation du discours en français contemporain*. Peter Lang, Berne, 1985.

[4] Pitrat affirme que les machines seront à l'avenir plus intelligentes que les humains [Pitrat, 1995]. S'il ne fait aucun doute qu'elles ont déjà une bien plus grande capacité de raisonnement logique, le problème sera celui de leurs adaptations cognitives aux humains, ce qui n'est pas un problème de crédibilité du fonctionnement cognitif.

des utilisateurs et une partie de ses raisonnements. Tout informaticien du dialogue homme-machine se pose la question suivante : qu'est-ce que le système doit dire à l'instant considéré, compte tenu de ce qui s'est déjà passé et de ce qui risque de se passer ? D'après la théorie des actes de langage ce que dit un agent dialoguant se présente sous la forme d'une force illocutoire appliquée à un contenu propositionnel. L'acte de langage permet « d'encapsuler » dans un énoncé ce que l'agent dialoguant à l'intention de communiquer.

L'intentionnalité est difficile à définir. Dans la phénoménologie[5], l'intentionnalité est ce qui caractérise la conscience : la conscience est toujours *conscience de quelque chose*, ce quelque chose étant nommé un contenu intentionnel. Husserl et Heiddeger ont proposé deux manières de rendre compte de l'intentionnalité [Dreyfus, 1993]. Heidegger s'est attaché à décrire un *sujet agissant*, tandis que Husserl, un *sujet connaissant*, reprenant en cela le projet cartésien d'une « science universelle » indubitable. Pour Searle, héritier de Husserl, toute action est intentionnelle, c'est-à-dire que, même lorsque le but n'a pas été posé antérieurement, comme lorsqu'un sujet se lève brusquement de sa chaise, il doit avoir à l'esprit ce qu'il est en train de faire. Autrement dit, chez Searle, il existe toujours une liaison causale entre une *intention dans l'action* et l'action [Searle, 1985]. Les actions préméditées possèdent en plus une *intention préalable*, au contraire des actions spontanées qui ne nécessitent qu'une intention dans l'action[6].

Pour Heidegger, à l'opposé de Searle, les actions quotidiennes sont guidées par la situation et ne requièrent absolument pas de représentation mentale du but. Il différencie l'action délibérée de l'action quotidienne automatique, c'est-à-dire sans intentionnalité causale. L'action délibérée à la Husserl n'apparaît que lorsque la situation devient difficile à appréhender. L'intentionnalité dont il est question chez Heidegger n'implique donc pas obligatoirement un contenu intentionnel. L'influence indirecte de Heidegger sur les informaticiens a été aussi forte que celle de Searle, que ce soit en intelligence artificielle, en interface homme-machine, ou en méthodologie de développement [Winograd, 1993]. Nous n'avons pas directement cherché à débattre de ce problème philosophique, mais il apparaîtra clairement au lecteur, à travers la présentation de notre approche qui va suivre (cf. Sect 3.4), que notre vision de l'intentionnalité est plutôt celle de Heidegger alors que celles des modèles fondés sur la planification serait plutôt celle de Husserl.

La nécessité de séparer actions communicatives et actions non-communicatives, autrement dit de séparer les aspects non-applicatifs et les aspects applicatifs d'un modèle du dialogue finalisé, préoccupation que l'on retrouve dans les travaux de Vilnat et Grau [Vilnat, 1997], a guidé le développement du modèle Génédic. Dans les modèles fondés sur la planification, les actes de langage sont des actions à part entière. Elles comprennent notamment des préconditions d'intentionnalité. Pour chaque action de communication l'agent doit avoir l'intention d'effectuer cette action [Guyomard *et al.*, 1995]. Par exemple, pour poser une question, l'agent

[5] La phénoménologie est à la fois une théorie de la connaissance et une méthode philosophique. Il s'agit de décrire de façon rigoureuse les phénomènes qui se présentent à la conscience intentionnelle.

[6] Searle ne considère pas les actes réflexes comme ronfler, respirer, éternuer, etc., comme de vraies actions.

doit d'abord avoir l'intention de poser cette question. Cette intention est soit inférée à partir des actions du plan suivi par l'utilisateur et reconnu par le système (il s'agit alors d'une intention attribuée à l'utilisateur), soit inférée à partir du plan du système lui-même (il s'agit alors d'une intention du système).

Il est évident que l'intention de communication se situe avant la production de l'acte communiquant cette intention. Les intentions communicatives dont il est question dans les modèles fondés sur la planification, sont des **intentions figées** : d'une part, elles sont liées statiquement à la forme qu'elles prendront en actes, d'autre part, elles sont également liées statiquement à la tâche applicative. La limite de ces modèles est qu'ils ne séparent pas l'intention elle-même de la forme qu'elle prendra en actes. Or, même si j'ai l'intention de poser une question parce qu'il me faut une information, l'énoncé produit pourra prendre la forme d'une requête « *Quand est-ce que le train...* » ou la forme d'un ordre « *Dites-moi quand le train...* » Autrement dit, l'intention communicative ne présage aucunement de la manière dont elle sera communiquée en actes, car d'autres aspects interviennent dans l'énonciation de l'intention (aspects sociaux notamment, mais aussi interactionnels).

Ainsi, les modèles initiaux de la planification ne proposent pas une séparation nette entre l'activité conversationnelle et l'activité applicative d'un système de dialogue, même s'ils permettent par ailleurs de modéliser efficacement les raisonnements portant sur l'application. Dans un système de dialogue homme-machine, l'activité conversationnelle est portée par l'intentionnalité communicative (ce que j'ai l'intention de dire) de chacun des agents, au contraire de l'activité applicative qui est portée par l'intentionnalité non-communicative (ce que j'ai l'intention de faire, qui ne fait pas intervenir d'actions langagières). Par exemple, faire un voyage implique l'établissement d'un plan d'actions physiques pour que je puisse faire ce voyage. Si l'obtention du billet de transport nécessite une interaction verbale auprès d'un agent de transport, l'intention communicative correspondant à ma requête initiale ne sera pas planifiée mais créée uniquement par la nécessité de communiquer verbalement ma demande à l'agent contacté.

Certes, si l'on veut inscrire les énonciations dans le cadre général d'une théorie de l'action, on peut vouloir représenter toutes les actions (verbales ou non) dans un même formalisme. Il faut cependant bien distinguer les différents types d'actions : des actions automatiques non planifiées (par exemple bouger son bras), des actions planifiées consciemment (par exemple ceux de mon fameux voyage), et des actions conscientes non-planifiées (par exemple demander que l'on ferme la fenêtre). Les actions langagières, résultats d'intentions communicatives, sont du dernier type ; de plus, les actions du second type et les actions du dernier type ne peuvent pas émaner des mêmes processus, puisque les unes sont planifiées et les autres non. Les approches fondées uniquement sur la planification des actions ne permettent pas de différencier ces deux types d'actions car ces approches partent au contraire de l'idée de l'uniformisation des actions. Les intentions communicatives n'y sont pas **dynamiquement** créées, ce qui a

deux conséquences :

- La première est un manque de dynamicité potentiel du système de dialogue lorsque l'on sort du cadre strict du dialogue prévu à la conception.
- La seconde conséquence est que l'approche par plans ne peut décidément pas convenir au modèle des interactions langagières que nous avons adopté qui est celui de Nicolle [Nicolle & Saint-Dizier De Almeida, 1999a]. Dans ce modèle, ce qu'un agent dit ne peut pas être planifié puisque l'intention de communication sous-jacente à son énonciation émane de l'enchaînement interlocutoire. Cette vision des actions langagières « effectuées » pour dialoguer est clairement heideggerienne.

Dans leur théorie de la pertinence [Sperber & Wilson, 1989], Sperber & Wilson définissent la communication ostensive-inférentielle comme mettant en jeu deux intentions qu'ils identifient de la façon suivante : d'une part, une *intention informative* d'informer les destinataires de quelque chose, et, d'autre part, une *intention communicative* d'informer les destinataires de cette intention informative. L'intention informative est d'informer que le locuteur attend une réaction du destinataire et que cette réaction attendue est pour le locuteur le moyen de réaliser son but. L'intention communicative est une intention informative de second ordre. Au contraire de Sperber & Wilson, nous pensons qu'il n'est pas nécessaire de faire cette distinction. À la suite de Bange nous considérons à l'opposé que « " l'intention informative " se réalise par " l'intention communicative " [...] c'est-à-dire que, dans une perspective actionnelle, elle a une place subordonnée par rapport au but visé. » [Bange, 1992, p. 200]. Nous sommes donc reparti de la définition de Grice de la signification en situation (ou signification non-naturelle) : un locuteur L par l'énonciation de x à l'intention de produire un effet r sur un récepteur R grâce à la reconnaissance de cette intention [Grice, 1968]. C'est cette intention unique que nous désignons sous le terme d'intention communicative.

Pour les agents artificiels, les intentions communicatives sont générées dynamiquement en suivant des **lois** interactionnelles et comportementales. Celles-ci prennent la forme de **règles** de génération appliquées aux éléments mémoriels (cf Sect. 3.4).

> Proposition 3
> Les intentions de communication d'un système de dialogue sont générées dynamiquement à partir des éléments de la mémoire de travail et de lois interactionnelles et comportementales.

3.1.4 Le modèle Génédic

Les trois propositions précédentes forment la base théorique du modèle **Génédic** (**Géné**ration **d**ynamique des **i**ntentions de **c**ommunications). Nous allons dans la suite de ce chapitre tout d'abord préciser la mémoire de l'interaction d'un système de dialogue homme-

machine, en définissant les unités minimales de mémoire et la manière dont elles sont structurées. Nous verrons comment fonctionne cette mémoire, et en particulier comment s'y intègre la représentation du terrain commun. Nous montrerons ensuite comment sont « automatiquement » générées les intentions de communication. Finalement, nous donnerons un exemple d'application du modèle Génédic.

3.2 Définition de la mémoire de travail

Bassi Acuña [Bassi Acuña, 1995] définit un modèle de la mémoire humaine pour la compréhension de texte. Au contraire, nous définissons un modèle de la mémoire de travail pour un système de dialogue homme-machine en visant celle de tout agent artificiel.

La mémoire de travail est un processus permettant la coopération des activités du système en supportant le partage des représentations maintenues en mémoire. Elle sert à la fois de support et de mode d'accès aux représentations construites durant le dialogue chez l'agent artificiel [Lemeunier & Lehuen, 1999].

De manière similaire à la mémoire naturelle, la mémoire de travail d'un système de dialogue maintient actifs des éléments de la mémoire à long terme [Tiberghien, 1997]. Cependant, l'activité de la mémoire ne consiste pas uniquement à assurer le maintien mnésique des représentations construites pour et dans le dialogue, mais consiste également à assurer leur *effacement*[7] de la mémoire. Dans cette première version du modèle Génédic ce second aspect n'a pas été entièrement traité. Les éléments mémoriels perdurent durant tout le dialogue ou sont « effacés » par les autres activités. À la fin d'un dialogue, la mémoire est « vidée » pour préparer le système au dialogue suivant. Par ailleurs, la notion d'histoire interactionnelle n'est pas prise en compte ; aucune connaissance sur les utilisateurs n'est mémorisée. Cela signifie aussi qu'aucune connaissance n'est remémorée lorsqu'une interaction s'effectue à nouveau avec un même utilisateur. Dans [Lemeunier, 1998a] nous avons proposé un système doté d'une telle capacité. La solution proposée n'est cependant pas satisfaisante car elle est, d'une part, monologique, et d'autre part dépendante de l'application (elle s'appuie sur l'identification explicite des utilisateurs).

La mémoire du modèle Génédic est un ensemble de **structures arborescentes** de composants élémentaires notés **UMM**[8] (**U**nités **M**inimales de **M**émoire). Ces composants élémentaires sont activés en mémoire de travail selon le déroulement du dialogue par les différentes activités du système. On aura donc des UMM activées par l'activité applicative, par exemple, comme résultat d'une recherche dans une base de connaissances, et d'autres activées par l'activité langagière, par exemple comme résultat d'une analyse linguistique qui bute sur un terme

[7] Cet effacement se fait d'ailleurs à deux niveaux : au niveau symbolique d'abord, en désactivant tel ou tel élément de la mémoire de travail ; au niveau physique ensuite, puisque l'espace mémoire réservé à cet élément symbolique n'est plus accessible ou n'est plus associé au même élément.

[8] Ce nom a été également choisi en référence aux UMI définis dans le modèle COALA de Lehuen (cf. Sect. 2.4).

inconnu.

3.2.1 Les unités minimales de mémoire

Le concept d'unité minimale de mémoire est une généralisation de la notion d'*attente*. Cette notion intuitive apparaît dans de nombreux travaux en dialogue homme-machine. En outre, elle peut être rattachée aux mécanismes de prédictions des systèmes de compréhension orale [Bilange, 1992, p. 89]. La notion d'attente, quant à elle, se situe clairement à un niveau pragmatique.

Il s'agit d'une notion importante car elle entre en compte dans le processus de compréhension de l'agent logiciel. Les attentes facilitent en effet une « interprétation » des énoncés qui consiste à mettre en correspondance, d'une part, les connaissances actives de l'interlocuteur signifiées par les attentes — connaissances inférables comprises —, et d'autre part les indices sémantiques de l'analyse des énoncés du locuteur. Dans la majorité des travaux mentionnés ci-après, il s'agit d'une interprétation statique car le contexte réel de chaque dialogue n'y est pas pris en compte. On s'appuie sur l'idée qu'un contexte préétabli suffit pour des dialogues orientés pas la tâche. Ainsi, les attentes font office de contexte d'interprétation. Elles sont établies à partir du modèle de la tâche et supposées être toujours identiques.

Ci-dessous, nous passons tout d'abord en revue quelques systèmes faisant explicitement appel à la notion d'attente. Nous définissons ensuite les unités minimales de mémoire du modèle Génédic.

3.2.1.1 Utilisations des attentes dans d'autres travaux

Dans le système Diabolo, fondé en partie sur la planification, les attentes « sont relatives à ce que l'utilisateur est susceptible de dire (ce qui implique qu'il n'est pas obligé de le dire), en raison des buts et des plans en cours de développement. » [Vilnat, 1997, p. 163]. Les attentes, sans être explicitement liées entre elles, servent à reconnaître les actes de langage présents dans l'intervention de l'utilisateur, afin de déterminer ses intentions, son but et de reconnaître son plan.

Dans une approche cognitive du TALN (Traitement Automatique du Langage Naturel), Dupont propose un modèle des attentes du lecteur pour le calcul de la référence [Dupont, 1996]. Dans ce modèle, les attentes sont des entités associées à un « schéma » évoqué par le texte. L'auteur propose d'associer à chaque attente un nombre représentant la « saillance » de l'attente, c'est-à-dire la probabilité de l'apparition de cette attente dans la suite du texte.

Dans le modèle COALA de Lehuen, nous avons vu au chapitre précédent (cf. Sect 2.4) que les attentes sont fortement liées à l'activité applicative du système. Elles contraignent le déroulement du dialogue ce qui ne permet pas encore pleinement de différencier le dialogue en lui-même de son utilisation pour mener à bien une tâche. Nous avons vu également que

l'ébauche de solution présentée après l'expérimentation du système AMI1, fondé sur le modèle COALA, n'était pas satisfaisante (cf. Sect. 2.4.2.1).

Dans le système Halpin-Documentaire [Rouillard, 2000], Rouillard s'appuie sur la notion d'attente pour prédire les interventions de l'utilisateur. Les attentes se succèdent dans un ordre prédéterminé au fur et à mesure que la tâche progresse (cf Fig. 3.1). La modélisation des attentes s'apparente plus à un automate d'état fini qu'à une hiérarchisation comme nous l'avons proposé. Nous avons déjà évoqué à la fin du chapitre précédent le risque d'une telle approche : le système de dialogue n'est pas capable de suivre autre chose que ce qui est prévu à la conception, ce que tente d'éviter l'auteur en distinguant la gestion du dialogue de la gestion des attentes. Le modèle Génédic tente de remédier à ce problème par une généralisation de la notion d'attente.

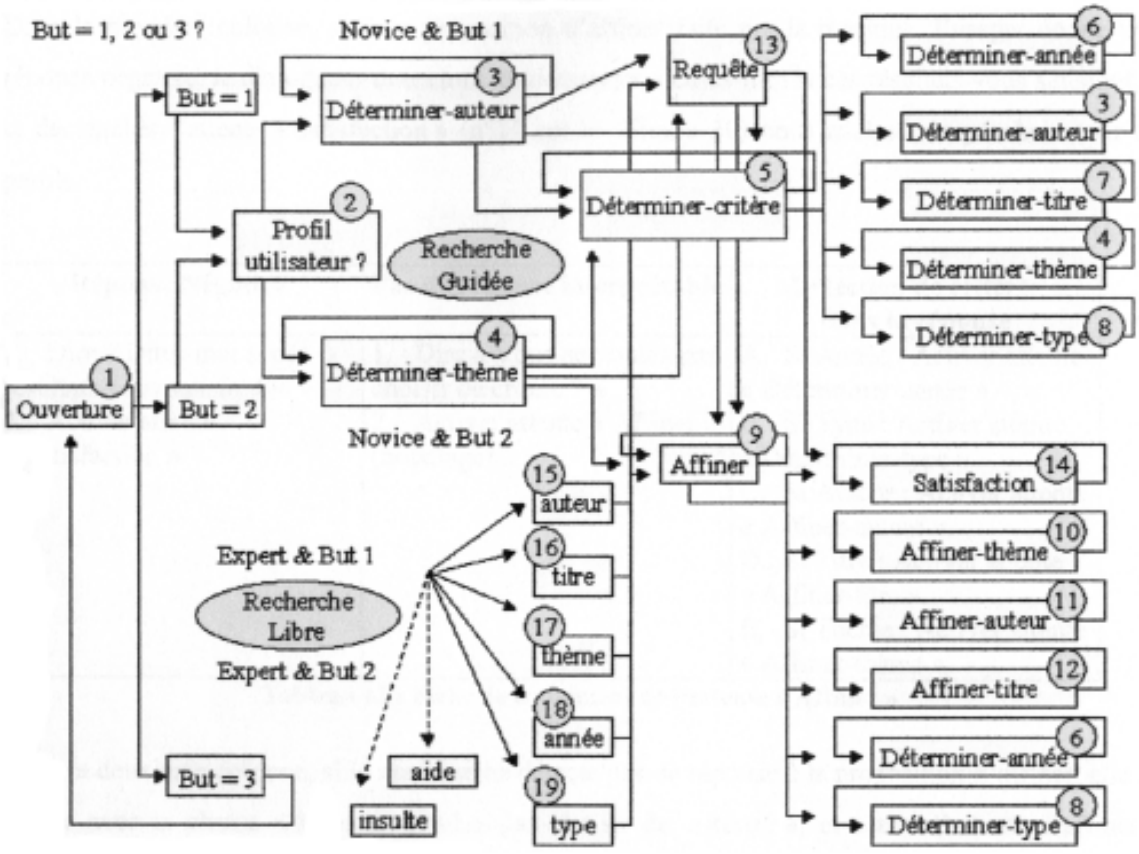


FIG. 3.1 – Les attentes dans le système Halpin-Documentaire [Rouillard, 2000].

3.2.1.2 Les UMM

Dans les travaux que nous venons de mentionner, l'attente possède le statut d'**objet**. Dans le modèle Génédic, l'attente n'est plus un objet mais un **état** particulier d'un objet plus générique : l'unité minimale de mémoire (UMM). Pour modéliser les interactions verbales

comme un processus de co-construction du sens s'appuyant sur un terrain commun, il faut abandonner la vision trop simplificatrice de l'unique existence d'attentes et prendre un point de vue plus global en prenant mieux en compte ce qui se passe réellement lors d'un dialogue.

L'UMM est une entité symbolique représentant les éléments activés dans la mémoire de l'interaction. Elle permet de réunir sous une seule désignation, d'une part, les objets que l'interactant escompte partager, et d'autre part les objets que l'interactant suppose être partagés.

Une UMM est un objet qui possède un certain nombre de champs caractérisant son état, les autres UMM qui lui sont liées, le nombre d'intentions par types dont elle a été l'objet, et enfin son statut selon qu'il s'agit d'un élément de monstration ou qu'il s'agit d'un élément de prédication.

3.2.2 Structuration des unités mnésiques

Les structures d'actions

Par l'interaction et pour interagir, un interactant construit des représentations à partir de l'interprétation des énonciations de l'autre interactant et à partir de ses inférences, certains des éléments de ces représentations étant considérés comme partagés.

Les éléments mnésiques sont organisés en **structures d'actions** du type action / actant(s) / circonstant(s). Ce ne sont cependant pas des structures syntaxiques mais des structures pragmatiques. Les actants sont les sujets (réels ou imaginaires) participant à l'action. Les circonstants sont les circonstances du déroulement de l'action et les objets (réels ou imaginaires) sur lesquels l'action prend effet. Enfin, l'action met en relation les actants et les circonstants. Elle fait référence à un schéma d'action qui prend en charge l'exécution de l'action correspondante. Par exemple, si la structure d'action correspond à la prise d'un message, le schéma d'action sera exécuté lorsque la structure d'action sera dans un état particulier (cf. Sect 3.3).

On retrouve dans ce type de structure les deux parties distinctes que sont la partie prédication et la partie monstration définies par Nicolle pour les actes illocutoires (cf. Sect 2.3.2). La prédication correspond aux actants et à l'action, tandis que la partie monstration « qui détermine le thème, le rhème, le lieu, le temps, la manière... » [Nicolle & Saint-Dizier De Almeida, 1999b, p. 130], correspond aux circonstants.

Lorsque l'activation d'une structure d'action en mémoire de travail provient directement de l'interprétation d'un énoncé, la force illocutoire de cet énoncé est associée à la structure d'action en question (cette force est également signifiée par une UMM). Par ailleurs, le système lui-même peut avoir comme attente une certaine force illocutoire sans qu'y soit attachée de contenu propositionnel. Par exemple, au début d'un dialogue, il peut s'attendre à une requête sans qu'il puisse spécifier sur quoi portera cette requête. À l'opposé, dans d'autres cas, il peut s'attendre à ce qu'une certaine action soit demandée sans qu'il connaisse *a priori* la force qui lui sera associée : une action peut être ordonnée, suggérée ou simplement demandée... De plus,

le système doit pouvoir modifier une force illocutoire qu'il croyait être un ordre et qui se révèle être plutôt une simple proposition. Cette manière flexible de pouvoir attacher ou détacher au gré de l'interprétation des énonciations, une UMM représentant une force illocutoire à une structure d'action, permet de suivre le contexte interactionnel réel. Enfin, la taxinomie des forces illocutoires sur laquelle nous nous appuyons est celle de Nicolle (présentée p. 31).

Les représentations arborescentes

Nous représentons les structures sous forme **d'arborescences** afin de rendre compte des contraintes des structures d'actions. Ces arborescences ne sont pas des hiérarchies d'actes de dialogues comme celles, par exemple, du système $\Delta ELTA$ de Bunt[9] [Bunt, 1995]. Une arborescence représente une structure d'action à laquelle peut correspondre un seul acte illocutoire. En général, il faut plusieurs actes illocutoires pour compléter une structure d'action.

Les structures d'actions sont liées soit à la tâche applicative du système soit à la tâche langagière. Par exemple, dans le système AMI (cf. la seconde partie du document), on trouvera des structures d'actions applicatives, comme celle de la Fig. 3.2, et des structures d'actions langagières, comme celle de la Fig. 3.3.

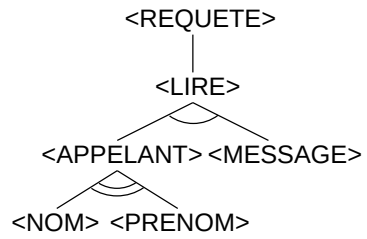


FIG. 3.2 – Exemple de structure d'action liée à l'activité applicative du système AMI2.

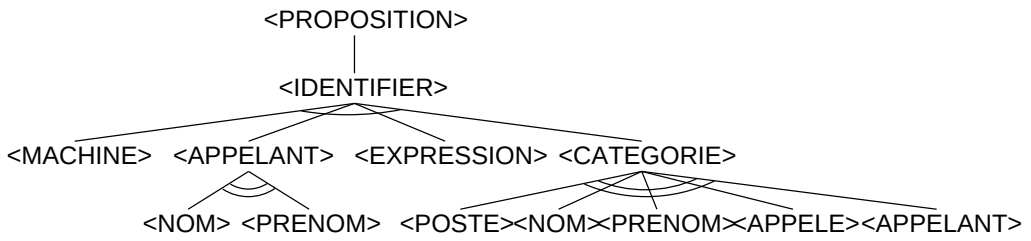


FIG. 3.3 – Exemple de structure d'action liée à l'activité langagière du système AMI2.

Ainsi, les structures d'actions applicatives doivent être déterminées par une expertise de l'application visée. Elles doivent être les plus exhaustives possibles afin de rendre compte de toutes les situations que le système peut rencontrer. Ces structures doivent être établies pour chaque application visée. En revanche, les structures d'actions langagières et dialogiques ne sont pas liées à l'application. Elles sont « portables » d'une application à l'autre.

[9] Bunt considère deux grands types d'actes de dialogue : les actes de dialogue normaux et les actes de gestion du dialogue.

Une structure arborescente permet de rendre compte des contraintes sur les actions représentées. Elle permet de spécifier les éléments optionnels et les éléments obligatoires de la structure d'action représentée. Les éléments obligatoires sont groupés ensembles et les éléments optionnels également. Ceci se fait en typant les liens entre nœuds père et nœuds fils des deux manières suivantes :

- Un nœud père maintient une relation de disjonction avec ses nœuds fils optionnels : l'arborescence est dite ***en OU***. Par exemple, on peut imaginer qu'une action sans actant possède des circonstants optionnels (c'est rarement le cas dans les dialogues finalisés). Ce type de lien est représenté graphiquement Fig. 3.4 et noté linéairement sous forme fonctionnelle infixée : $< X > (< Y > \vee < Z >)$. Il est à noter qu'il s'agit d'une relation inclusive. Le modèle Génédic peut être modifié si on veut distinguer les liaisons disjonctives exclusives des liaisons disjonctives inclusives.

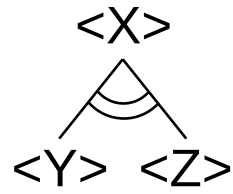


FIG. 3.4 – Une arborescence en OU.

- Un nœud père maintient une relation de conjonction avec ses nœuds fils obligatoires : l'arborescence est dite ***en ET***. Par exemple, pour que certaines actions soient exécutées, la présence de tous leurs actants et de tous leurs circonstants est obligatoire. Ce type de lien est représenté graphiquement Fig. 3.5 et noté sous la forme : $< X > (< Y > \wedge < Z >)$.

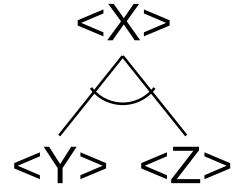


FIG. 3.5 – Une arborescence en ET.

Il peut arriver qu'une action ait besoin de plusieurs actants tout en possédant plusieurs circonstants optionnels. Un exemple de combinaison est donné Fig. 3.6. Les éléments obligatoires sont représentés graphiquement et notés avant les éléments optionnels. Pour simplifier le modèle, nous avons considéré que les arborescences ET-OU devaient être transformées en arborescences ET avec ajout d'une sous-arborescence OU dont la racine est un élément *vide*. Par exemple, l'arborescence de l'exemple notée $< X > (< Y > \wedge < Z >)(< W > \vee < T >)$ est transformée en $< X > (< Y > \wedge < Z > \wedge \|U\| (< W > \vee < T >))$. L'élément $\|U\|$ est un

élément spécial qui prend l'état de son frère gauche. Quand celui-ci change, il change de même. Un élément spécial ne peut pas être l'objet d'une intention de communication puisqu'il s'agit d'un élément vide, sans signification pour l'arborescence, même s'il intervient normalement dans le processus de génération.

FIG. 3.6 – Transformation d'une arborescence ET-OU.

3.3 Interprétation et états des unités mnésiques

3.3.1 L'interprétation des énoncés

De notre point de vue, l'interprétation des énoncés de l'utilisateur par la machine consiste à confronter le résultat de l'analyse sémantique avec les éléments présents en mémoire de travail. Autrement dit, l'interprétation se fait en fonction ou non d'une sorte de ***contexte cognitif*** [10] dont une partie est peut-être déjà partagée.

Il existe plusieurs cas selon que l'interprétation se fait en contexte, c'est-à-dire selon les éléments présents en mémoire, ou hors contexte c'est-à-dire sans tenir compte des éléments mnésiques. De plus, l'interprétation se fait selon que l'énoncé interprété est porteur d'un seul sens ou de plusieurs sens sémantiques. En conséquence, il existe donc quatre cas à considérer. Nous les avons résumé Tab. 3.1.

Résultat de l'analyse sémantique	Énoncé porteur d'un seul sens	Énoncé porteur de plusieurs sens
Interprétation en contexte	Appariement du sens avec les éléments de la mémoire => Changement d'état des UMM appariées	Appariement d'un des sens avec les éléments de la mémoire => Changement d'état des UMM appariées
Interprétation hors contexte	Pas d'appariement => Création automatique d'UMM pour mémoriser le sens de l'énoncé	Pas d'appariement => Deux possibilités : - Hypothèse d'interprétation - Poser la question à l'utilisateur

TAB. 3.1 – Les quatres cas de l'interprétation.

Le cas le plus favorable est lorsque l'analyseur sémantique attribue un seul sens possible à l'énoncé et que les éléments de signification (prédication et monstration) de ce sens unique cor-

[10] Dans leur livre, Sperber & Wilson nomme cela *l'environnement cognitif* [Sperber & Wilson, 1989].

respondent aux éléments d'une arborescence. Dans ce cas, nous disons qu'il y a ***appariement*** des UMM. La conséquence est un changement de l'état des UMM appariées.

Le second cas se produit lorsque l'analyseur sémantique attribue un seul sens possible à l'énoncé mais que ce sens ne correspond à aucune des UMM actives. Dans ce cas, cet énoncé est tout de même pris en compte par le système grâce à une création automatique d'UMM mémorisant le sens de cet énoncé. Cette arborescence sera peut-être prise en compte ultérieurement au cours du dialogue.

Les deux autres cas se produisent lorsque l'analyseur sémantique attribue non plus un seul sens mais plusieurs sens à l'énoncé de l'utilisateur. Si parmi ce potentiel de sens un seul correspond aux éléments présents en mémoire de travail, nous nous retrouvons dans le premier cas évoqué précédemment. Il y a donc un changement de l'état des UMM appariées, mais il faut remarquer qu'il se peut alors que la machine se méprène sur le sens intentionné par l'utilisateur. La réponse de l'utilisateur validera ou invalidera cette interprétation de la machine.

Enfin, le dernier cas se produit lorsqu'aucun des sens que l'analyseur sémantique attribue à l'énoncé ne correspond aux éléments en mémoire de travail. Nous avons ici deux possibilités de comportement : ou le système choisi un sens au hasard et fait l'hypothèse que c'est celui intentionné par l'utilisateur, avec un risque de devoir revenir sur cette hypothèse dans la suite du dialogue, ou bien le système ne pose pas d'hypothèse mais pose directement la question à l'utilisateur sur le sens intentionné parmi ceux que l'analyseur donne. Pour une raison de robustesse des échanges, nous préférons le second type de comportement.

La section suivante détaille chaque état que peut prendre les UMM au gré de l'interprétation et du déroulement de l'application.

3.3.2 Les états mnésiques

Les unités de la mémoire de travail passent par différents **états d'activations** (cf. Fig. 3.7). Avant d'être *active* en mémoire de travail, une UMM est *inactive* en mémoire à long terme. Le passage de l'un à l'autre état se fait par les fonctions d'oubli et de remémoration dont nous avons déjà amplement débattues (cf. Sect. 3.2). Une fois activée, une UMM peut passer par différents états actifs.

Étant donné que les éléments des structures d'action sont soit issus des énonciations, soit le sujet des énonciations, les états des UMM sont en relation directe avec ces énonciations. Ils ont donc été définis en prenant en compte les notions de réussite et de satisfaction de la théorie des actes de langage développées par Vanderveken [Vanderveken, 1990]. Cependant, nous ne nous sommes pas appuyé sur les définitions de Vanderveken, mais sur celles de Nicolle et Saint-Dizier De Almeida (présentées p. 34), car elles sont en adéquation avec le modèle de la co-construction du sens en interaction.

Normalement, une UMM est d'abord *en attente*, puis devient *réussie*, et enfin devient

accomplie. Nous allons expliciter chacun de ces trois états possibles à partir des définitions de la valeur interlocutoire (cf. p. 31) et des contraintes de satisfiabilité des actions langagières[11] (cf. p. 34).

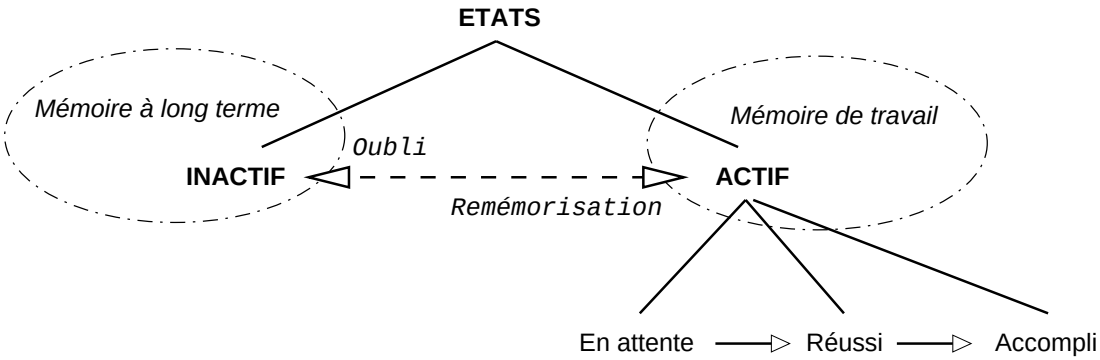


FIG. 3.7 – Les états mnésiques des UMM.

L'état en attente

Cet état correspond à la notion d'attente définie précédemment. Seulement, il ne s'agit plus ici d'un objet — *l'attente* — mais du statut d'un objet — *une UMM en attente*. Une UMM $< X >$ en attente est notée X sans signe distinctif.

C'est l'état par défaut d'un élément mnésique actif. L'activation d'une UMM provient de l'activité langagière et de l'activité applicative. Si elle provient de l'activité applicative, cette activation est sur l'initiative du système. Cela signifie qu'il s'attend à la présence de cet élément dans le terrain commun. Il s'agit donc d'une hypothèse émise sur les propos des échanges à venir du dialogue. À l'opposé, si l'activation provient de l'activité langagière (la partie interprétation des énoncés de l'utilisateur), elle est sur l'initiative de l'utilisateur. Cela signifie que cet élément fait potentiellement partie du terrain commun.

Ce cas de figure se rencontre lorsque le système ne possède aucune attente correspondant aux éléments de l'interprétation de l'énoncé, et que les processus langagiers ont alors activé des UMM pour les mémoriser. Dans les deux cas, une UMM en attente change d'état s'il y a partage d'une valeur interlocutoire entre le système et l'usager.

L'état réussi

Un acte illocutoire est réussi lorsque les interactants lui attribuent une valeur interlocutoire. Par conséquent, si un élément de monstration ou de prédication de cet acte réussi correspond à une UMM en attente, alors celle-ci est automatiquement réussie. Par exemple, à partir de l'arborescence Fig. 3.8, le système peut proposer à l'utilisateur (l'UMM $< APPELANT >$) de prendre un message pour l'appelé (l'UMM $< APPELE >$). L'attente $< PRENDRE >$

[11] La première contrainte définie par Nicolle porte sur l'acceptabilité de l'acte (cf. Sect 2.3.2.3.2), c'est-à-dire sur la dimension sociale de l'échange. Nous avons déjà dit que nous ne prenions pas en considération cette dimension dans notre travail de modélisation. Elle n'intervient donc pas dans le modèle Génédic.

de l'arborescence deviendra réussie si et seulement si le système interprète que l'utilisateur accepte ou rejette sa proposition. De plus, une force illocutoire commissive sera associée à cette arborescence puisque c'est la force de l'énoncé du système qui est satisfait par l'utilisateur.

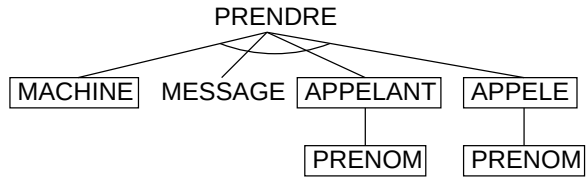


FIG. 3.8 – Arborescence d'une structure d'action correspondant à une prise de message.

Une UMM réussie est considérée comme faisant partie du terrain commun. Cet état dépend donc de l'intercompréhension des participants. Une UMM $< X >$ réussie est notée [X] graphiquement et $|X|$ fonctionnellement.

L'état accompli

Un acte illocutoire réussi est satisfiable si les deux contraintes portant sur la capacité à satisfaire et la volonté de satisfaire sont levées. La contrainte 2 (la capacité à satisfaire) ne peut être levée que si l'état du monde permet la satisfaction de l'acte et si les capacités de celui qui est engagé dans l'acte (l'allocutaire pour un directif et le locuteur pour un commissif ou un assertif) sont suffisantes. La contrainte 3 (la volonté de satisfaire) est levée lorsque les croyances, les désirs et les intentions de l'allocutaire sont compatibles avec la satisfaction de l'acte.

Dans le modèle Génédic, les attitudes mentales de l'allocutaire — l'agent informatique — ne sont pas explicitement représentées. Elles sont remplacées par des lois comportementales et interactionnelles[12] (voir Sect. 3.4.1). Ces lois et l'accomplissement des unités de mémoire vont conditionner la satisfaction des actes de l'utilisateur. Les UMM réussies, représentant des éléments de monstration, deviennent accomplies si elles sont correctement référencées par le système ou simplement catégorisées[13], ce qui aide à lever la contrainte 2. Les UMM réussies représentant des éléments de prédication, deviennent accomplies si elles sont faisables, ce qui aide à lever les contraintes 2 et 3.

De plus, une UMM ne peut être accomplie que si toutes les autres UMM qui lui sont liées conjonctivement sont accomplies ou, au moins, si l'une des UMM qui lui est liée disjonc-

[12] La granularité de la représentation des attitudes mentales du modèle Génédic actuel est donc très faible. Afin de mieux pouvoir modéliser les états mentaux, un grand travail reste à faire dans cette direction, notamment avec l'aide des approches par plans. Une plus grande granularité permettrait sans doute de doter le système de réaction plus fine, et ainsi de donner l'image d'une machine plus intelligente. En particulier, cela lui donnerait une « personnalité numérique » plus riche. Mais cela est-il nécessaire ?

[13] Dans le cadre du modèle Génédic, nous dirons qu'une UMM est catégorisée lorsqu'il lui est attribuée une catégorie sémantique. Par exemple « Thierry » est catégorisé comme un prénom, « Lemeunier » est catégorisé comme un nom. Par contre l'UMM $< APPELANT >$ est catégorisée comme la personne qui appelle et référencée comme l'unique personne s'appelant « Thierry Lemeunier ».

tivement est accomplie. Le changement de l'état réussi à l'état accompli se fait par propagation du bas vers le haut. Ainsi, dans l'arborescence en ET de l'exemple Fig. 3.9, l'UMM $< PARLER - A >$ ne sera accomplie que lorsque l'action correspondante sera exécutable. Cela ne peut être vrai que si l'UMM $< APPELANT >$ devient accomplie et que l'appelé (l'UMM $< APPELE >$) est disponible pour parler à l'appelant.

Une UMM $< X >$ accomplie est notée $\boxed{X}$ graphiquement et $\underline{\overline{X}}$ fonctionnellement.

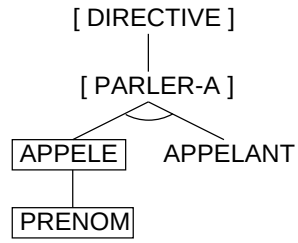


FIG. 3.9 – Structure d'action de la mise en contact d'un appelant avec un appelé.

Les deux exemples Fig. 3.10 illustrent la manière dont plusieurs UMM d'états différents peuvent se lier les unes aux autres. Dans l'exemple (a), l'UMM $< W >$ est réussie mais non-accomplie, car les UMM $< U >$ et $< V >$ qui lui sont conjonctivement liées sont encore en attente. Dans l'exemple (b), l'UMM $< X >$ est réussie mais non-accomplie, car l'UMM $< T >$ est non-accomplie du fait du non-accomplissement de $< U >$ et de $< V >$. Tout ceci entraîne que $< W >$ soit elle aussi non-accomplie.

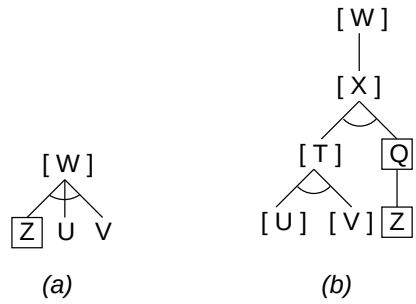


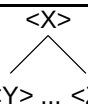
FIG. 3.10 – Exemples d'arborescences d'éléments de différents états.

Le tableau Tab. 3.2 résume les conventions de notations que nous avons adoptées pour le modèle Génédic.

3.4 Génération des intentions de communication

3.4.1 Lois comportementales et interactionnelles

La production des énoncés du système provient des intentions de communication, qui apparaissent du fait de l'activation de certaines configurations des structures d'action en mémoire

Notation graphique	Notation fonctionnelle	Signification
X	X	L'UMM X est en attente.
[X]	$\|X\|$	L'UMM X est réussie.
$\boxed{X}$	$\overline{\underline{X}}$	L'UMM X est accomplie.
<X>	$< X >$	L'UMM X est dans un état indifférent.
<X> | <Y>	$< X > (< Y >)$	L'UMM X est liée à l'UMM Y.
	$< X > (< Y > \wedge < Z >)$	Les UMM sont liées par une conjonction.
	$< X > (< Y > \vee < Z >)$	Les UMM sont liées par une disjonction.
	$< X > (< Y >^{+})$	L'UMM X est liée à plusieurs UMM d'états indifférents.

TAB. 3.2 – Conventions de notation des arborescences.

de travail du système. Les intentions de communication ne sont donc pas planifiées mais **spontanées**. Après cette étape automatique, un énoncé est construit sur la base des intentions et des contenus intentionnels (ce sur quoi porte l'intention).

Ainsi, dans le modèle Génédic, par opposition aux actions non-langagières qui résultent de processus de raisonnement personnels, les actions langagières spontanées participent aux processus de co-construction du sens interlocutoire. Ceci aboutit à un comportement réactif, par opposition à un comportement préétabli, et cela permet une plus grande autonomie de l'activité conversationnelle vis-à-vis de l'activité applicative qui, elle, peut être planifiée ou, plus généralement, modélisée par des formalismes logiques.

Les configurations « intéressantes » n'ont pas été établies au hasard mais en appliquant des **lois** comportementales et interactionnelles. Nous avons actuellement défini trois lois suffisantes pour motiver les interactions du système de dialogue avec son environnement, en particulier avec l'agent humain. Nous présentons d'abord ces lois, puis nous discutons de leurs applications et de leurs limites.

La première loi est une loi comportementale générale. Elle régit les croyances et les désirs du système en assurant la liaison entre le dialogue et l'application. Cette loi intervient pour lever la contrainte 3 de la satisfaction des actes illocutoires (contrainte sur la volonté de satisfaire). De fait, elle permet l'accomplissement des actions représentées en mémoire. Nous l'avons formulée de la manière suivante :

Première loi
Un système de dialogue homme-machine finalisé collaboratif doit faire tout ce qui est de son ressort pour satisfaire l'utilisateur.

La deuxième et la troisième loi gouvernent le comportement interactionnel du système. Elles sont le « moteur » de l'interaction du système avec son environnement car elles le poussent à interagir afin de respecter la première loi. Nous avons formulé ces lois de la manière suivante :

Deuxième loi
Un système de dialogue homme-machine finalisé ne doit informer l'utilisateur que de ce que le système croit que l'utilisateur ne sait pas.

Troisième loi
Un système de dialogue homme-machine finalisé doit s'informer de ce qu'il ne sait pas auprès de l'utilisateur.

Ces lois régissent toutes les deux la génération des intentions de communication mais diffèrent par la direction de la communication : la deuxième loi gère la direction de la machine vers l'utilisateur, dans la mesure où c'est de la machine que provient l'information à partager, alors que la troisième loi gère la direction de l'utilisateur vers la machine dans la mesure où, dans ce cas, l'information est détenue initialement par l'utilisateur.

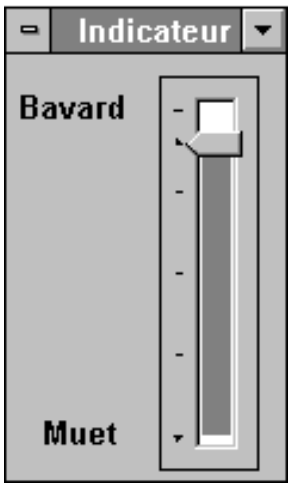
Nous avons donné une plus grande importance à la troisième loi par rapport à la deuxième : le système doit en priorité acquérir les informations nécessaires dans le but de progresser dans ses différentes activités. Ceci a une conséquence sur le choix de l'intention à privilégier vis-à-vis des autres intentions générées au même instant à partir des éléments mnésiques : les intentions dont la troisième loi est à l'origine sont sélectionnées avant les intentions éventuelles fondées sur la deuxième loi.

Application et transgression des lois

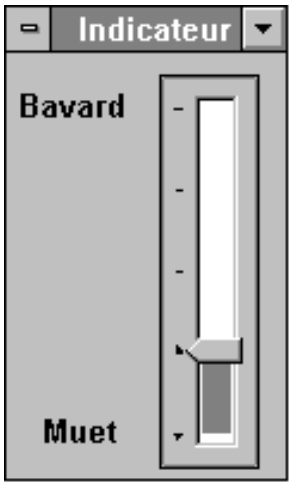
Le comportement du système est normal lorsqu'il suit strictement les trois lois que nous avons définies. En revanche, lorsqu'elles sont transgressées, dans un sens ou dans l'autre, le système peut avoir un comportement « insolite ». Ainsi, si la deuxième loi n'est pas appliquée strictement, le système devient très bavard, avec de nombreux actes redondants. À l'extrême opposé, si les lois comportementales ne sont pas appliquées, le système devient muet puisque plus rien ne le motive à interagir avec l'utilisateur.

Afin d'illustrer notre propos, deux exemples de dialogues sont donnés Fig. 3.11. Dans le premier dialogue, la seconde loi n'est pas strictement appliquée (la machine donne des informations que l'utilisateur possède déjà), ce qui entraîne de nombreux énoncés redondants. Au contraire, dans le second dialogue, le comportement semble normal lorsque les lois sont

strictement suivies. Ces dialogues sont tirés d'une version intermédiaire du système AMI, qui se situe entre l'expérimentation de la première version et le développement de la seconde (voir la deuxième partie de la thèse).



M1 : Pouvez-vous m'indiquer quelle est votre requête ?
H1 : je voudrais parler à daniel
M2 : Vous voulez contacter daniel,
je sais que vous voulez parler à un membre du laboratoire,
pouvez-vous m'indiquer votre identité ?
H2 : thierry lemeunier
M3 : Vous voulez contacter daniel,
vous êtes thierry lemeunier,
je sais que vous voulez parler à un membre du laboratoire,
vous ne pouvez pas parler à votre correspondant,
voulez-vous lui laisser un message ?
H3 : non
M4 : Nous vous remercions d'avoir utilisé notre service de messagerie.



M1 : Pouvez-vous m'indiquer quelle est votre requête ?
H1 : je voudrais parler à daniel
M2 : pouvez-vous m'indiquer votre identité ?
H2 : thierry lemeunier
M3 : vous ne pouvez-pas parler à votre correspondant,
voulez-vous lui laiiser un message ?
H3 : non
M4 : Nous vous remercions d'avoir utilisé notre service de messagerie.

FIG. 3.11 – Exemple de comportements selon le degré de respect des lois comportementales.

Mise en oeuvre des lois comportementales

Avec de telles lois, on pourrait penser avec raison que beaucoup trop de choses seront dites en particulier au regard des deux dernières lois. En effet, il est certain que, par elles-mêmes, ces lois sont trop générales pour produire un comportement interactionnel *pertinent,* au sens des maximes de conversation de Grice : beaucoup trop de questions devraient être posées, en même temps que beaucoup trop d'informations devraient être données.

À titre d'illustration, observons Fig. 3.12 l'extrait d'un « dialogue » fictif, dans lequel les trois lois sont appliquées sans restriction. Les interventions Mi du système sont ici remplacées par les intentions communicatives avant l'étape de génération linguistique des énoncés

correspondants, tandis que les interventions Hi sont celles d'un utilisateur qui réagirait à des interventions Mi correctes.

M1a : Question sur <APPELANT> (pouvez-vous m'indiquer votre identité)
M1b : Question sur <LAISSER-MESSAGE> (voulez-vous laisser un message)
M1c : Question sur <LIRE-MESSAGE> (voulez-vous lire vos messages)
M1d : Question sur <PARLER> (voulez-vous parler à un membre du laboratoire)
M1e : Question sur <REQUETE> (pouvez-vous m'indiquer quelle est votre requête)

H1 : je voudrais parler à thierry

M2a : Information sur <APPELE> (je vous informe que je connais la personne que vous voulez contacter)
M2b : Question sur <APPELANT> (pouvez-vous m'indiquer votre identité)

H2 : daniel luzzati

M3a : Information sur <MESSAGE> (je vous informe qu'il y a un message pour vous :
M3a : Lemeunier Thierry vous a laissé un message le mercredi 30 décembre à 15h10
M3a : Salut vieille branche)
M3b : Information sur <APPELANT> (je vous informe que vous êtes la personne qui appelle)
M3c : Information sur <APPELE> (je vous informe que je connais la personne que vous voulez contacter)
M3d : Indication sur <PARLER-A> (je vous indique que vous ne pouvez pas parler à un membre du laboratoire)
M3e : Indication sur <REQUETE> (je vous indique que je ne peux pas satisfaire votre requête)
M3f : Question sur <MESSAGE> (voulez-vous laisser un message)

H3 : non

M4a : Information sur <APPELANT> (je vous informe que vous êtes la personne qui appelle)
M4b : Information sur <APPELE> (je vous informe que je connais la personne que vous voulez contacter)
M4c : Indication sur <PARLER> (je vous indique que vous ne pouvez pas parler à un membre du laboratoire)
M4d : Indication sur <REQUETE> (je vous indique que je ne peux pas satisfaire votre requête)
M4e : Proposition sur <APPELE> (je vous propose de contacter Vivet Martial)
M4e : Proposition sur <APPELE> (je vous propose de contacter Lehuen Jérôme)

H4 : non

FIG. 3.12 – Un exemple de comportement interactionnel non-pertinent.

Examinons ce pseudo dialogue. Tout d'abord, en M1, le système pose cinq questions car il possède initialement autant d'attentes. Un comportement raisonnable serait de n'en poser qu'une seule, en l'occurrence la dernière (M1e), celle portant sur l'UMM $< REQUETE >$ qui représente l'attente d'une énonciation de force directive. Ensuite en M2, le système pose une question (M2b) et donne une information (M2a). La question est justifiée par le but applicatif, mais l'information donnée est-elle nécessaire ? Elle ne l'est certes pas, mais sa présence n'est pas non plus incongrue : on peut l'interpréter comme ayant une double fonction : d'une part, celle d'expliciter l'accord sur la valeur interlocutoire de M1e, et d'autre part celle de justifier la question portant sur l'UMM $< APPELANT >$. En M3, une fois que l'appelant et l'appelé sont identifiés, le système donne un message destiné à l'appelant (M3a). Cette partie de M3 est justifiée par l'application. L'intention M3b n'est pas obligatoire. Cependant, comme précédemment, cette intention peut être interprétée comme l'explicitation de l'accord sur la valeur interlocutoire de M2b. L'intention M3c est identique à celle de M2a. Cette redondance n'est pas justifiée. L'intention M3d est là pour indiquer que la requête H1 ne peut être satisfaite pour une raison non spécifiée. De plus, elle permet d'expliquer l'intention M3f par laquelle le

système propose une solution alternative. L'intention M3e n'est pas nécessaire car elle est redondante avec M3d. Pourtant, elle pourrait être interprétée comme une volonté d'insister sur l'insatisfiabilité de la requête H1. En M4, à la dernière intervention de la machine, les intentions M4a à M4d sont toutes les quatre redondantes avec des intentions précédentes. Seule est justifiée l'intention M4e car le système continue d'essayer de satisfaire l'utilisateur.

Un tel comportement est loin de respecter le principe de coopération observé par Grice dans les conversations naturelles. Dans l'exemple précédent, il est flagrant que la maxime de quantité — *que votre contribution contienne autant d'informations qu'il est requis et ne contienne pas plus d'informations qu'il est requis* — n'est pas respectée et que ce non-respect n'a pas pour intention de signifier quelque chose de particulier à l'utilisateur. Même si, nous le rappelons, les conversations homme-homme sont à bien des égards différentes des conversations homme-machine, l'agent artificiel doit être adapté (ou s'adapter automatiquement) aux capacités interactionnelles de ses utilisateurs humains pour être compris. Nos trois lois répondent donc à une condition nécessaire mais non suffisante pour assurer la pertinence des interactions.

Notons que le problème auquel nous sommes confronté est tout à fait normal. Notre point de vue sur le dialogue nous a conduit à le concevoir comme un processus de co-construction du sens en interaction, et ce faisant, à essayer de dégager l'activité dialogique de l'activité applicative. La pertinence des énonciations de la machine, qui est garantie dans les modèles fondés sur la planification dans la mesure où elle y est « câblée », ne l'est plus dans le modèle Génédic. Nous allons voir maintenant quelle solution nous avons adoptée pour tenter de garantir la pertinence, au regard de la situation interactionnelle, des énoncés de la machine.

3.4.2 Les configurations remarquables

Les actions de communication sont issues d'intentions communicatives propres au système de dialogue. Elles sont créées, en respectant les trois lois définies ci-dessus, en fonction des arborescences et de l'état de leurs éléments constitutifs. Le travail consiste donc ici à répertorier l'ensemble des configurations susceptibles d'être à l'origine de la création d'intentions communicatives pertinentes. Ces configurations doivent suivre les quatre cas principaux suivants :

- Le système doit informer l'utilisateur qu'il peut accomplir une action demandée dans le cas où toutes les UMM de l'arborescence deviennent accomplies.
- Le système doit informer l'utilisateur qu'il ne peut pas effectuer une action demandée dans le cas où certaines UMM de l'arborescence ne sont pas accomplies.
- Le système doit s'informer auprès de l'utilisateur d'informations manquantes pour accomplir une action demandée dans le cas où certaines UMM sont en attente.
- Le système doit informer l'utilisateur qu'il peut accomplir une action (dont les éléments sont en attente) à la place de l'action demandée par l'utilisateur et qu'il ne peut pas accomplir.

Nous avons vu que pour obtenir des énoncés pertinents, les intentions communicatives ne devaient pas être redondantes, injustifiées et surtout contradictoires. Par exemple, si dans une arborescence conjonctive l'UMM racine est réussie sans être accomplie, il ne faut rien dire de son état tant que d'autres UMM restent en attente dans cette arborescence. Ainsi, une intention doit être créée, dans la mesure du possible, en prenant en compte toute l'arborescence, c'est-à-dire sa structure et l'état de tous ses éléments, et non à partir de l'état de quelques UMM constitutives.

Le problème majeur auquel nous sommes confronté ici est que le nombre de configurations à répertorier est *a priori* inconnu. Certes, il n'est pas infini, car les arborescences ne doivent pas pouvoir dépasser une certaine hauteur et un certain nombre d'UMM, mais il reste tout de même très grand. Par exemple, en tenant compte des liens optionnels et obligatoires entre père et fils, il existe 25 920 arborescences de hauteur trois dont chaque nœud a trois fils[14]. L'étude de l'ensemble des arborescences étant impossible à réaliser, nous avons limité notre investigation aux arborescences de hauteur zéro et un.

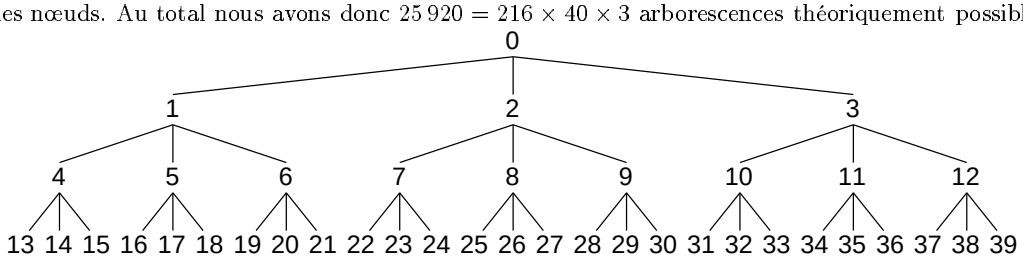
Le risque d'un manque de pertinence est limité par un certain nombre de *règles dialogiques* qui filtrent les intentions générées. Pour résumer, on peut dire que des configurations remarquables assurent le premier niveau de pertinence et que les règles dialogiques assurent le second niveau de pertinence.

Dans cette section, nous allons définir les configurations remarquables à partir des arborescences de hauteur zéro et de hauteur un puis, dans la section suivante, nous montrerons comment sont appliquées ces configurations remarquables.

Les arborescences de hauteur zéro et de hauteur un

Pour répertorier les cas intéressants pour notre objectif, nous avons développé la combinatoire de toutes les arborescences de hauteur zéro (un seul nœud) et de hauteur un (au moins un nœud racine et un nœud fils). À partir des trois états qu'une UMM peut prendre, on peut *théoriquement* développer 9 (3×3) arborescences de hauteur un avec un fils, puis 54 ($9 \times 3 \times 2$) arborescences conjonctives et disjonctives de hauteur un avec deux fils, et enfin 162 (54×3) arborescences conjonctives et disjonctives de hauteur un ayant trois fils.

[14] Il y a 40 nœuds dans une arborescence de hauteur trois dont chaque nœud possède trois fils. D'une part, chacun de ces noeuds peut prendre trois états différents. D'autre part, compte tenu des deux types de liens possibles entre père et fils (obligatoire ou optionnel), il y a 216 arborescences différentes sans tenir compte de l'état des nœuds. Au total nous avons donc $25\,920 = 216 \times 40 \times 3$ arborescences théoriquement possibles.



En fait, le total de 228 $(3 + 9 + 54 + 162)$ cas théoriques n'est pas atteint du fait des dépendances entre nœud père et nœud fils. En effet, nous avons dit qu'une UMM ne pouvait être accomplie que si toutes les autres UMM qui lui sont liées conjonctivement sont accomplies ou que si au moins l'une des UMM qui lui est liée disjonctivement est accomplie. Ainsi, un certain nombre de cas sont *normalement* impossibles. Par exemple, dans la Fig. 3.13, les cas (a), (b) et (c) sont théoriquement impossibles. En particulier, une arborescence conjonctive (cas (b) ou (c)) dans laquelle un nœud racine accompli possède un ou plusieurs nœud fils accompli et un ou plusieurs nœuds frères non-accomplis, est anormale. Si une telle arborescence existe en mémoire de travail, cela signifie que le système a fait une erreur dans la construction de l'arborescence.

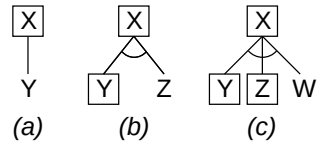


FIG. 3.13 – Extrait des arborescences *normalement* impossibles.

Pour simplifier le modèle, nous avons choisi de ne pas répertorier ces cas anormaux. Nous aurions pu faire le choix inverse et ainsi être en mesure de générer des intentions portant sur les propres erreurs de fonctionnement du système. Il serait en effet intéressant d'étudier comment la machine pourrait réagir[15] lors de ces cas anormaux.

En éliminant les cas anormaux, nous obtenons toutes les arborescences de hauteur zéro et de hauteur un possibles. Ce sous-ensemble n'a cependant pas besoin d'être connu dans son intégralité par la machine car, en effectuant des *opérations de focalisation,* nous obtenons finalement un partitionnement en quatre classes d'équivalence regroupant onze *configurations remarquables.*

Les opérations de focalisations

Les **focalisations** sont des opérations de structuration **temporairement** effectuées sur les arborescences. D'une part, elles permettent de restreindre considérablement le nombre de configurations à prendre en considération par la machine en intervenant dans la détermination des configurations remarquables et d'autre part, elles interviennent normalement sur les structures d'actions avant la phase proprement dite de génération des intentions de l'activité dialogique du système.

Ces opérations consistent à remplacer plusieurs UMM de même état par une seule unité de mémoire, que nous appelons **UMM composée,** dont l'état est celui des UMM qui la composent. Elles sont exécutées autant de fois que cela est possible. Du point de vue mnésique, les opérations de focalisation consistent à simplifier une hiérarchie en assimilant plusieurs

[15] Dans ce genre de situation, la solution idéale serait bien sûr que le système puisse trouver et remédier temporairement ou définitivement à ses erreurs auto-détectées via le processus de génération des intentions.

UMM en une seule, plus générale, représentative des UMM qui la compose. Ainsi, plusieurs UMM sont *vues* comme une seule UMM. Une focalisation permet de masquer temporairement les détails des arborescences avant que soient créées les intentions de communication.

Il existe deux opérations de focalisation : le regroupement horizontal et le regroupement vertical. Un regroupement horizontal est possible entre feuilles[16] de même état et de même père. Par exemple, à la Fig. 3.14, les UMM $< Z >$ et $< W >$ sont regroupables car ce sont des feuilles accomplies de père $< X >$. À l'opposé, les UMM $< X >$ et $< T >$ ne sont pas regroupables, malgré qu'elles soient de même état, car $< X >$ n'est pas une feuille.

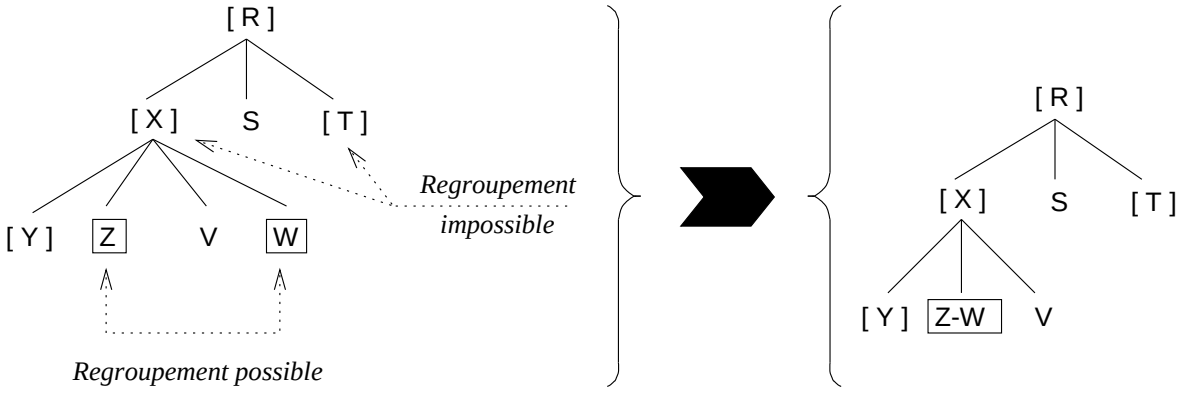


FIG. 3.14 – Exemples de regroupements horizontaux.

Un regroupement vertical est possible entre un nœud racine et un nœud fils de même état. Par exemple, les UMM $< X >$ et $< W >$ de la Fig. 3.15 sont regroupables car $< X >$ est un nœud racine et qu'elles sont réussies. Il en est de même pour les UMM $< Z >$, $< V >$ et $< R >$ moyennant un regroupement horizontal de $< V >$ et $< R >$. En revanche, les UMM $< W >$, $< Y >$ et $< Z >$ de même état ne sont pas regroupables car $< Y >$ n'est pas une feuille.

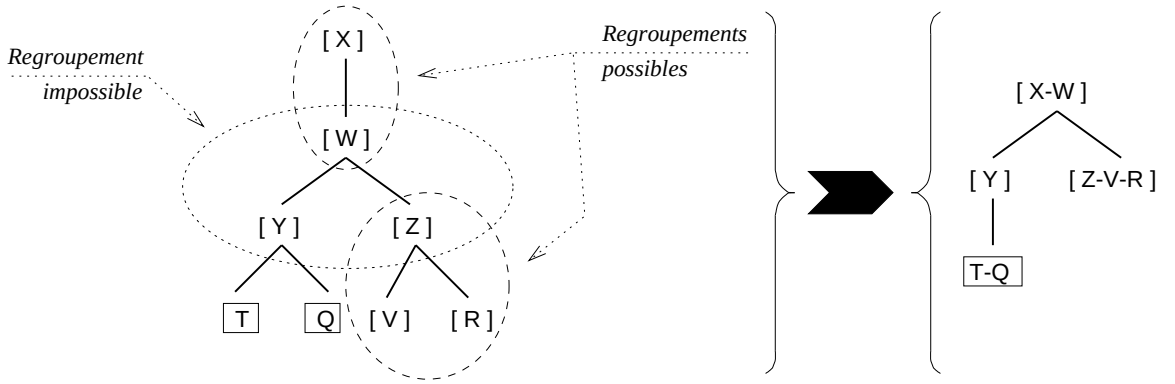


FIG. 3.15 – Exemples de regroupements verticaux.

[16] Une feuille est un nœud sans fils.

Les classes d'équivalences

À l'aide des deux opérations de focalisation et en généralisant les configurations, les cas normaux sont réductibles en onze configurations remarquables représentant l'ensemble de ces cas. Elles sont regroupées en quatre classes d'équivalences (voir Tab. 3.16). Les configurations remarquables assurent le premier niveau de pertinence.

Classe $C1$

Cette classe regroupe les configurations remarquables $C1a$ et $C1b$. La configuration $C1a$ est une UMM unique en attente. La configuration $C1b$ est une arborescence dont le nœud racine est en attente et dont l'état et le nombre de fils sont indifférents quelle que soit la liaison avec le nœud racine (conjonctive ou disjonctive). La racine peut être une UMM composée.

Lorsqu'un nœud racine est en attente, quels que soient l'état et le nombre de ses fils, le système doit agir en sorte de le rendre réussi. Ainsi, dans les deux cas, l'intention associée porte sur le nœud racine.

De manière générale, l'intention attachée à une UMM en attente est fonction de la direction de la communication. Si la machine croit que l'utilisateur connaît une information dont il a besoin pour accomplir ses différentes activités (application de la troisième loi), le système pose une question pour qu'ils puissent partager cette information — $Questionner(X)$. À l'opposé, si le système veut partager une information, il fera une proposition (application de la deuxième loi) — $Proposer(X)$.

Si la racine X est une UMM composée, elle s'écrit $X = W(T)$. On reconnaît à nouveau la configuration remarquable $C1b$. L'intention porte alors sur W. Ce schéma peut se reproduire autant de fois que cela est possible.

Classe $C2$

Cette classe regroupe les configurations $C2a$, $C2b$, et $C2c$. Il s'agit de trois configurations de hauteur 1 dont la racine est réussie. Dans $C2a$, il y a un seul fils en attente ; dans la disjonction $C2b$, il y a un fils en attente et un fils réussi ; dans la conjonction $C2c$, il y a un fils en attente et un fils accompli. Dans les trois cas, les fils peuvent être des UMM composées.

Lorsqu'un nœud racine est réussi, on ne peut rien dire de son état s'il possède un ou plusieurs fils en attente liés conjonctivement ou s'il possède au moins un fils en attente lié disjonctivement. L'intention de communication doit porter sur le fils $< Y >$ en attente.

Comme dans la classe $C1$, l'intention est, selon la direction de la communication, soit une question ($Questionner(Y)$) soit une proposition ($Proposer(Y)$). Cepen-

dant, cette intention est différente si les fils sont issus de regroupements horizontaux :

Si $Y = V \vee W \vee T \vee \ldots \vee R$ alors

$Questionner(Y) \Rightarrow Questionner(V \vee W \vee T \vee \ldots \vee R))$

Si $Y = V \wedge W \wedge T \wedge \ldots \wedge R)$ alors

$Questionner(Y) \Rightarrow Questionner(V \wedge W \wedge T \wedge \ldots \wedge R))$

Classe $C3$

Dans cette classe, nous avons quatre configurations remarquables $C3a$, $C3b$, $C3c$ et $C3d$. Il s'agit d'arborescences dont la racine est réussie alors qu'elle devrait être accomplie au regard des fils présents. Dans $C3a$, il y a une seule UMM réussie ; dans $C3b$, il y a un seul fils accompli ; dans la disjonction $C3c$, il y a un fils accompli et un ensemble de fils non-accomplis (en attente ou réussis) ; enfin, dans la conjonction $C3d$, il y a un fils réussi et un ensemble de fils non-réussis (en attente ou accomplis). Dans les autres cas, la racine peut être une UMM composée.

Il existe deux possibilités qu'un noeud racine soit réussi et non-accompli : soit cet état est inhérent à l'UMM, soit cet état provient de l'état de ses fils. Les configurations $C3a$, $C3b$ et $C3c$ sont du premier cas, tandis que la configuration $C3d$ est du second cas. L'intention de communication du système doit être d'informer l'utilisateur du non-accomplissement de la racine ($Indiquer(NA(X))$ qui se lit « *indiquer que X n'est pas accompli* »), avec, dans le second cas, la possibilité de justifier de cet état ($Indiquer(CAR(NA(X), NA(Y)))$ qui se lit « *indiquer que X n'est pas accompli car Y n'est pas accompli* »).

Si la racine $|X|$ est une UMM composée, elle provient d'un regroupement vertical et s'écrit $|X| = |W|\,(|T|)$. L'expression $NA(X)$ s'écrit alors $CAR(NA(W), NA(T))$. De même, si $|Y|$ est une UMM composée et s'écrit $|Y| = |U|\,(|V|)$ alors $NA(Y)$ s'écrit $CAR(NA(U), NA(V))$. Le contenu intentionnel se complexifie si les UMM composées sont issus de regroupements horizontaux. Par exemple, si $|V| = |A| \vee |B| \vee \ldots \vee |C|$ alors $NA(V)$ s'écrit $OU(NA(A), NA(B), \ldots, NA(C))$.

Classe $C4$

Dans cette dernière classe, nous avons les deux configurations $C4a$ et $C4b$ représentant les cas normaux où la racine est accomplie. Cela ne peut se produire que dans trois cas : soit il n'y a pas de fils (cas $C4a$), soit tous les fils sont accomplis (cas $C4a$ où $\underline{\overline{X}}$ est une UMM composée), soit au moins un fils est accompli dans le cas d'une arborescence disjonctive (cas $C4b$).

Dans tous ces cas, l'intention porte sur le nœud racine et doit être d'informer l'utilisateur de son état — $Indiquer(X)$. Si $\underline{\overline{X}}$ est une UMM composée, l'intention

continue de porter sur l'UMM la plus haute dans l'arborescence. Par exemple, si $\overline{X} = \overline{P}(\overline{Y} \wedge \overline{W}(\overline{U} \vee V))$ alors l'intention est $Indiquer(P)$.

3.4.3 Applications des configurations remarquables

3.4.3.1 Limites des configurations remarquables

Les intentions de communications sont produites en mettant en correspondance les arborescences actives en mémoire de travail soumises aux opérations de focalisation et les configurations remarquables que nous venons de décrire. Ce principe permet d'assurer un premier niveau de pertinence mais il possède certaines limites car, d'une part, les configurations de hauteur supérieure à 1 n'ont pas été déterminées, et d'autre part il ne permet pas de limiter le nombre d'intentions. D'autres critères devront donc être utilisés. À titre d'exemple, on examinera les trois arborescences suivantes : la première ne pose aucun problème ; les deux autres illustrent les limites mentionnées.

Exemple correct d'application des configurations remarquables

Dans l'arborescence (c) de la Fig. 3.17 qui est équivalente à l'arborescence (a) après l'application de deux opérations de focalisation ($|T| = |X|\,(|Y|)$ et $\overline{U} = \overline{Z}(\overline{V})$), on reconnaît la configuration $C2c$ qui s'applique aux UMM $\{|T|\,;\overline{U};W\}$: aucune autre configuration n'est reconnaissable. Ainsi, seule l'intention $Questionner(W)$ est générée ; cette arborescence ne pose donc pas de problème particulier.

Premier exemple incorrect d'application des configurations remarquables

À la Fig. 3.18, l'arborescence (b) est équivalente à l'arborescence (a) ($T = W \vee R$). On y reconnaît la configuration $C3d$ qui s'applique respectivement aux UMM $\{|X|\,;\overline{Y};|Z|\}$ et $\{|V|\,;T;\overline{S}\}$, et la configuration $C2b$ qui s'applique aux UMM $\{|Z|\,;U;|V|\}$. Les trois intentions sont donc normalement les trois suivantes :

$$Indiquer(CAR(NA(X), NA(Z)))\,;\,Questionner(U)\,;\,Indiquer(CAR(NA(V), NA(S)))$$

La première intention est générée intempestivement par rapport à la seconde et la pertinence de la dernière intention est discutable. En effet, premièrement, on ne peut rien dire du non-accomplissement de $|X|$ du fait du non-accomplissement de $|Z|$ puisque le fils U de $|Z|$ est en attente. Deuxièmement, dans ce cas précis, est-il pertinent que le système a l'intention de communiquer le non-accomplissement de $|V|$ du fait du non-accomplissement de $|S|$? Cela pourrait être interprété comme la justification de la question générée par la reconnaissance de la configuration $C2b$, mais cette justification est-elle pour autant nécessaire ? Les deux possibilités sont envisageables.

Second exemple incorrect d'application des configurations remarquables

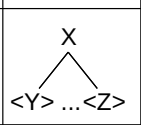
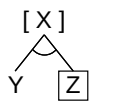
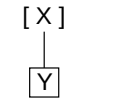
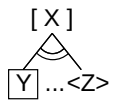
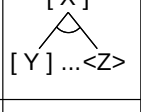
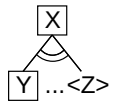
Classe		Notation graphique	Notation fonctionnelle	Configuration remarquable	Intention de communication
C1	C1a	X	X	X est en attente sans père et sans fils.	Questionner(X) ou Proposer(X)
	C1b		X (<W>+)	X est en attente. L'état et le nombre de fils sont indifférents.	
C2	C2a		|Y| (Y)	X est réussi avec un fils Y. Y est en attente.	Questionner(Y) ou Proposer(Y)
	C2b		|X| (Y v |Z|)	Dans une disjonction, la racine réussie possède deux fils : un fils en attente et un fils réussi.	
	C2c		|X| (Y ^ $\overline{Z}$)	Dans une conjonction, la racine réussie possède deux fils : un fils en attente et un fils accompli.	
C3	C3a	[X]	|X|	X est réussi sans père et sans fils.	Indiquer(NA(X))
	C3b		|Y| ($\overline{Y}$)	X est réussi avec un fils Y. Y est accompli.	
	C3c		|Y| ($\overline{Y}$ v <Z>+)	Dans une disjonction, X est réussi et un fils est accompli. Le nombre des autres fils non-accomplis est supérieur à un.	
	C3d		|X| (|Y| ^ <Z>+)	Dans une conjonction, X est réussi et un fils est réussi. Le nombre des autres fils non-réussis est supérieur à un.	Indiquer(CAR(NA(X),NA(Y)))
C4	C4a		$\overline{X}$	X est accompli sans père et sans fils.	Indiquer(X)
	C4b		$\overline{X}$ ($\overline{Y}$ v <Z>+)	Dans une disjonction, X est accompli et un fils est accompli. Le nombre des autres fils non-accomplis est supérieur à un.	

FIG. 3.16 – Les quatre classes d'équivalence.

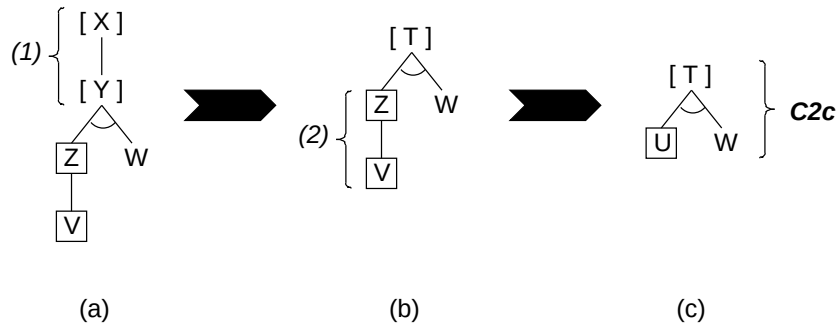


FIG. 3.17 – Exemple correct d'application des configurations.

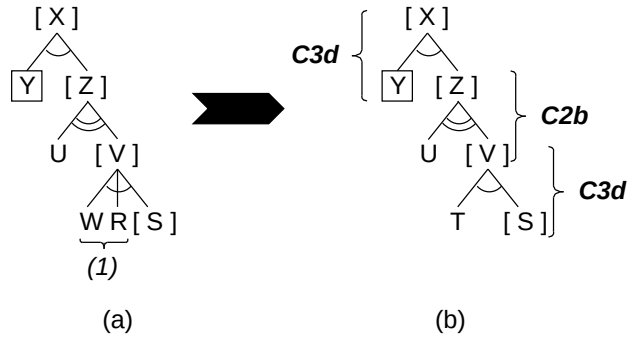


FIG. 3.18 – Exemple incorrect d'application des configurations.

Dans ce dernier exemple Fig. 3.19, l'arborescence (b) équivalente à l'arborescence (a) $(A = U \vee V \vee W \vee T)$ génère par reconnaissance des configurations remarquables $C2a$ sur $\{|Z|\,;A\}$ et $C2b$ sur $\{|X|\,;Y;|Z|\}$ les deux intentions suivantes :

$$Questionner(Y)\ ou\ Proposer(Y)$$
$$Questionner(U \vee V \vee W \vee T)\ ou\ Proposer(U \vee V \vee W \vee T)$$

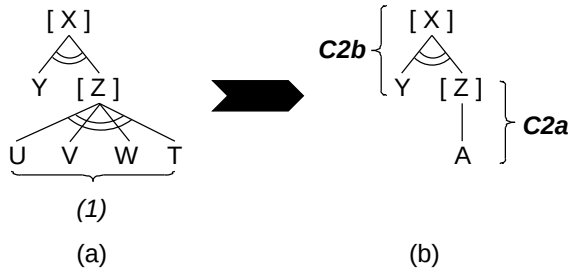


FIG. 3.19 – Exemple incorrect d'application des configurations.

Les problèmes posés sont ceux du nombre d'intentions générées et de la complexité des contenus intentionnels. Premièrement, deux intentions de même type peuvent être générées en même temps : peut-on poser deux questions ou faire deux propositions quand ces deux actes ne sont pas directement liés, et peut-on poser autant de question à la fois ? Cela présente encore

plus de difficultés lorsque les intentions de même type portent sur des UMM d'arborescences différentes. Deuxièmement, le contenu intentionnel de la seconde intention est trop important en quantité : un système informatique posant quatre alternatives à l'utilisateur peut-il s'attendre à être bien compris ? À l'écrit cela est certes possible, mais à l'oral cela dépend de l'adéquation entre le nombre d'alternatives et les capacités cognitives de l'utilisateur humain, sans parler des autres facteurs (états physiologiques, psychologiques, affectifs, etc.).

3.4.3.2 Les règles dialogiques

Nous venons de voir que les configurations remarquables n'assurent pas à elles seules la garantie de la pertinence des intentions communicatives du système de dialogue. Un second niveau doit être mis en place. La solution actuelle retenue consiste à effectuer un *filtrage* des intentions par une série de **règles dialogiques**. Il s'agit d'heuristiques qui **modulent** les trois lois comportementales à la base des configurations remarquables.

Cette solution comporte deux limites. Premièrement, comme la nature heuristique des règles l'indique, la pertinence des intentions résultant du filtrage n'est pas garantie dans tous les cas : il ne s'agit pas de règles absolues mais probabilistes. Deuxièmement, il faudrait alors étudier dans quels cas ces règles peuvent être transgressées.

Le filtrage s'effectue juste avant la génération proprement dite de l'énoncé du système. Comme la troisième loi est plus importante que la seconde, les intentions du type $Question(X)$ sont plus importantes que les intentions du type $Proposer(X)$ qui sont elles-mêmes plus importantes que les intentions du type $Indiquer(Z)$:

$$Questionner(X) > Proposer(Y) > Indiquer(Z)$$

En outre, une intention dont le contenu intentionnel a déjà été l'objet d'un même type d'intention est moins importante qu'une intention qui est générée pour la première fois. Ceci évite en particulier les répétitions en laissant le système insister lorsqu'il ne peut pas agir autrement. Les intentions sont donc d'abord classées les unes par rapport aux autres selon ces deux critères d'importance. En dernier recours, s'il existe plusieurs intentions de même niveau d'importance, une seule est choisie au hasard.

Pour l'instant nous avons établi les trois heuristiques suivantes :

- Il ne peut pas exister au même moment plusieurs intentions dont les contenus intentionnels comportent des UMM provenant d'une même arborescence, même si les contenus intentionnels n'ont pas d'UMM communes. Autrement dit, nous limitons à 1 le nombre d'intentions portant sur les UMM d'une arborescence.
- Il ne peut exister qu'une seule intention de même type à la fois — une seule question, une seule proposition, ou une seule indication — lorsque les intentions portent sur des UMM de différentes arborescences. Autrement dit, nous limitons à un le nombre de type d'intentions portant sur l'ensemble des UMM des arborescences actives.
- Le contenu intentionnel des questions et des propositions ne peut pas être composé de

plus de deux UMM. Par exemple, si l'intention initiale issue de la reconnaissance des configurations est $Questionner(U \vee V \vee W \vee Z \vee T)$ seule l'intention $Questionner(U \vee V)$ doit être prise en compte dans l'acte de langage final de la machine.

3.5 Application du modèle Génédic

Dans cette dernière section, nous allons montrer à travers l'analyse complète d'un exemple comment le modèle Génédic permet la co-construction du terrain commun à deux agents, l'un humain et l'autre artificiel, qui interagissent au cours d'un dialogue orienté par la tâche.

3.5.1 Le fonctionnement d'un dialogue selon la machine

Du point de vue de la machine, le fonctionnement des dialogues suit un cycle de deux phases qui se déroule tant que la machine génère des intentions de communication.

Les différentes activités de la machine — l'activité langagière, l'activité dialogique et l'activité applicative — interviennent au cours de ces deux phases. Ces activités modifient les arborescences à leur gré ; elles ont, en particulier, la faculté de pouvoir supprimer les arborescences ou certaines UMM jugées inutiles.

D'un point de vue fonctionnel, la séquentialité du cycle ne doit pas déboucher pour autant sur une architecture modulaire dans laquelle chaque phase est assurée par un module spécifique et le résultat d'un module est l'entrée du module suivant. Les trois phases prennent place au cœur de processus, certes différents, mais interdépendants dont le fonctionnement doit être opportuniste. Nous reviendrons sur ces aspects logiciels dans la seconde partie de la thèse (cf. Sect. 4.3 p. 100).

Les deux phases sont les suivantes :

Phase 1 Des intentions de communications sont générées par reconnaissance des configurations remarquables sur les arborescences en mémoire de travail. L'ensemble des intentions est filtré puis un énoncé généré. Celui-ci peut se composer d'une ou plusieurs phrases selon le nombre d'intentions finalement gardées. Cette phase est essentiellement assurée par l'activité dialogique de l'agent artificiel. La génération de la forme de surface des énoncés est assurée par l'activité langagière.

Phase 2 La réponse de l'utilisateur est analysée puis interprétée. L'interprétation consiste à comparer puis associer les éléments apparaissant dans l'analyse aux UMM des arborescences actives. L'état des UMM est modifié pour refléter le résultat des appariements éventuellement effectués.

L'interprétation se fait de façon discontinue : les arborescences ne sont pas développées entièrement en un seul coup mais au fur et à mesure de l'interprétation et du haut vers le bas. Une arborescence est développée, dans la mesure du possible, d'abord par l'action puis par les actants, et enfin les circonstants. Lorsqu'un

élément de l'analyse correspond à une UMM, celle-ci est développée sur un seul niveau à la fois. Ceci se reproduit jusqu'à ce qu'il n'y ait plus d'appariement entre les éléments présents dans l'analyse et les UMM actives.
Au cours de cette phase intervient en premier lieu l'activité langagière pour l'analyse puis, de façon partagée, l'activité dialogique pour l'interprétation, et l'activité applicative pour le développement des arborescences.

L'énoncé de l'utilisateur est analysé au début de la seconde phase. Si l'énoncé de l'utilisateur pose des problèmes d'analyse linguistique (du point de vue sémantique uniquement), la machine agit en sorte d'identifier le sens global de l'énoncé ou le sens des expressions inconnues. L'activité langagière crée donc une structure d'action adéquate à poser ensuite, lors de la première phase, une question sur l'énoncé entier ou sur les éléments non-connus de l'énoncé. Si le système est capable de poser une hypothèse d'analyse, la question portera sur cette hypothèse afin de la valider ou de la falsifier. Dans le cas contraire, la question portera sur le contenu intentionnel du dernier énoncé de la machine. Enfin, si le système fournit une analyse de l'énoncé (parce qu'il possède assez de connaissances linguistiques) mais que des éléments de l'analyse ne correspondent pas aux UMM des arborescences actives, ces éléments, à défaut d'être interprétés, sont tout de même convertis en UMM pour être éventuellement pris en compte ultérieurement. On illustrera ce dernier cas par l'exemple de début de dialogue du Tab. 3.3.

Mémoire de travail	Équivalence	Énoncé
REQUETE	REQUETE } *C1a*	M1 : Quelle est votre requête ?
		H1 : Je suis Thierry Lemeunier.
APPELANT (PRENOM, NOM) REQUETE	APPELANT PRENOM NOM } *C4a* REQUETE } *C1a*	M2 : Je sais que vous êtes Thierry Lemeunier, dites-moi quelle est votre requête ?

TAB. 3.3 – Exemple de répétition d'une intention.

Au début du dialogue, le système s'attend à une requête. Il pose une question sur cette attente (énoncé M1). Les éléments de l'analyse de l'énoncé H1 de l'utilisateur ne peuvent pas être associés aux UMM actives du système. Ce dernier les mémorise cependant sous la forme de l'arborescence $\overline{APPELANT}(\overline{PRENOM} \vee \overline{NOM})$. Les UMM sont accomplies car, d'une part, elles sont considérées comme faisant partie du terrain commun, et d'autre part elles sont référencées par le système. L'UMM $< REQUETE >$ reste en attente.

Les arborescences présentes maintenant en mémoire vont engendrer deux intentions. La moins importante porte sur l'UMM composée accomplie. L'acte généré à partir de cette intention (« *Je sais que vous êtes Thierry Lemeunier* ») permet de satisfaire l'acte H1. L'intention la plus importante porte sur l'UMM $< REQUETE >$ en attente. Elle est identique à l'intention initiale du début du dialogue, mais cette fois-ci un champ de l'UMM $< REQUETE >$ indique

qu'elle a déjà été l'objet d'une intention de même type. Cela a donc pour conséquence que le système donne un énoncé différent du premier en insistant sur sa demande (« *dites-moi...* »).

3.5.2 Exemple de dialogue

Il s'agit d'un exemple de mise en œuvre du principe de génération des intentions de communication dans le cas d'une compréhension partielle de l'énoncé. Cet exemple est tiré d'un début de dialogue avec le système AMI2, dans lequel le système ne comprend pas l'expression « *c'est de la part de* » de l'énoncé H2 de l'utilisateur (voir Tab. 3.4).

Premier temps : détermination de la requête d'action de l'utilisateur

Phase 1

Initialement la mémoire de travail ne comporte qu'une arborescence elle-même limitée à la seule UMM $< REQUETE >$ en attente. Ceci est dû au fait que le système AMI2 s'attend au début de chaque dialogue à une requête d'action sans préjuger de l'action en question. La configuration $C1a$ est reconnue à partir de cet unique élément actif en mémoire de travail. Une intention portant sur cette attente est donc créée : le système a l'intention de demander à l'utilisateur (troisième loi) s'il a une requête particulière dans le but de la connaître puis de la satisfaire (première loi). Cette intention produit l'énoncé M1.

Phase 2

La réponse H1 de l'utilisateur est interprétée comme une requête d'action : *l'utilisateur veut entrer en contact avec une tierce personne*. Cette interprétation en actes de M1 satisfait la demande de requête du système. Elle se fait de la manière suivante :

1. Le système associe la structure d'action correspondant à cette requête d'action en complétant l'UMM initiale $< REQUETE >$ par l'UMM $< PARLER - A >$ (l'action demandée correspondant à l'élément prédictif de l'analyse de H1), ainsi que, d'une part, l'UMM $< APPELE >$ — le premier actant obligatoire de l'action dont on connaît le prénom —, et, d'autre part, l'UMM $< APPELANT >$ — le second actant obligatoire de l'action. À ce stade, toutes les UMM sont encore en attente. Le système va alors les associer aux éléments de l'interprétation de l'acte H1.
2. L'UMM $< APPELE >$ devient réussie car le système s'y attend et elle correspond à un élément de monstration apparaissant dans l'analyse de H1.
3. Puisque l'UMM $< APPELE >$ est réussie, le système continue le développement de l'arborescence dans cette branche : les UMM $< PRENOM >$ et $< NOM >$ lui sont ajoutées.
4. L'UMM $< PRENOM >$ devient à son tour réussie (elle correspond à un élément de l'analyse) et aussitôt accomplie car elle a été catégorisée.

Mémoire de travail	Équivalence	Énoncé
REQUETE	REQUETE } C1a	M1 : Quelle est votre requête ?
		H1 : je voudrais parler à jérôme
		M2 : Quelle est votre identité ?
		H2 : c'est de la part de thierry Remarque : l'expression "c'est de la part de" n'est pas connue par le système.
		M3 : Je sais que thierry est un prénom. Lorsque vous dites "c'est de la part de", est-ce que vous vous présentez ?
		H3 : oui je suis thierry lemeunier
		...

TAB. 3.4 – Exemple de dialogue avec AMI2.

5. L'UMM $< APPELE >$ devient alors accomplie car elle est référencée par l'activité applicative du fait de l'accomplissement de son fils $< PRENOM >$.
6. L'UMM $< APPELANT >$ reste en attente car aucun élément de l'analyse de H1 ne lui correspond.
7. L'UMM $< PARLER - A >$ devient réussie car elle correspond à un élément de prédication de l'analyse de H1 mais elle n'est pas accomplie car l'UMM $< APPELANT >$ est en attente.
8. L'UMM $< REQUETE >$ qui était en attente est maintenant réussie (faisant partie du terrain commun) car l'acte M1 est réussi, mais elle n'est pas accomplie car l'action $< PARLER - A >$ n'est pas accomplie.

L'acte H1 est réussi mais ne peut pas être satisfait car l'état du monde ne le permet pas ; en effet, le système ne connaît pas l'identité de l'appelant.

Seconde temps : identification de l'appelant pour satisfaire la requête d'action

Phase 1

Le système doit maintenant satisfaire la requête d'action de l'utilisateur. De la structure et de l'état de l'arborescence maintenant en mémoire de travail est reconnue la configuration remarquable $C2c$. Le système a l'intention de demander à l'utilisateur (troisième loi) s'il peut lui donner son identité, information dont il a besoin pour satisfaire sa requête. Cela aboutit à l'énoncé M2.

Phase 2

L'analyse de la réponse H2 de l'utilisateur échoue en partie car l'expression « *c'est de la part de* » ne fait pas partie des connaissances linguistiques du système AMI2. L'activité langagière intervient donc pour créer une structure d'action adaptée à l'identification de l'expression inconnue. Comme dans ce cas précis elle n'est pas capable de poser une hypothèse d'analyse, elle va confronter cette expression au contenu intentionnel de son énoncé précédent, c'est-à-dire une identification de la personne qui appelle. L'arborescence créée est donc :

$$IDENTIFIER(SOURCE(|EXPRESSION|) \wedge CIBLE(ACTION(SE - PRESENTER)))$$

En outre, le système reconnaît « *thierry* » comme un prénom. Cet élément d'analyse ne correspond à aucune UMM non-catégorisée active. Une UMM est donc créée pour le mémoriser.

Phase 1

Des arborescences en mémoire sont générées trois intentions de communication. Il y a une question portant sur l'UMM $< APPELANT >$ qui a déjà été générée précédemment, une question portant sur l'UMM $< IDENTIFIER >$ puis une indication portant sur l'UMM $< PRENOM >$. Sur les deux premières intentions, seule la seconde est gardée car elle est

plus importante que la première, du fait qu'elle n'a pas encore été générée. Cela aboutit à l'énoncé M3.

Phase 2

L'énoncé H3 est analysé sans problème. Il comporte deux parties : une réponse positive à la question M3 du système et une présentation de l'appelant. La confirmation ne correspond à aucune UMM active. Elle est directement prise en compte par l'activité langagière qui modifie ses connaissances en conséquence. De plus, elle supprime la structure de l'action d'identification de l'expression inconnue car elle ne sert plus et risque d'engendrer des intentions sans intérêt.

L'énoncé H2 est de nouveau analysé et cette fois bien interprété :

1. L'UMM $< APPELANT >$ devient réussie car elle correspond à un élément de monstration de l'analyse de l'énoncé H2.
2. L'activité applicative continue de développer l'arborescence : les UMM $< NOM >$ et $< PRENOM >$ sont ajoutées à l'UMM $< APPELANT >$ réussie.
3. L'UMM $< PRENOM >$ devient à son tour réussie (elle correspond à un élément de l'analyse) puis aussitôt accomplie car elle est catégorisée. L'UMM $< PRENOM >$ accomplie qui était déjà présente est supprimée de la mémoire car ces deux UMM sont redondantes.
4. En conséquence de l'accomplissement de l'UMM $< PRENOM >$, l'UMM $< APPELANT >$ devient accomplie car elle est référencée par l'activité applicative.
5. La structure d'action créée par l'activité dialogique à partir des éléments de l'analyse de « *je suis thierry lemeunier* » est aussitôt détruite par l'activité applicative car elle ne lui sert plus à rien (l'identité de l'appelant est déjà connue).

3.6 Conclusion

Le modèle Génédic que nous avons présenté en détail dans ce chapitre est un modèle simple des intentions de communications pour une machine. Il est fondé sur l'idée que, pour interagir, le système et l'utilisateur doivent construire un espace de référenciation (le terrain commun) dont chacun se fait sa propre représentation dans une mémoire interactionnelle (mémoire de travail spécialisée pour l'occasion du dialogue). Les intentions de communication proviennent de l'état et de la structuration des éléments mnésiques actifs.

Ce modèle est issu d'une démarche de travail pragmatique : nous avons commencé par développer des systèmes simples ; l'examen de ces systèmes a alimenté notre réflexion ; celle-ci a abouti au modèle Génédic. Ce cycle doit être poursuivi. Nous avons donc développé une nouvelle version de notre système initial implémentant notre modèle des intentions. Cette démarche de développement et ce nouveau système sont l'objet du prochain chapitre.

Chapitre 4

Le projet AMI

Le modèle Génédic présenté dans le chapitre précédent est mis en œuvre dans l'application AMI. Notre objectif n'est pas de développer un système opérationnel exploitable à l'usage du public, mais d'implanter un prototype afin, d'une part, de l'expérimenter et d'obtenir des corpus de dialogue homme-machine, et d'autre part, de mettre en œuvre nos modèles et de tester leurs validités.

Une première version (AMI1) est fondée exclusivement sur le modèle COALA de Lehuen (présenté à la Sect. 2.4 p. 37). Une seconde version (AMI2) a ensuite été développée pour mettre en œuvre le modèle Génédic. Celle-ci est en cours d'expérimentation.

Les sections de ce chapitre présentent, dans l'ordre, la méthode de développement, l'architecture logicielle, et le fonctionnement du système AMI2, en détaillant ses trois activités : l'activité applicative, puis l'activité langagière, et enfin l'activité dialogique.

4.1 Présentation générale

4.1.1 L'application AMI

Nous avons développé deux versions de l'application. Dans AMI1, la première version de l'application qui est uniquement accessible *in situ*, le système simule la fonction de standardiste téléphonique d'un organisme quelconque, en l'occurrence nous avons pris l'IC2[1] pour faciliter les tests du système (Fig. 4.1).

L'application AMI1 consiste, d'une part, à mettre en relation une personne appelante et une personne appelée (de façon virtuelle), et d'autre part, à mettre en œuvre un serveur de messagerie écrite. Dans AMI2, la seconde version de l'application accessible à distance, le système ne simule que la partie serveur de messagerie afin d'obtenir une situation d'interaction plus réaliste. Le dialogue Fig. 4.2 est un exemple fictif d'interaction telle que nous souhaitons pouvoir en mener avec le système AMI2.

[1] L'IC2 (Institut d'Informatique Claude Chappe) de l'Université du Maine regroupe les trois entités suivantes : le LIUM, l'IUP MIME et le département d'informatique de la faculté des sciences.

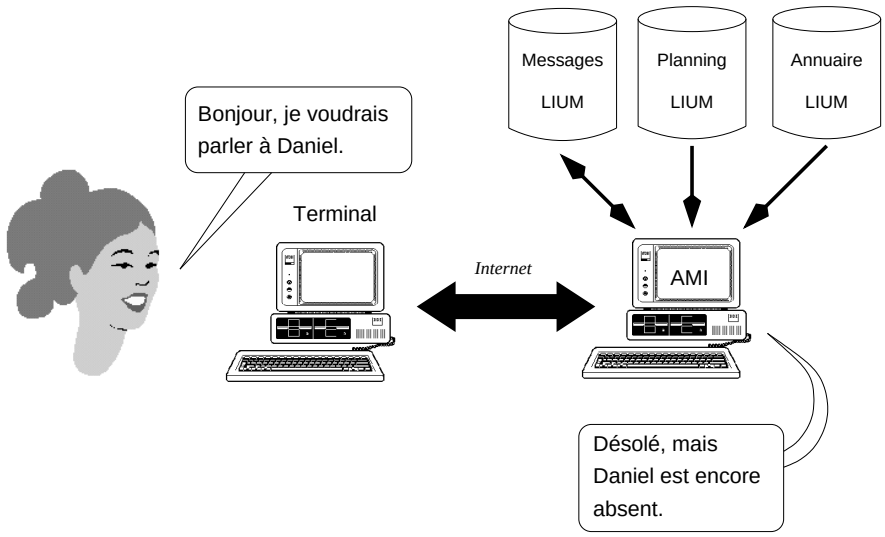


FIG. 4.1 – Schéma de l'application AMI.

M1 : Service de messagerie, bonjour. Que puis-je faire pour vous ?
H1 : C'est thierry, est-ce que j'ai des messages ?
M2 : Non, vous n'avez pas de message en attente.
H2 : Je voudrais laisser un message pour Daniel et Jérôme
M3 : Quel est votre message ?
H3 : Dites-leur que je serais en retard à la réunion de jeudi.
M4 : Votre message est enregistré.
Que voulez-vous faire maintenant ?
H4 : Rien de plus
M5 : Nous vous remercionns. Au revoir.

FIG. 4.2 – Un dialogue fictif avec AMI.

Cette application a été retenue pour deux raisons en particulier. La première est que, comparée à d'autres applications telle que l'assistance à la conception par ordinateur par exemple, la complexité n'en est pas importante. Ceci permet de nous concentrer en premier lieu sur la programmation de notre modèle. La seconde raison est qu'il s'agit d'une application très répandue dans la communauté française de recherche en DHM. Nous pouvons ainsi comparer plus facilement les divers modèles utilisés et les résultats obtenus. Dans la première partie de la thèse, nous avons souvent fait référence au système STANDIA du LIMSI [Grau *et al.*, 1994], aux travaux de Balkanski du LIMSI également [Balkanski & Hurault-Plantet, 1997] et aux travaux en communication parlée de Guyomard, Nerzic et Siroux à l'IRISA de Rennes [Guyomard *et al.*, 1995].

4.1.2 Interfaces du système AMI

La première version du système a été écrite pour être utilisée localement, tandis que la seconde version a été écrite pour être accessible à distance, via Internet. Les deux versions sont écrites en CLOS (*Common Lisp Object System*). *Common Lisp* est la version ANSI X3J13[2] du langage LISP [Steele, 1990].

Première version

AMI1 a été écrite sous Allegro CL for Windows V3.0.2[3]. Nous avons choisi cet environnement pour sa possibilité de construire des interfaces graphiques, ce qui est plus convivial que le mode texte. L'interface graphique sous Windows 95® de AMI1 (cf. Fig. 4.3) permet une visualisation des traces du fonctionnement (cf. Fig. 4.4) et un accès en lecture aux objets internes (cf. Fig. 4.5).

Seconde version

La seconde version est accessible à distance via le réseau Internet et un navigateur HTML (*Hypertext Markup Language*). L'utilisateur lance l'application à partir du site AMIWeb[4] (cf. Fig. 4.6). Un exemple d'interaction est donné Fig. 4.7.

Ce site permet également de remplir un questionnaire d'évaluation (Sec. 5.3 p. 124), de consulter les statistiques sur les réponses au questionnaire, et de visualiser le corpus obtenu. Actuellement, il n'est pas accessible de l'extérieur du bâtiment de l'IC2. De plus, une seule personne peut être connectée en même temps.

[2] *American National Standard Institute*, comité X3, sous-comité J13.

[3] Allegro CL est un environnement de programmation commercial en Common Lisp, avec la possibilité de construire rapidement des interfaces graphiques classiques. L'adresse du site Web est `http://www.franz.com`. Une version limitée est disponible gratuitement.

[4] L'adresse du site est `http://www-ic2.univ-lemans.fr:8081/`.

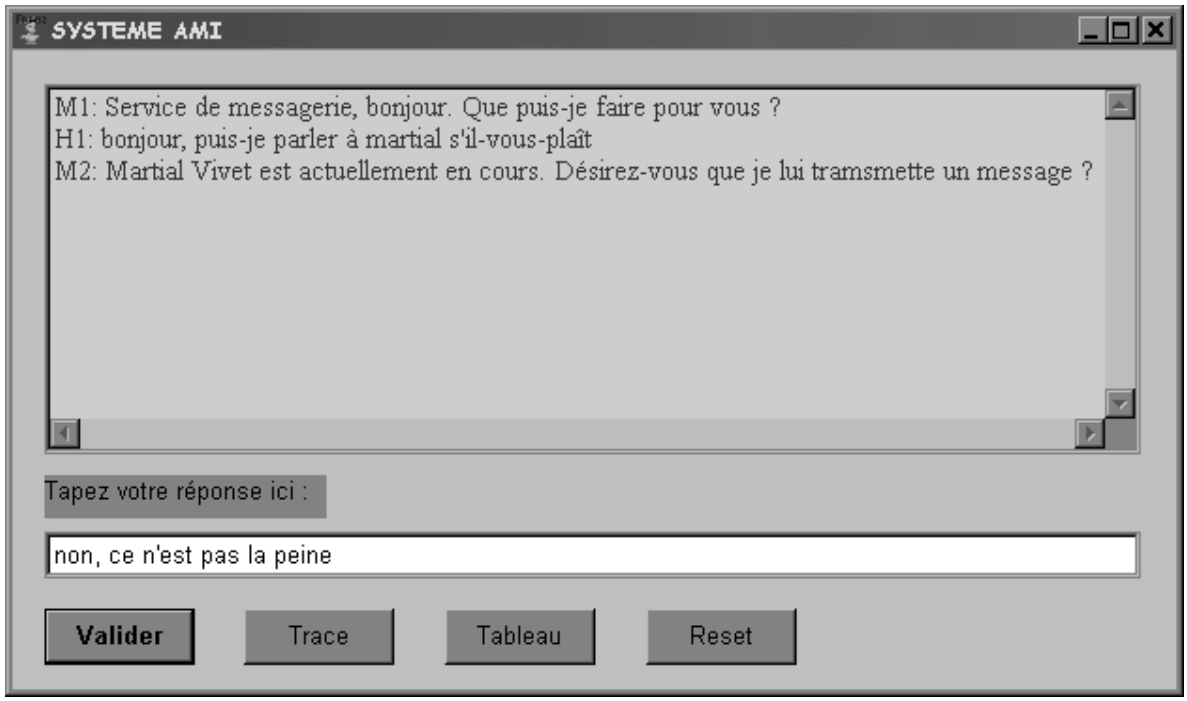


FIG. 4.3 – Interface de AMI1.

FIG. 4.4 – Fenêtre de traces de AMI1.

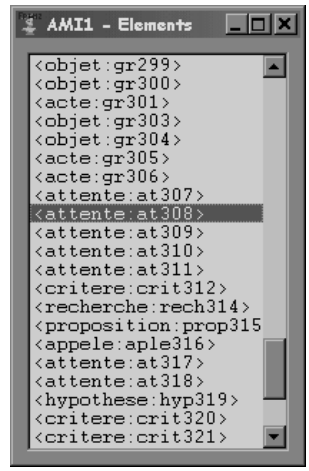


FIG. 4.5 – Accès en lecture aux objets internes de AMI1.

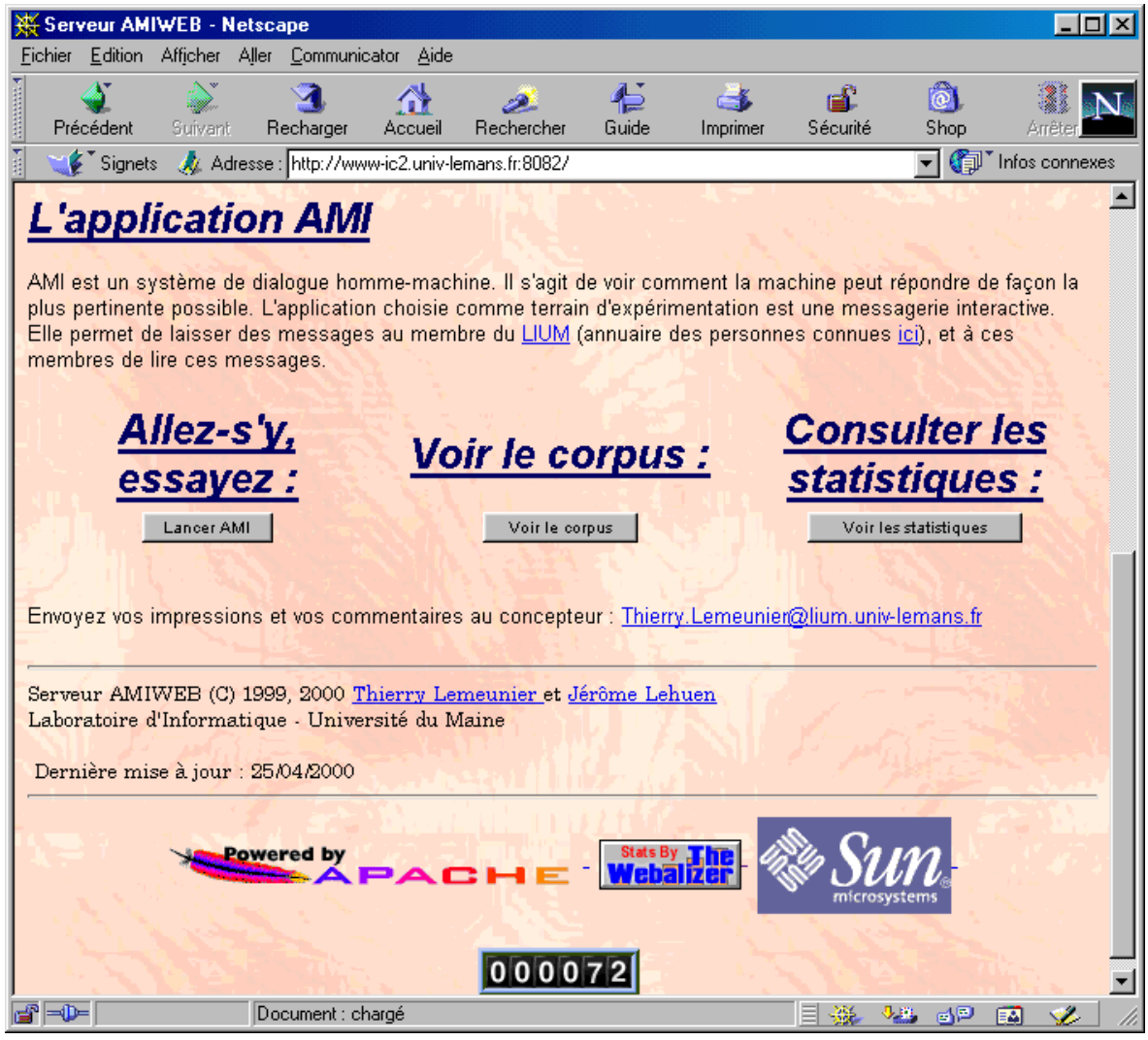


FIG. 4.6 – Image du site AMIWeb.

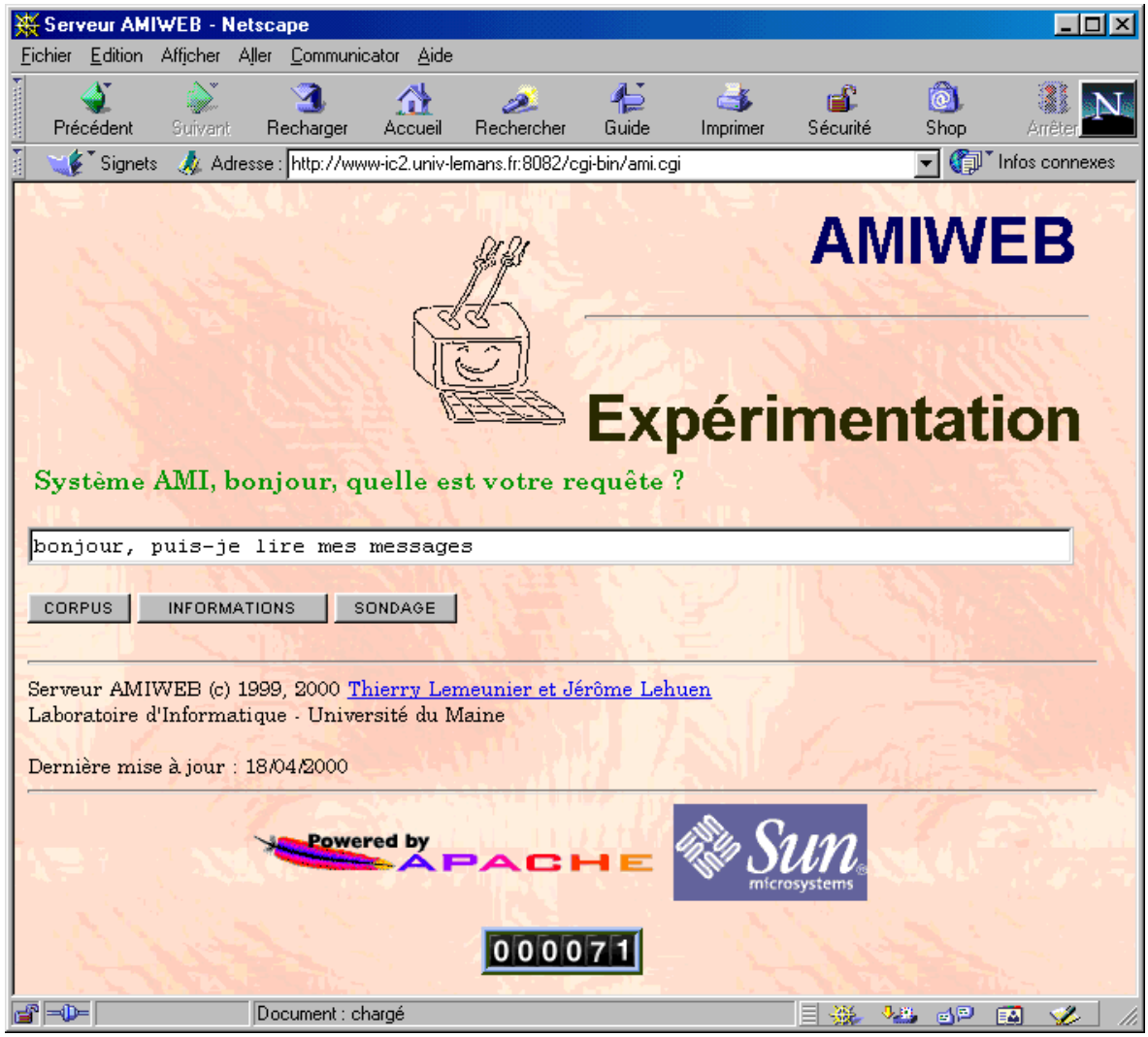


FIG. 4.7 – Interaction avec le serveur AMIWeb.

Le système AMI2 a été développé avec CLisp[5], une implémentation de Common Lisp, et CLIPS[6], un générateur de systèmes experts. Ce dernier a été utilisé pour écrire l'analyseur linguistique des énoncés de l'utilisateur. L'analyseur initialement écrit en CLOS était en effet trop lent pour une mise à distance.

La partie serveur est assurée par une CGI (*Common Gateway Interface*) écrite en langage ANSI C. Les différentes parties communiquent par fichiers. La figure 4.8 présente l'architecture physique du système AMI2.

4.2 Méthode de développement

Dans cette section, nous présentons tout d'abord la méthode de développement adoptée pour le projet AMI, puis nous nous positionnons par rapport aux différentes méthodes de génie logiciel.

4.2.1 Principe de la méthode

La méthode de développement adoptée pour le projet AMI est itérative [Lemeunier, 1999]. Face à la complexité des phénomènes langagiers et dialogiques sur lesquels nous travaillons, nous avons adopté une démarche expérimentale et pragmatique (dans le premier sens du terme). En procédant de la sorte, nous résolvons les problèmes au fur et à mesure qu'ils apparaissent. À la fin de chaque étape, le système doit être capable de fonctionner comme l'exige cette étape de développement. Ainsi, nous n'avons pas cherché à développer en une seule fois un système complet, dont nous savons par ailleurs que les différentes parties ne sont pas autonomes [Pierrel, 1987].

Cette méthode de développement a un inconvénient assez important en terme de développement logiciel : le système étant développé dans son ensemble, des modifications d'une partie doivent être répercutées dans les autres parties. Nous avons donc dû faire face à d'importants développements logiciels.

La méthode que nous avons adoptée n'est pas nouvelle en soi, mais elle était jusqu'à récemment peu utilisée en DHM. Nous nous sommes fortement inspiré de la méthode présentée dans la thèse de Rouillard [Rouillard, 2000] [Rouillard & Caelen, 1998]. Cette méthode consiste à développer successivement plusieurs versions d'un système qui est amélioré au fur et à

[5] CLisp est une implémentation presque complète de la seconde version du langage Common Lisp. Il a pour avantages d'être distribué sous les termes de la GNU-GPL et sous une multitude de plates-formes matérielles et logicielles (de l'Amiga aux systèmes Unix), mais pour désavantage d'être assez lent même après compilation du code source en code binaire. Le site officiel est `http://clisp.cons.org/~haible/clisp.html`.

[6] CLIPS est un générateur de système expert écrit par la NASA. C'est un logiciel complet de développement permettant la représentation des connaissances sous forme de règles d'inférences. On peut programmer des algorithmes selon les paradigmes objet et fonctionnel. La syntaxe est inspirée de celle de LISP. Ce logiciel est libre de droit et d'accès. Écrit en langage C standard, il a été porté sous différents systèmes d'exploitation (notamment Windows, MacOS et les systèmes Unix). Il est accessible à l'adresse Web : `http://www.ghgcorp.com/clips/CLIPS.html`.

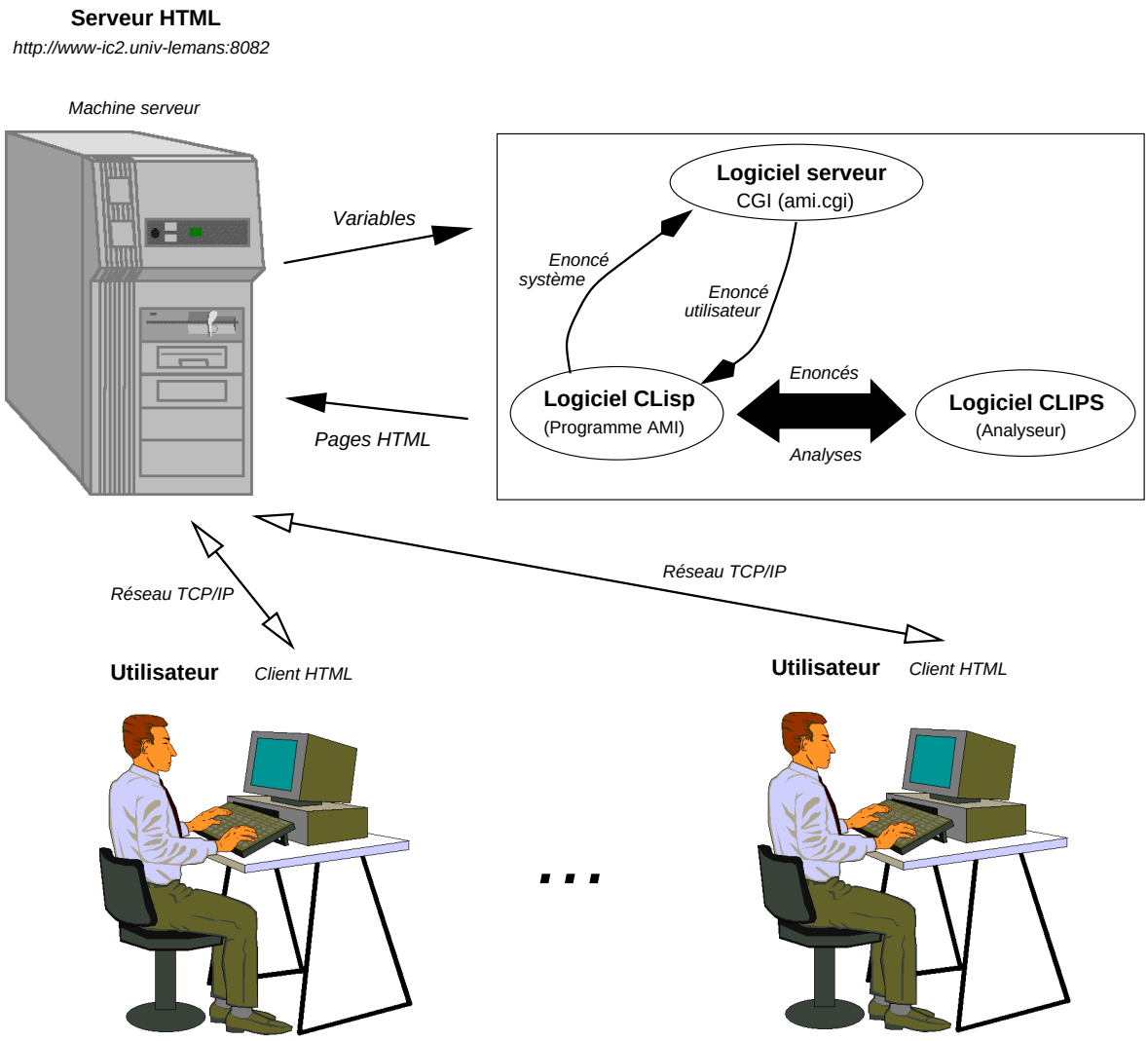


FIG. 4.8 – Architecture matérielle du système AMI2.

mesure, en prenant en compte les résultats de l'expérimentation de la version n pour développer la version $n+1$. Ainsi, un cycle élémentaire (Fig. 4.9) est composé de deux objets, un corpus et un système, sur lesquels sont appliquées trois procédures : analyse du corpus n, développement du système $n+1$, expérimentation de ce système pour obtenir un corpus $n+1$. On peut alors enchaîner les cycles élémentaires autant de fois que cela est nécessaire.

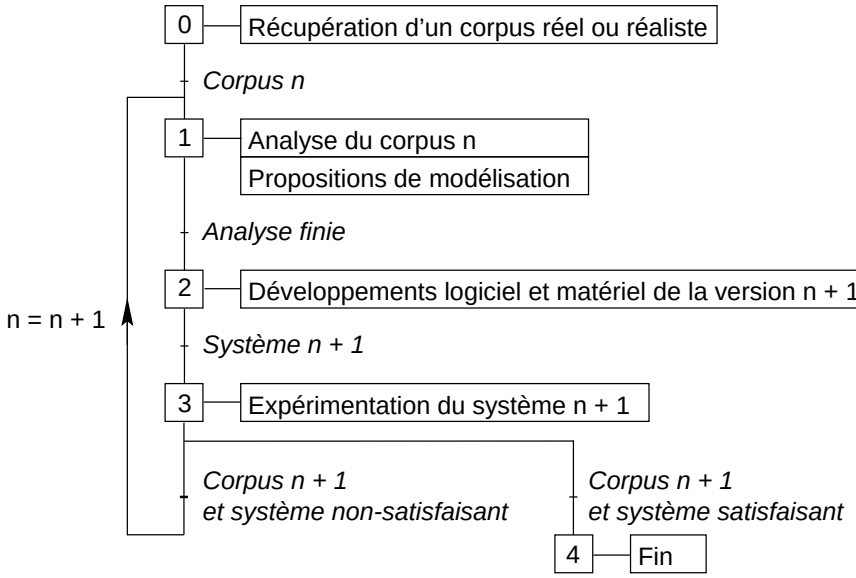


FIG. 4.9 – Le cycle élémentaire de la méthode de conception.

Pour amorcer le premier cycle de développement, il faut obtenir au préalable un premier corpus qui provient donc soit de la transcription de dialogues homme-homme authentiques, soit de la transcription de dialogues homme-machine obtenus par un protocole du type Magicien d'Oz.

Cette dernière méthode est bien connue. Elle permet d'obtenir des corpus de dialogues ou d'interactions entre une machine et l'utilisateur. Plus précisément, il s'agit de faire simuler les comportements de la machine par un compère humain à l'insu de l'utilisateur. Ce dernier pense donc dialoguer avec une machine alors qu'il n'en est rien. Sans l'existence de système de dialogue opérationnel, cette méthode est nécessaire, d'une part pour étudier les spécificités des interactions homme-machine, bien qu'il soit clair qu'elles n'en sont pas vraiment, et d'autre part pour amorcer le développement d'un système informatique qui prendra la place du compère humain.

Cette méthode de recueil est cependant critiquable et Vilnat rappelle les biais inhérents à cette méthode [Vilnat, 1997]. Premièrement, le compère ne peut pas suivre exactement le comportement d'une machine. Il est plus lent dans les recherches d'informations, et il lui est difficile de simuler une capacité de compréhension limitée, sauf s'il s'agit d'un bon « acteur » ! Plus précisément, nous ajouterons que le compère simulant la machine peut avoir du mal à se positionner par rapport aux capacités du système : il peut être en deçà ou au contraire

au-delà des capacités visées ou réalisables du système qu'il remplace. Deuxièmement, le comportement de l'utilisateur sera également différent selon qu'il s'agit de « vrais » utilisateurs ou de « faux » utilisateurs. Les vrais utilisateurs ne savent pas qu'ils sont des sujets d'observations. En revanche, les faux utilisateurs ont reçu une consigne à respecter. Ils ne sont pas là pour se servir réellement du système, mais pour « jouer un rôle ». Leur comportement est nécessairement différent. Notamment, ils seront tentés de tester les limites du système (cf. Sec. 2.4.2.1), ou bien ils manqueront de motivation [Pernel, 1994, p. 44]. Pour conclure sur ce point, nous admettrons qu'en DHM les corpus obtenus avec la méthode du magicien d'Oz sont nécessaires afin de représenter non pas une réalité interactionnelle à étudier mais plutôt un objectif inaccessible à atteindre[7], une sorte de limite supérieure.

Pour développer le système AMI, nous sommes parti du modèle de dialogue proposé par Lehuen [Lehuen, 1997b]. Nous avons repris l'expertise dialogique développée et nous avons programmé l'expertise du domaine d'application. Nous sommes ainsi passé de la simulation d'un poste de demande de renseignements bibliographiques à la simulation d'un standard téléphonique. La modélisation de la nouvelle tâche a été faite en nous fondant sur l'analyse d'un corpus de dialogues réels homme-homme enregistrés dans un standard téléphonique et transcrits dans [Castaing, 1993].

La méthode que nous employons permet de répondre de façon détournée au problème de la circularité de l'obtention des corpus de DHM réels [Rouillard, 1998]. Le problème est le suivant : en s'interdisant d'utiliser la méthode du Magicien d'Oz du fait des nombreux biais inhérents à cette méthode, comment développer des modèles du DHM en se fondant sur des corpus réels lorsqu'il n'existe aucun système de DHM opérationnel pour les obtenir, justement parce que nous manquons de matériaux linguistiques et conversationnels pour les développer. Avec une méthode itérative, le problème n'est pas résolu directement mais contourné. Pendant un certain temps nous acceptons de travailler sur des données peu fiables, puisqu'elles ont été obtenues dans des conditions où le système expérimenté est très rudimentaire. Cependant, ces premières expérimentations suffisent tout de même à améliorer le système, dont on finira par obtenir une opérationnalisation acceptable, après deux ou trois cycles de développement seulement.

4.2.2 Positionnement de la méthode de conception

En génie logiciel, il existe au moins quatre méthodologies de développement d'un projet informatique [Gaudel *et al.*, 1996]. La plus ancienne méthode est la méthode *de la cascade*. Elle consiste à convenir d'un nombre précis d'étapes qu'il faut atteindre successivement. L'étape $n+1$ n'est commencée que si l'étape n est validée. Si une étape n'est pas validée seule l'étape immédiatement précédente est remise en cause.

[7] Nous avons repris ici une idée présentée dans le Rapport d'Activité de l'INRIA de Nancy de 1998, Thème 3A, Projet Langue et Dialogue [INRIA, 1998].

La seconde méthode est celle dite *en V*. Dans cette méthode, les premières étapes de conception (spécification du logiciel) doivent préparer les dernières étapes (validation et vérification). Le principe est que toute description d'un composant est accompagnée des tests permettant de s'assurer qu'il correspond à sa description. Par exemple, la conception architecturale doit permettre de décrire complètement le protocole d'intégration et les jeux de test d'intégration.

La troisième méthode est celle dite *en spirale*. Il s'agit d'une méthode cyclique en quatre phases :

1. Détermination des objectifs du cycle.
2. Analyse des risques, maquettage.
3. Développement et vérification de la solution retenue.
4. Analyse des résultats et planification du cycle suivant.

Des prototypes sont ainsi développés, testés puis éventuellement modifiés. Cette méthode est bien adaptée à des projets à risques ou innovants.

Enfin, la dernière méthode de conception est le modèle *par incréments*. Elle est utilisée pour les projets importants. Elle consiste à développer un logiciel noyau puis à y intégrer des modules développés en parallèle. Chaque incrément peut être développé selon l'une des méthodes précédentes.

La méthode que nous avons utilisée est proche de la méthode en spirale[8] (cf. Fig. 4.9). Cette méthode est intéressante pour les chercheurs en dialogue homme-machine car elle permet de prendre en compte les attentes des utilisateurs. Contrairement aux autres méthodes, l'analyse des besoins n'est pas figée lors de la première étape de développement, mais progressivement affinée au fur et à mesure des cycles de développement. Les utilisateurs sont confrontés le plus tôt possible aux prototypes développés. Ils collaborent tout au long du développement du logiciel, y compris, tout au moins idéalement, dans la rédaction des notices d'emplois. Ainsi, développer une application en restant au plus près d'une utilisation réelle de par les expérimentations successives permet de mieux tenir compte des usages de cette application.

Dans le cadre de cette thèse, nous avons développé deux versions de l'application AMI. L'expérimentation de AMI1 est décrite à la section 2.4.2.1 p. 41. L'analyse du corpus et des traces du fonctionnement du système nous a amené à réfléchir à une modélisation des intentions communicatives. Cette dernière a été mise en œuvre dans la seconde version de l'application AMI (section 4.4.1 p. 109).

[8] Cette méthode a été adoptée dans de nombreux domaines de recherches. Par exemple en ingénierie des connaissances [Trichet, 1998] ou dans l'enseignement assisté par ordinateur [Bruillard & Vivet, 1994].

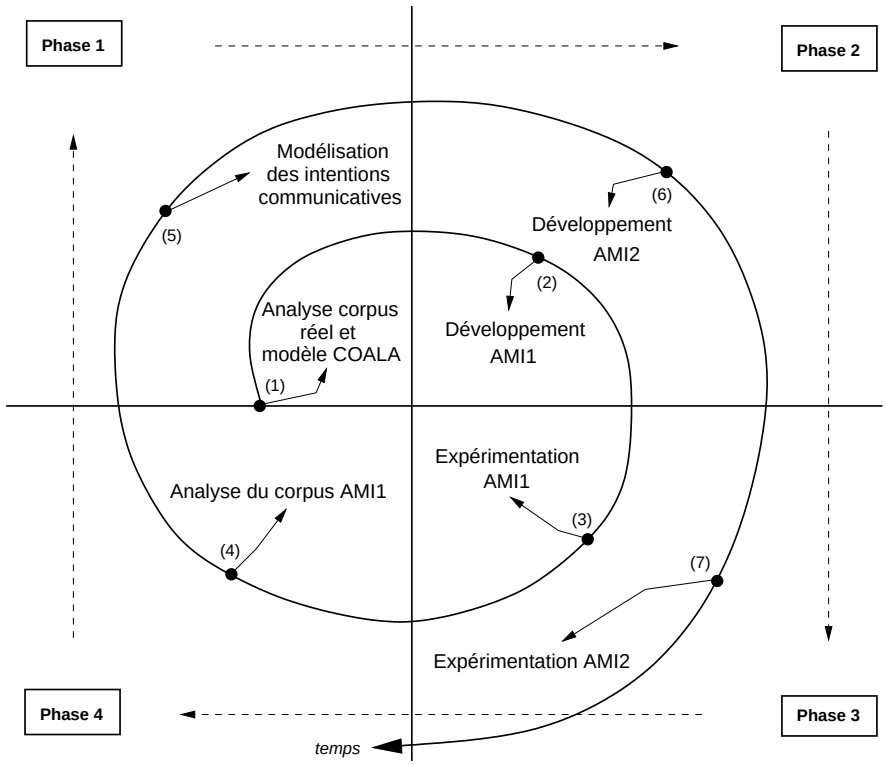


FIG. 4.10 – Développement en spirale du projet AMI.

4.3 Architecture logicielle du projet AMI

La conception d'un agent logiciel effectuant plusieurs activités nous a conduit naturellement à nous tourner vers une architecture modulaire, dans laquelle il y aurait autant de modules que d'activités différentes. Nous nous situons de fait dans le paradigme de l'intelligence artificielle distribuée (IAD).

Dans cette section, nous allons commencer par un bref rappel des deux principales architectures du paradigme de l'IAD et ensuite de ce qui les différencie l'une de l'autre. Nous discuterons ensuite de l'architecture la plus adéquate pour les systèmes de dialogue avant de présenter l'architecture retenue pour l'application AMI.

4.3.1 Contrôle distribué *versus* contrôle centralisé

Les apports de l'IAD sont bien connus [Ferber & Ghallab, 1988] [Haton *et al.*, 1991]. La modularité permet une plus grande souplesse de développement et une facilité de modification des connaissances, qui sont alors indépendantes les uns des autres, ce qui permet la réutilisabilité des modules. Elle réduit la complexité d'une tâche par sa décomposition en sous-tâches plus simples. Ainsi, l'efficacité, la fiabilité et la sûreté du fonctionnement global du système sont accrues. Cependant, ce type d'architecture pose des problèmes liés à la communication

entre les modules et au contrôle du fonctionnement [Bachimont, 1994]. Les systèmes à contrôle centralisé ont une architecture de tableau noir tandis que ceux à contrôle distribué ont une architecture multi-agents. Les systèmes du premier type sont moins concernés par les problèmes de communication que par les problèmes de contrôle, tandis que cela s'inverse pour les systèmes du second type.

4.3.1.1 Rappel sur l'architecture de tableau noir

Une architecture de tableau noir est ainsi appelée car son fonctionnement est analogue à celui de la résolution collective d'un problème par une classe d'écoliers. L'exemple généralement pris est celui de la construction d'un puzzle. Les écoliers n'ont pas le droit de communiquer entre eux, mais observent sur le tableau les pièces déjà posées. Dès qu'un élève s'aperçoit qu'il peut poser une pièce, c'est-à-dire contribuer à la résolution du problème, il lève le doigt. Le professeur (qui assure le contrôle) choisit parmi les doigts levés qui doit aller poser sa pièce au tableau. La somme des contributions individuelles permet finalement de terminer le puzzle.

Dans la terminologie du tableau noir, les élèves sont appelés des **sources de connaissances** (notées KS pour *Knowledge Source*). Chaque KS a le droit de créer et de modifier les données présentes sur le tableau noir. Lorsqu'une KS peut s'activer (c'est-à-dire que ses conditions d'activations sont vraies), un **KSAR**[9] correspondant à cette possibilité d'activation est créé et placé dans **l'agenda** du tableau. Le contrôle consiste alors à choisir parmi les KSAR de l'agenda celui qui sera activé.

La figure 4.11 illustre le fonctionnement d'un tableau noir. Il s'agit d'un fonctionnement séquentiel — il n'y a qu'une seule KS active à un instant donné —, et opportuniste — on ne sait pas d'avance quelle KS sera active à un instant donné. Pour un exemple d'utilisation du tableau noir le lecteur pourra se reporter à la thèse de Bricon-Souf sur la compréhension des énoncés en langue naturelle [Bricon-Souf, 1994].

Le cycle de fonctionnement d'un tableau noir est assez simple dans son principe :

(1) Selon les éléments présents dans le tableau noir, si une KS peut être activée, le contrôleur crée un KSAR correspondant dans l'agenda. Cette étape est répétée autant de fois qu'il y a de KS dont les conditions d'activation sont réalisées.

(2) Les KSAR de l'agenda sont ordonnés selon différentes stratégies et selon le type de contrôle. Le KSAR correspondant le plus à la stratégie de sélection est finalement choisi.

(3) La KS du KSAR choisi à l'étape précédente est déclenchée. Ceci a généralement pour conséquence de modifier le tableau noir par ajouts de nouveaux éléments, destructions de certains éléments, ou modifications d'éléments déjà présents. Le cycle reprend à la première étape et le fonctionnement continue tant qu'il y a des KSAR dans l'agenda.

[9] KSAR est un acronyme de *Knowledge Source Activation Record*.

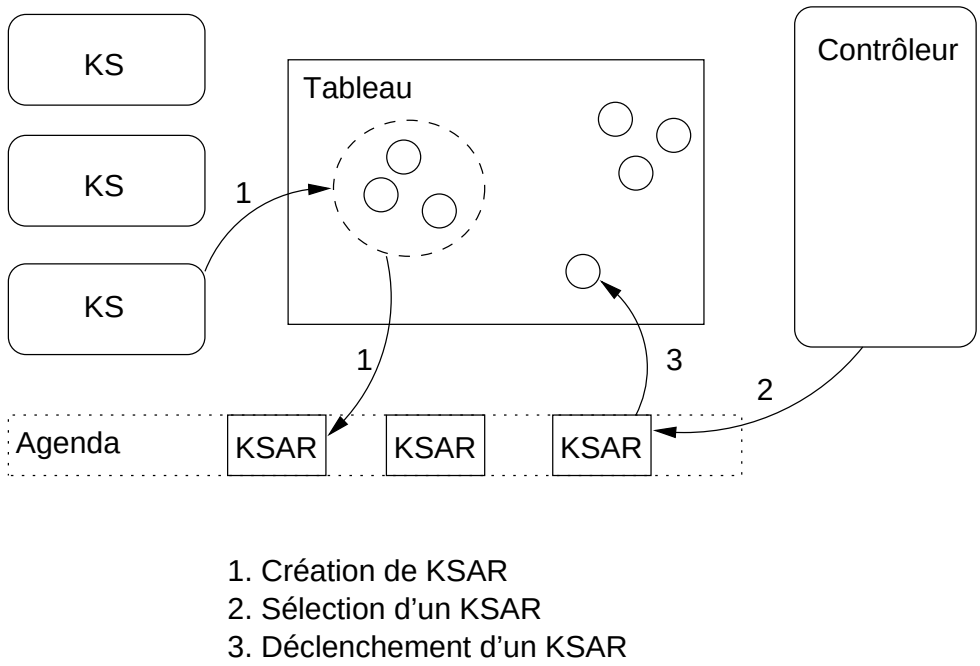


FIG. 4.11 – Cycle de base d'un tableau noir d'après [Bachimont, 1994].

4.3.1.2 Rappel sur l'architecture multi-agents

Dans une architecture multi-agents la résolution du problème est faite par les interactions d'agents autonomes. Les agents peuvent être plus au moins « intelligents ». Ils perçoivent partiellement ou totalement l'environnement, c'est-à-dire les autres agents et les objets visibles possédant un certain nombre de propriétés. Selon leurs capacités cognitives, ils peuvent avoir des buts simples ou complexes. Pour atteindre ces buts ils peuvent communiquer entre eux ou agir directement sur les objets. Une solution unique au problème est trouvée lorsque l'environnement est stabilisé. On parle dans ce cas d'un phénomène d'émergence. La figure 4.12 est un exemple d'une architecture multi-agents assez simple. Son fonctionnement s'apparente à la simulation d'un micro-monde dans lequel les agents agissent en parallèle. Les systèmes multi-agents sont très bien appropriés à l'étude des systèmes sociaux (voir par exemple [Durand, 1996] pour une application en épidémiologie animale).

Dans notre exemple, nous avons trois agents, nommés A1, A2 et A3. L'agent A1 est constructeur de briques. Il est capable d'en construire à partir de l'argile qu'il récupère en se déplaçant dans l'environnement. Il peut recevoir des commandes de construction de l'agent A3 et donner des informations à l'agent A2 sur la position des briques qu'il a construites. L'agent A2 convoyeur de briques est capable de se déplacer à un certain endroit pour chercher des briques selon les informations de l'agent A1. Il entasse les briques à un endroit précis dont il informe la position à l'agent A3. Enfin, l'agent A3 est un constructeur de mur. Il est capable à partir de briques disposées à un endroit précis de fabriquer un mur à un endroit donné. Pour

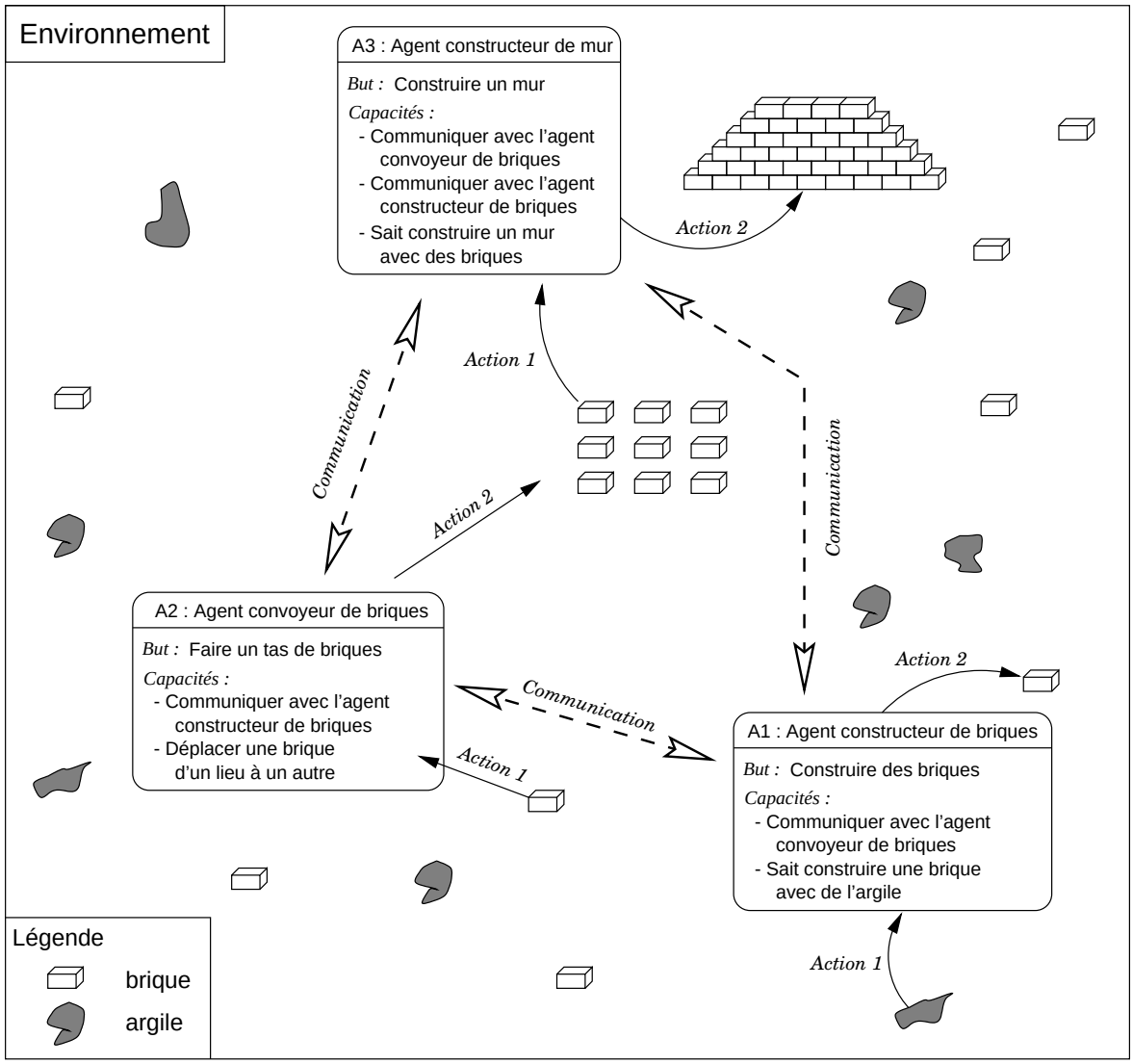


FIG. 4.12 – Exemple d'une architecture multi-agents.

cela, il ordonne à l'agent A1 de construire des briques, et à l'agent A2 de lui donner les briques construites. La simulation s'arrête lorsque l'agent A3 apprécie la hauteur et la longueur du mur construit comme étant conforme à la norme en vigueur.

4.3.2 Résolutions ascendante et descendante

Les deux types d'architectures du paradigme de l'IAD mettent en œuvre deux types différents de résolution de problèmes : *une résolution ascendante* et une *résolution descendante*.

Dans une architecture de tableau noir, les KS peuvent être vues comme l'implantation d'un ensemble d'expertises. L'expression « *système multi-experts* » est d'ailleurs également utilisée pour désigner une telle architecture. Les *experts* interviennent selon leurs compétences, c'est-à-dire que chacun d'eux n'est capable de résoudre qu'un nombre restreint de sous-problèmes relativement bien circonscrits. Autrement dit, dans une architecture de tableau noir, la résolution ne peut se faire que si le problème est décomposable en sous-problèmes (voir à ce propos la thèse commune de Lâasri & Maître [Lâasri & Maître, 1989]). Il s'agit donc d'une démarche *descendante* de résolution. On sait par avance que le problème est résolvable grâce à sa décomposition. À partir de solutions partielles qui sont assemblées et transformées, une solution complète est finalement produite. Pour résumer, nous dirons qu'il s'agit d'une méthode de résolution individuelle de sous-problèmes avec troncature explicite du problème à résoudre.

Avec une architecture multi-agents, « *une intelligence collective* » apparaît [Erceau & Ferber, 1994]. Une intelligence collective n'est pas la somme des intelligences individuelles mais le résultat d'interactions intelligentes entre les individus. Elle ne peut apparaître que si les agents peuvent directement interagir entre eux, ce qui n'est pas le cas dans une architecture de tableau noir. La résolution d'un problème par les agents ne peut donc pas se faire par étapes, mais progressivement et surtout globalement, puisque les agents opèrent en parallèle. Cela revient à adopter cette fois-ci une démarche *ascendante* de résolution. On ne sait pas à l'avance comment le problème peut être résolu ou bien même s'il existe une solution, mais on connaît les éléments élémentaires intervenant dans sa résolution[10]. L'interaction de ces divers éléments peut faire émerger une solution unique lorsque le système devient stable (c'est-à-dire qu'il n'y a plus de modification de l'environnement). Si le système oscille périodiquement d'un état à un autre, c'est qu'il y a deux solutions possibles. Pour résumer cette démarche de résolution, nous dirons qu'il s'agit d'une méthode de résolution collective sans troncature explicite du problème à résoudre.

[10] Pour comprendre l'intérêt des systèmes multi-agents, le lecteur ne doit pas prendre comme exemple celui qui est donné à la fig. 4.12. Il est clair que construire un mur n'est pas un problème typiquement résolvable par un système multi-agents. Il peut aussi être résolu par un système multi-experts. Cet exemple très simple n'est là que pour illustrer le principe de fonctionnement, et non les types de problèmes qui peuvent être résolus par un système multi-agents.

4.3.3 Application aux systèmes de DHM

En ce qui concerne les systèmes de dialogue homme-machine, le « problème » global que doit « résoudre » le système peut être présenté dans les termes suivants : que doit-il dire à un instant donné compte tenu de la situation courante et des dialogues passés ? D'après ce que nous avons dit précédemment, il faut déterminer si ce problème est résolvable par une méthode ascendante ou une méthode descendante, et sous quelles conditions il est possible de mettre en œuvre la méthode adoptée.

Un système de dialogue homme-machine est un ensemble complexe de par la complémentarité et **l'interdépendance** des différentes activités gérées. Il doit posséder un grand nombre de connaissances (phonétiques, morphologiques, lexicales, syntaxiques, sémantiques, dialogiques, pragmatiques), qui sont utilisées par un grand nombre de processus (analyseurs multiples, moteurs d'inférences), et tout cela selon des formalismes précis [Sabah, 1988] [Sabah, 1989].

Nous savons maintenant qu'un système de dialogue ne peut pas fonctionner de façon linéaire. Il a été montré à maintes reprises que les architectures en série entraînent des sources d'ambiguïtés artificielles : « Nous appelons *point d'embarras* des situations où l'ensemble des éléments de décision ne permettent pas au programme, à un moment donné du traitement, de prendre la bonne décision. Une ambiguïté artificielle est un point d'embarras qui *n*'est *pas* dû à la langue elle-même, mais au programme. » [Sabah, 1993, p. 209]. Par exemple, certains énoncés seront plus ou moins pertinents — la génération des intentions communicatives relève de l'activité dialogique —, selon les connaissances de l'application que possède le système — ces connaissances sont surtout utiles à l'activité applicative. Autrement dit, si le système possède un mauvais modèle de la tâche, des incohérences pourront apparaître dans ses énoncés, même s'il possède de bonnes connaissances de compréhension et de génération. De même, si le système n'est pas capable de construire un terrain commun, il ne pourra pas dialoguer correctement, car il lui manquera des informations du contexte pour faire les bonnes inférences. Il est donc nécessaire d'avoir une grande **flexibilité** des processus de résolution : un fonctionnement **opportuniste** est nécessaire.

Cependant, il est aussi nécessaire d'établir une cohérence globale du fonctionnement du système. Par exemple, bien que les connaissances pragmatiques aident à déterminer la valeur sémantique des énoncés (notamment pour résoudre les tropes), ces premières ne doivent pas entièrement prendre la place de l'analyse sémantique tant que celle-ci n'est pas finie. Un **contrôle** doit également intervenir.

Il apparaît donc que les modules et les connaissances interviennent à la fois spécifiquement à un instant donné du fonctionnement global, et de façon ordonnée dans le temps. Un **espace commun de travail** semble le plus approprié pour mettre en place efficacement ce processus.

Pour toutes ces raisons, nous concluons que le « problème global » que doit résoudre un système de dialogue homme-machine ne peut être résolu que par une méthode descendante telle que nous l'avons définie auparavant. On connaît les grandes étapes de résolution, et on

sait comment interagissent les différentes connaissances, même si de nombreux détails nous échappent encore. Une architecture de type tableau noir semble donc parfaitement adéquate. Il faut d'ailleurs remarquer que c'est dans le domaine des systèmes de dialogues oraux qu'est apparu ce type d'architecture avec le système HEARSAY II[11].

4.3.4 Architecture du système AMI

Nous avons développé une architecture de tableau noir à contrôle procédural classique. D'une part, cette architecture est suffisante pour atteindre nos objectifs d'expérimentation du modèle Génédic. D'autre part, une forte analogie entre l'espace commun du tableau noir et la mémoire de travail, dont nous avons montré dans la première partie de la thèse la nécessité pour la co-construction d'un terrain commun, a également guidé notre choix.

Le système AMI est un système à base de connaissances : il se compose d'une expertise de l'application, d'une expertise de traitement de la langue naturelle et d'une expertise de traitement du dialogue homme-machine (voir Fig. 4.13).

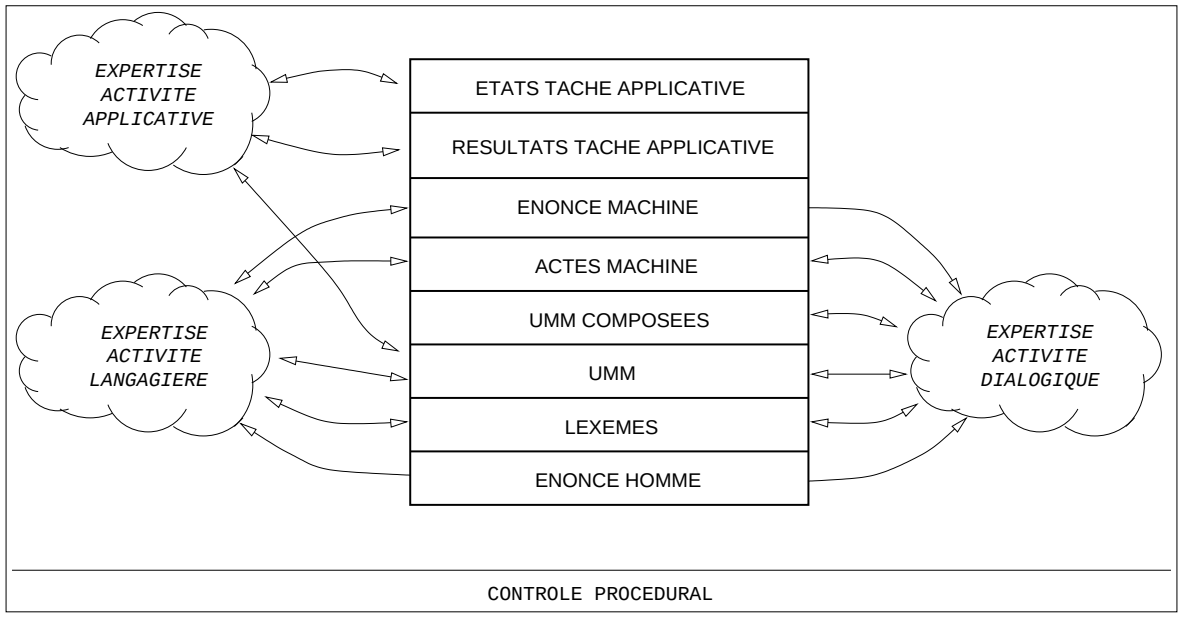


FIG. 4.13 – Le tableau noir du système AMI.

Ces trois expertises sont codées sous forme de sources de connaissances qui exploitent les éléments communs des niveaux du tableau et les bases de connaissances (sémantiques, procédurales, factuelles, etc.) privées. La valeur d'intérêt d'un KSAR est calculée en prenant en compte la valeur statique de la KSOURCE correspondante et le nombre d'éléments vérifiant ses conditions de déclenchement. Les huit niveaux retenus sont décrits ci-dessous.

[11] Ce rapporter à *Blackboard Systems*, R. Engelmore et T. Morgan (éds.), Addison-Wesley, Massachusetts, 1988.

- Le niveau ÉNONCÉ HOMME mémorise les énoncés de l'utilisateur du système sous forme de phrases écrites en vu de leurs analyses. Il intervient évidemment dans les KS de l'expertise langagière mais également dans les KS de l'expertise de traitement du dialogue pour calculer la structure du dialogue en court.
- Le niveau LEXÈMES mémorise les éléments de l'analyse linguistique des énoncés de l'utilisateur. Les KS de l'expertise de traitement du langage interviennent naturellement à ce niveau. Ce niveau intervient également dans les KS de l'activité dialogique pour la phase d'interprétation des éléments analysés.
- Le niveau UMM est la mémoire de travail à proprement parler. Il garde actives les unités minimales de la mémoire de l'interaction courante. Les KS des trois expertises ont accès à ce niveau pour construire les arborescences de structure d'action.
- Le niveau UMM COMPOSÉES sert à construire les arborescences équivalentes c'est-à-dire les arborescences soumises aux opérations de focalisations décrites Sect. 3.4.2. Ce niveau intervient uniquement dans certaines KS de l'expertise du dialogue. Nous l'avons défini afin de simplifier le processus de reconnaissance des intentions de communications.
- Le niveau ACTES MACHINE est utilisé pour construire les actions langagières de la machine. Elles sont créées à partir des intentions du niveau précédent par les KS de l'expertise du dialogue puis transformées en phrase de la langue naturelle par les KS de l'expertise adéquate.
- Le niveau ÉNONCÉ MACHINE mémorise les énoncés créés à partir des actions langagières du niveau précédent. C'est évidemment l'expertise du traitement de la langue qui y accède principalement, mais aussi les KS de l'expertise du dialogue pour mettre à jour la structure en UMI.
- Les niveaux intitulés RÉSULTATS TÂCHE APPLICATIVE et ÉTATS TÂCHE APPLICATIVE ne servent qu'aux KS de l'expertise de l'application. Le premier niveau sert à mémoriser divers résultats issus de la recherche dans les différentes bases de données de l'application, tandis que le second niveau sert à indiquer l'état courant de l'activité applicative.

Toutes les KS sont écrites dans un même formalisme : elles sont nommées, possèdent une valeur d'intérêt et font intervenir un ou plusieurs niveaux du tableau noir. Elles possèdent une ou plusieurs pré-conditions universelles (qui s'appliquent à un ou plusieurs éléments du niveau de la précondition considérée) et/ou existentielles (qui s'appliquent à un seul élément du niveau de la précondition considérée) et éventuellement une condition principale. Le code de chaque KS est lu au chargement du système et transformée en objet CLOS par un réseau d'ATT (*Augmented Transition Tree*). Par exemple, la Fig. 4.14 est la KS qui lance l'analyseur linguistique écrit en CLIPS.

```
(defregle-r analyse *dialogueur* 50000

  DOCUMENTATION
    "Analyse l'énoncé de l'utilisateur"

  PRECONDITIONS
    niv-enonceh #'disponible -> enonce

  ACTIONS
    ;; Création du fichier phrase à analyser
    (when (probe-file *fichier-phrase*) (delete-file *fichier-phrase*))
    (with-open-file
     (ofile *fichier-phrase* :direction :output :if-exists :overwrite :if-does-not-exist :create)
     (format ofile "~(~A~)" (cons 'analyser (str2cons (expression enonce)))))

    ;; On fait place nette avant de lancer l'analyse
    (when (probe-file *fichier-sortie*) (delete-file *fichier-sortie*))
    (when (probe-file *fichier-liste-facts*) (delete-file *fichier-liste-facts*))

    ;; On lance l'analyse
    (run-program "clips" :arguments `("-f2" ,*fichier-analyse*))

    ;; On récupère le résultat
    (let (list-facts)
      (with-open-file
       (ifile *fichier-liste-facts* :direction :input)
       (setf liste-facts (reverse (loop while (listen ifile) do
                                  append (read-from-string (format nil "(~S)" (read ifile)))))))
      (with-open-file
       (ifile *fichier-sortie* :direction :input)
       (loop while (listen ifile) do
         (setf lex (read-from-string (read-line ifile)))
         (cree-lexeme
          :pos (cadr (second lex))
          :hypothetique (case (cadr (third lex)) (FALSE nil) (TRUE t))
          :cat (cadr (fifth lex))
          :expression (list (format nil "~A" (cadr (sixth lex))))
          :constituants (if (> 0 (cadr (second lex))) nil (cdr (eighth lex)))
          :origine (car liste-facts)
          :cems (cdr (ninth lex)))
         (setf liste-facts (cdr liste-facts)))))
)
```

FIG. 4.14 – Exemple de la KS d'analyse des énoncés.

4.4 Le système AMI2

4.4.1 Les trois experts du système AMI2

Les sources de connaissances du système AMI2 sont groupées en trois domaines d'expertises[12]. Nous avons ainsi défini trois « experts virtuels » indépendants qui communiquent indirectement via les éléments du tableau noir. Nous avons identifié les trois experts suivants :

- un expert de l'application visée qui fournit des services à l'utilisateur ;
- un expert de la langue qui analyse les énoncés de l'utilisateur et génère les énoncés de la machine ;
- et un expert du dialogue dont le rôle est d'assurer une interaction dialogique cohérente et pertinente.

Ce découpage n'est pas artificiel car il correspond à notre approche du dialogue homme-machine présenté dans le chapitre précédent. Il permet en effet de mettre en œuvre l'autonomie des différentes activités, ce qui est d'ailleurs plus simple pour le développement. De plus, les trois experts peuvent être modifiés sans que cela entraîne de trop lourdes conséquences sur l'ensemble du logiciel. Par exemple, l'analyseur linguistique était initialement écrit en Common Lisp. Nous avons pu le changer facilement par une version plus rapide écrite en CLIPS. On peut également envisager de changer de langue en passant par exemple du français à l'anglais, ou même de garder les deux ce qui sera utile pour une expérimentation de grande envergure via le Web.

4.4.1.1 L'expert de l'application

But

Le but de l'expert de l'application est d'assurer un service de messagerie écrite. Cette activité est interne au système ; elle n'intervient pas directement avec l'utilisateur mais agit sur le comportement du système par la modification des éléments du niveau UMM.

Objectifs

Les objectifs à atteindre sont liés à la gestion des messages : l'expert doit être capable de sauvegarder puis de restituer les messages des utilisateurs. Il doit être également capable de savoir s'il a des messages en attente pour l'appelant et les lui restituer à sa demande. Lors de cette restitution, il indique la date, l'heure et l'expéditeur du message, et donne alors le contenu du message tel qu'il a été écrit par l'expéditeur.

Les messages peuvent être adressés à plusieurs destinataires en même temps. Ils sont sauvegardés *en clair*, c'est-à-dire non cryptés, dans un fichier. Un message est effacé de ce fichier lorsque tous les destinataires en ont pris connaissance.

[12] Le code source complet du système AMI2 est donné en Annexe B.

Moyens

Les ressources exploitées par cette activité sont une représentation de la tâche (environ 25 sources de connaissances), un annuaire du personnel du laboratoire (une base de données objet d'environ 40 personnes dont un extrait est donné Fig. 4.15), un planning journalier des personnes (inclus dans la base de données) et enfin un serveur de messagerie écrite qui enregistre et restitue les messages laissés par les utilisateurs.

```
(cree-item-personne
 :LENOM "Lemeunier"
 :LEPRENOM "Thierry"
 :POSTE "3865"
 :RESPONSABLE nil
 :GROUPE 'GE1G2
 :NIVEAU 'ATER
 :EMPLOI (make-array '(11 5)
                     :initial-contents '(( "R" "R" "R" "R"  "TP" )     ; 8 - 9 h
                                         ( "R"  "R" "R"  "ABS" "TP" )  ; 9 - 10 h
                                         ( "C"  "R" "R"  "R"  "TD" )   ; 10 - 11 h
                                         ( "C"  "R" "R"  "TD" "TD" )   ; 11 - 12 h
                                         ( "RE" "R" "RE" "TD" "ABS" )  ; 12 - 13 h
                                         ( "R"  "R" "ABS" "RE" "R" )   ; 13 - 14 h
                                         ( "R"  "R" "ABS" "ABS" "R" )  ; 14 - 15 h
                                         ( "ABS" "R" "ABS" "ABS" "R" ) ; 15 - 16 h
                                         ( "ABS" "R" "ABS" "ABS" "R" ) ; 16 - 17 h
                                         ( "ABS" "R" "ABS" "ABS" "R" ) ; 17 - 18 h
                                         ( "ABS" "R" "ABS" "ABS" "R" )) ; 18 - 19 h
```

FIG. 4.15 – Exemple d'une entrée de l'annuaire.

Le modèle de la tâche peut être représenté sous la forme de graphes d'états finis (cf. Fig 4.16). Nous avons deux tâches principales qui sont l'enregistrement d'un message et la restitution des messages et deux sous-tâches secondaires qui sont l'identification de l'appelant et si nécessaire l'identification de l'appelé. Les graphes expriment les contraintes suivantes :

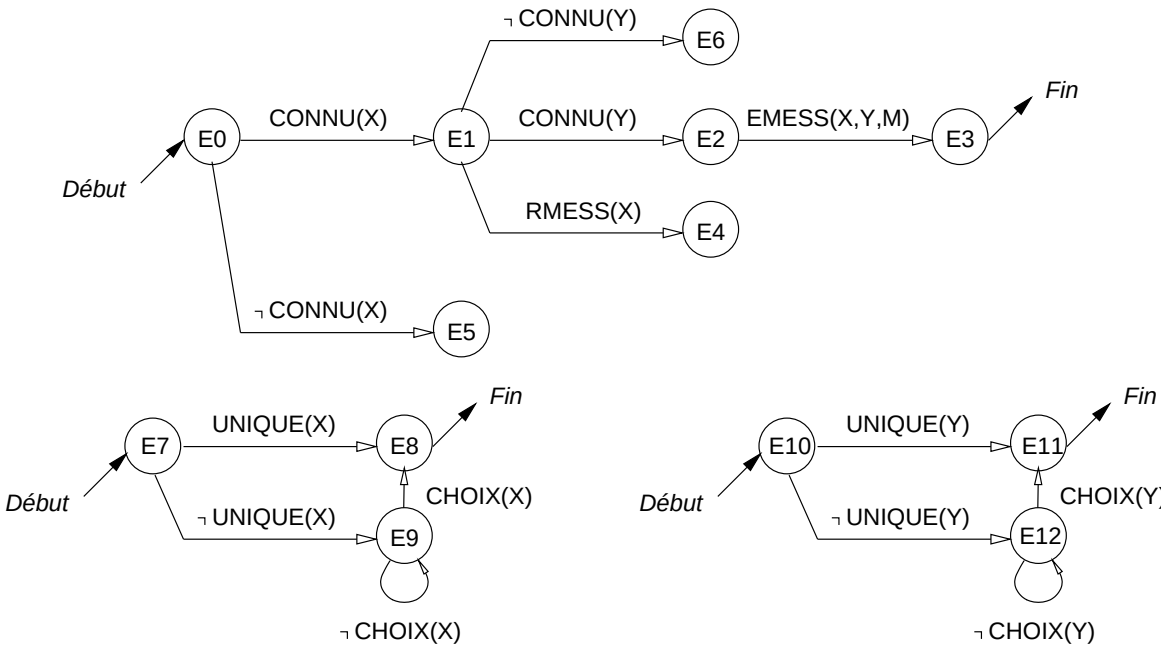
- pour enregistrer un message le système doit connaître les identités de l'appelé et de l'appelant ;
- pour restituer les messages il doit connaître uniquement l'identité de l'appelant.

Pour modifier l'application de telle sorte qu'elle devienne accessible de l'extérieur du bâtiment de l'IC2, nous devrons introduire les contraintes suivantes : les appelants ne sont pas forcément des membres de l'IC2 (donc ils ne sont pas dans tous les cas connus du système), tandis que les appelés doivent l'être.

4.4.1.2 L'expert du traitement de la langue naturelle

But

Le but de l'expert du traitement de la langue est d'assurer l'analyse linguistique des énoncés de l'utilisateur et la génération des énoncés du système.



X : la personne appelante
Y : la personne appelé
M : le message

E0 : étape principale d'initialisation de l'application
E1 : la personne appelante est référencée dans l'annuaire
E2 : la personne appelé est référencée dans l'annuaire
E3 : enregistrer le message M de X pour Y
E4 : restituer les messages pour X
E5 : la personne appelante n'est pas connue
E6 : la personne appelé n'est pas connue
E7 : début de la sous-tâche d'identification de l'appelant
E8 : l'appelant est identifié
E9 : choisir parmi plusieurs choix l'identité de l'appelant
E10 : début de la sous-tâche d'identification de l'appelé
E11 : l'appelé est identifié
E12 : choisir parmi plusieurs choix l'identité de l'appelé

FIG. 4.16 – Graphes de l'application AMI2.

Objectifs

Les objectifs à atteindre sont de fournir un analyseur robuste fondé sur l'idée que la langue utilisée dans un dialogue n'est pas une langue « normée » et qu'elle diffère par cela de la langue des discours écrits ou oraux. De plus, les connaissances linguistiques doivent pouvoir être facilement enrichies de nouvelles lexies et de nouvelles structures syntaxiques.

L'expert doit pouvoir agir en sorte de différencier les termes ambigus. Par exemple, dans la phrase « *je voudrais parler à Jean* », la lexie « *Jean* » est ambiguë car il peut s'agir du prénom ou du nom d'une personne de l'IC2. Dans ce cas, l'expert crée une structure d'action en mémoire de travail (niveau UMM). Celle-ci donne alors lieu à différentes intentions de communication.

De la même manière, l'expert doit être capable d'agir en sorte de vérifier la validité des termes hypothétiques[13] pris en compte pour construire une signification potentielle. Par exemple, si l'expression « *c'est* » n'est pas catégorisée, il peut faire l'hypothèse que l'énoncé « *c'est Thierry* » signifie que l'appelant décline son identité.

D'après le modèle Génédic, l'analyseur devrait être théoriquement capable de poser des hypothèses sur les actes illocutoires (l'énoncé en entier), sur les actions et les actants (la partie prédicative de l'énoncé), ou sur les circonstants (la partie monstration de l'énoncé). Dans le système AMI2, en plus de gérer les ambiguïtés lexicales (voir l'exemple Sect. 4.4.2.3), notre analyseur gère les hypothèses sur les actes illocutoires (voir l'exemple Sect. 4.4.2.4).

Moyens

L'expert dispose d'un analyseur inspiré de [Gérard & Nicolle, 1998] et de [Lehuen & Luzzati, 1999]. Il exploite un lexique et une grammaire de dépendances syntactico-sémantiques qui permet de reconstruire des représentations du sens de la phrase. En ce qui concerne la génération des énoncés du système, elle est très rudimentaire : il s'agit de phrases à trous attachées aux connaissances pragmatiques du système[14].

La gestion des hypothèses et des ambiguïtés lexicales est assurée par un certain nombre de tactiques dialogiques inspirées du modèle COALA. Celles-ci sont implémentées sous formes de sources de connaissances.

Modèle de la langue Le modèle de la langue que nous utilisons est très minimal. C'est un modèle dans lequel le lexique est attaché à une sémantique et à une syntaxe simplifiées (voir Fig. 4.17). Il se compose de trois types d'éléments :

[13] Les termes hypothétiques sont des lexies dont la catégorie a été inférée à partir des trigrammes et des autres lexèmes.

[14] La génération des énoncés d'un système de dialogue homme-machine est liée au modèle de la langue adopté. Comme le lecteur pourra sans convaincre à la lecture du paragraphe suivant, notre modèle de la langue est très rudimentaire, ce qui nuit fortement à la génération des énoncés du système. Cette limite peut hélas interférer (mais ne devrait pas) dans la validation du modèle Génédic, car ce dernier porte sur la génération des intentions communicatives et non sur la génération des énoncés.

- des ***catégories*** sémantico-lexicales qui attachent aux lexies des sens conventionnels potentiels ;
- des ***trigrammes*** qui sont des structures syntaxiques de trois catégories (une catégorie avant, une catégorie après et une catégorie centrale), et permettent de proposer un début de signification ;
- des ***actes*** qui sont des suites ordonnées de catégories associant une signification potentielle aux segments de phrases.

```
(categorie (nom presente01) (expression je suis))
(categorie (nom prenom) (expression thierry) (umms contacts appele appelant prenom))
(categorie (nom connecteur-et) (expression et))
(categorie (nom interlocuteur) (expression je))
(categorie (nom demande01) (expression voudrais))
(categorie (nom parler01) (expression parler) (umms requete parler))
(categorie (nom dest) (expression a))
(categorie (nom prenom) (expression daniel) (umms contacts appele appelant prenom))
(categorie (nom nom) (expression luzzati) (umms contacts appele appelant nom))

(trigramme (avant nil) (categorie presente01) (apres prenom))
(trigramme (avant prenom) (categorie connecteur-et) (apres interlocuteur))
(trigramme (avant connecteur-et) (categorie interlocuteur) (apres demande01))
(trigramme (avant nil) (categorie demande01) (apres parler01))
(trigramme (avant nil) (categorie parler01) (apres dest))
(trigramme (avant dest) (categorie prenom) (apres nom))
(trigramme (avant prenom) (categorie nom) (apres nil))

(acte (suite presente01 prenom) (umms appelant prenom))
(acte (suite parler01 dest nom prenom) (umms requete parler contacts appele nom prenom))
```

FIG. 4.17 – Extrait des connaissances linguistiques.

Principe de l'analyse L'analyse consiste à partir des catégories et des trigrammes pour reconstruire des actes illocutoires dont l'énoncé est potentiellement porteur. Pour cela, on part de la forme écrite de l'énoncé que l'on cherche à « *matcher* » avec les catégories et les trigrammes. Les comparaisons réussies donnent des *lexèmes* (mot ou groupe de mots) hypothétiques ou non-hypothétiques. Les trigrammes permettent de donner un indice de confiance aux lexèmes suivant qu'ils sont ***ancrés*** à droite ou à gauche ou des deux côtés à la fois. En cas de conflit, les lexèmes ancrés bilatéralement l'emportent sur ceux ancrés unilatéralement. De même, les lexèmes non-hypothétiques l'emportent sur les lexèmes hypothétiques. Finalement, les actes les plus couvrants (prenant en compte le plus grand nombre de lexies) sont gardés. Il s'agit donc d'une analyse séquentielle en temps limité en quatre phases résumée ainsi :

1. Création des lexèmes à partir des catégories par reconnaissances de *patterns* et création des hypothèses à partir des trigrammes et de lexèmes créés.
2. Ancrage, propagation des ancrages et suppression des lexèmes conflictuels.

3. Création des actes potentiels.
4. Suppression des actes synonymes.

Dans l'exemple de la Fig. 4.18 où nous avons repris le résultat de l'analyse de l'énoncé « *je suis thierry et je voudrais parler à daniel luzzati* », nous remarquons que l'analyseur a fait un grand nombre d'hypothèses (les lexèmes L*H), mais qu'aucune d'entre elles n'est reprise dans les deux actes A834 et A792.

Le nombre d'hypothèses posées à l'analyse varie selon les trigrammes présents dans la grammaire. Les hypothèses sont prises en compte uniquement lorsqu'il y a échec dans l'analyse : soit aucun acte n'est généré, soit il y a des actes hypothétiques (constitués d'au moins un lexème hypothétique).

4.4.1.3 L'expert du dialogue homme-machine

But

Le but de l'expert du dialogue homme-machine est de mener à bien l'interaction avec l'utilisateur dans le cadre de l'application considérée.

Objectifs

L'expert doit être capable d'assurer l'intercompréhension nécessaire pour mener à bien l'interaction. En particulier, l'expert doit assurer la co-construction du sens interactionnel, c'est-à-dire permettre la négociation du sens. Les objectifs à atteindre sont d'une part, l'interprétation des énoncés de l'utilisateur, et d'autre part, la génération des intentions de communications.

Moyens

Cet expert s'appuie sur un interpréteur dont le principe de fonctionnement est expliqué ci-après (cf. Fig. 4.19) et sur l'implantation du modèle Génédic. Pour ce dernier, les configurations remarquables, les opérations de focalisations et les filtres des intentions sont tous codés sous forme de sources de connaissances. Seules les lois comportementales ne sont pas explicitement représentés[15].

L'interprétation En premier lieu, l'interprétation consiste à confronter les résultats de l'analyse donnés par l'expert du traitement de la langue avec les UMM en attente de la mémoire de travail. L'analyseur peut en effet donner plusieurs valeurs sémantiques à un même énoncé. Ces significations doivent alors être confrontées avec le contexte réel du dialogue en cours. La confrontation permet de ne retenir que la signification correspondant aux attentes

[15] Il serait intéressant de représenter les lois comportementales afin d'en tenir compte explicitement dans le filtrage des intentions de communications et d'en étudier les cas de transgression.

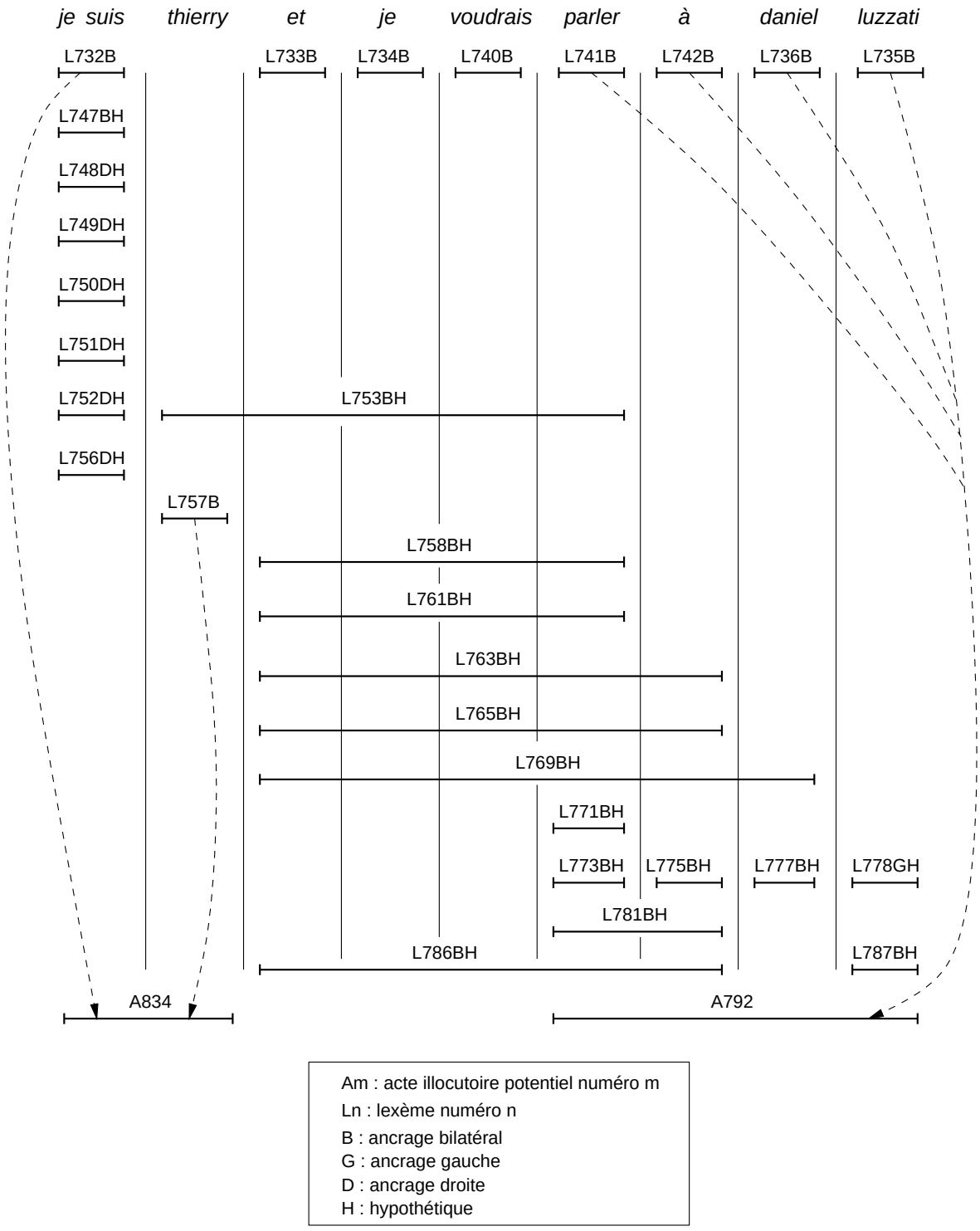


FIG. 4.18 – Exemple d'analyse.

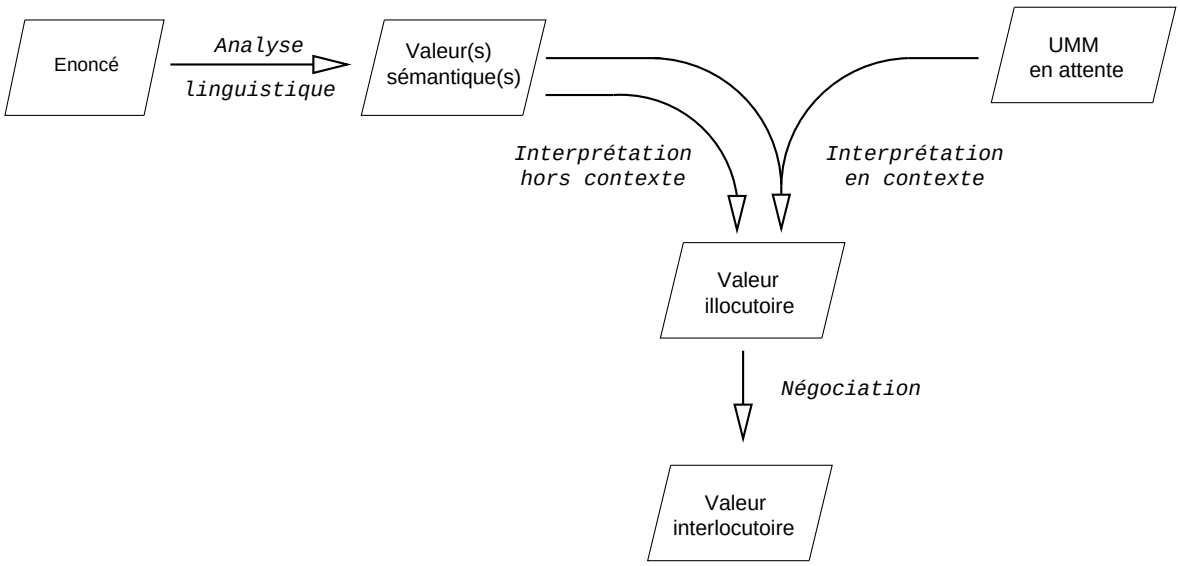


FIG. 4.19 – Le principe de l'interprétation.

courantes. Dans ce cas, la négociation est implicite puisqu'il y a accord sur la valeur illocutoire. L'interprétation de la machine va donc dans le sens de celui qu'a intentionné l'utilisateur.

L'autre cas possible d'interprétation se déroule lorsque les valeurs sémantiques de l'analyse de l'énoncé correspondent à aucune UMM en attente. Il s'agit en l'occurrence du problème général des « non-attendus ». Si la valeur sémantique est unique, le système va confondre délibérément la valeur sémantique de l'énoncé avec sa valeur illocutoire car il est dans l'incapacité d'inférer cette dernière à partir du contexte. S'il existe plusieurs valeurs sémantiques et qu'aucune d'elle n'est attendue, le système pose autant d'hypothèses d'interprétation qu'il y a de valeurs et en choisie une comme valeur illocutoire. La suite de l'échange confirmera ou infirmera cette hypothèse. Il s'agit alors d'une négociation explicite du sens interactionnel. Ce dernier cas n'est pas géré dans le système AMI2.

4.4.2 Le développement d'AMI2

La programmation de la seconde version du projet AMI est finie et nous avons pu effectuer la phase de tests de fonctionnement du logiciel. Les exemples des quatre sections suivantes illustrent ce bon fonctionnement.

Nous envisageons maintenant d'expérimenter le logiciel en restreignant d'abord l'accès au site, puis, si les résultats sont positifs, en permettant un plein accès de l'extérieur. Il sera sans doute nécessaire à cette occasion d'augmenter les fonctionnalités de l'application, comme par exemple la gestion d'une boîte aux lettres avec un classement automatique des messages. L'objectif de cette expérimentation sera d'étudier les limites du modèle Génédic et non d'évaluer le système AMI2 (voir à propos de l'évaluation la discussion du dernier chapitre Sect 5.3 p. 124).

4.4.2.1 Exemple d'enregistrement d'un message

Dans le dialogue Fig. 4.20, l'utilisateur souhaite poster un message. Son premier énoncé, qui est entièrement compris, donne toutes les informations nécessaires pour que le système puisse effectuer l'action demandée. Celle-ci est donc effectuée puis le système confirme son bon déroulement. Le dialogue est alors clos sur l'initiative de l'utilisateur (il aurait pu aussi formuler une nouvelle requête).

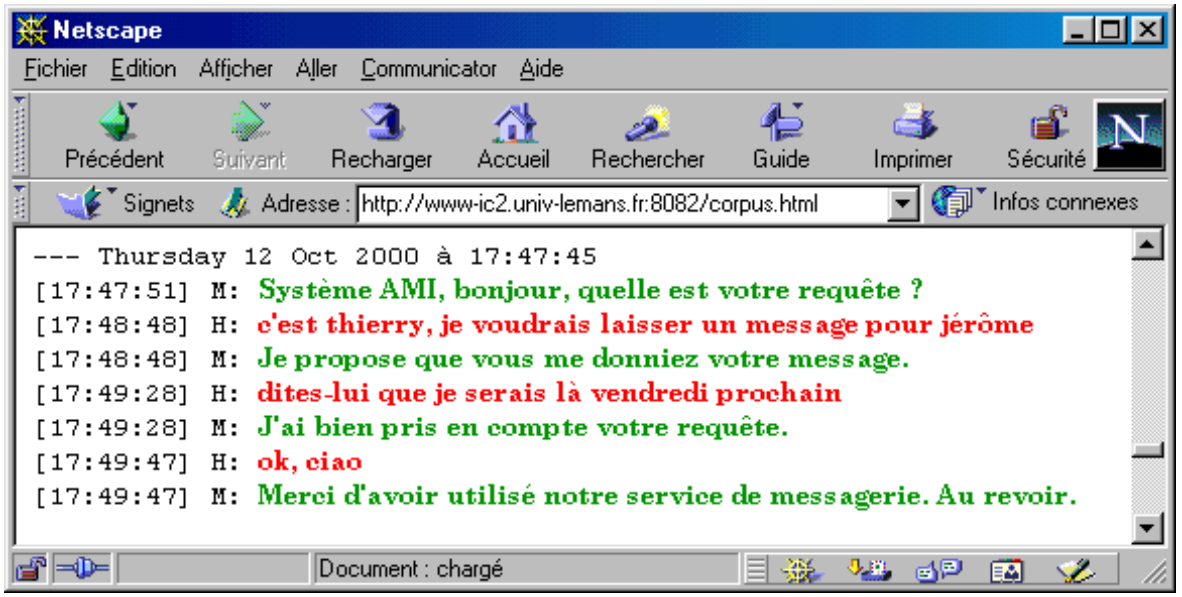


FIG. 4.20 – Enregistrement d'un message.

4.4.2.2 Exemple de restitution d'un message

Dans le dialogue Fig. 4.21, un utilisateur souhaite lire ses messages. Sa requête est prise en compte, mais le système a besoin de connaître l'identité de l'utilisateur pour chercher les éventuels messages postés à son attention. Le système lui indique alors la présence de messages en attente. L'utilisateur semble décidé à les lire, et le système les lui restitue. Le dialogue est clos par l'utilisateur.

4.4.2.3 Exemple de gestion d'une ambiguïté lexicale

Dans le dialogue Fig. 4.22, l'énoncé initial de l'utilisateur pose un problème d'analyse au système car, selon ses connaissances, la lexie « *richard* » peut être catégorisée soit comme un nom soit comme un prénom. Le reste de l'énoncé est analysé sans difficulté. L'expert du traitement du langage va donc agir de sorte de résoudre cette ambiguïté lexicale : il crée une structure d'action en mémoire de travail ; celle-ci donne une intention de communication qui débouche sur l'énoncé « *Est-ce que 'RICHARD' est un nom ?* » La confirmation de cette hy-

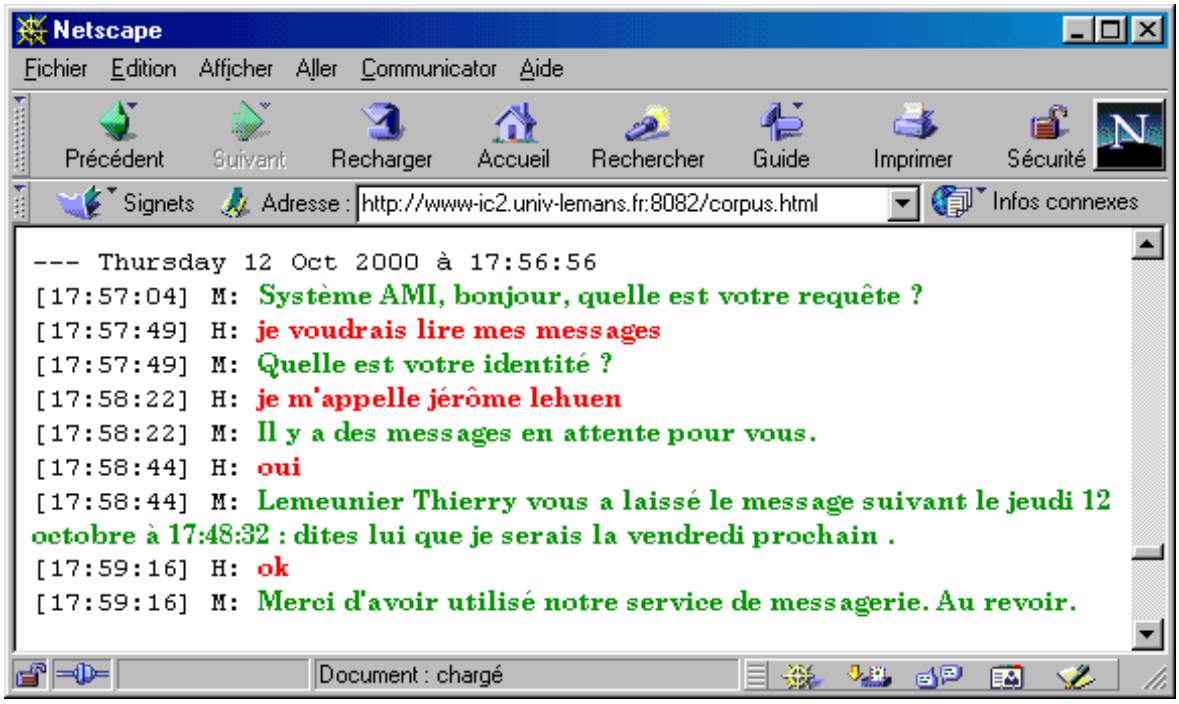


FIG. 4.21 – Restitution d'un message.

pothèse d'analyse par l'utilisateur permet au système de finir son analyse de l'énoncé initial de l'utilisateur, puis d'effectuer une recherche dans la boîte de messages de l'utilisateur, recherche qui s'avère négative. L'interaction est close par l'utilisateur.

4.4.2.4 Exemple de gestion d'une incompréhension sémantique

Dans le dialogue Fig 4.23, la machine ne comprend pas totalement l'énoncé initial de l'utilisateur, mais est en mesure de poser une hypothèse d'analyse grâce à ces connaissances linguistiques. Le système essaie alors de valider cette hypothèse. Cette dernière est validée par l'utilisateur. Le dialogue continue : après avoir obtenu l'identité de l'utilisateur, la machine indique la présence de messages en attentes ; ceux-ci sont restitués ; l'utilisateur met alors fin à l'interaction.

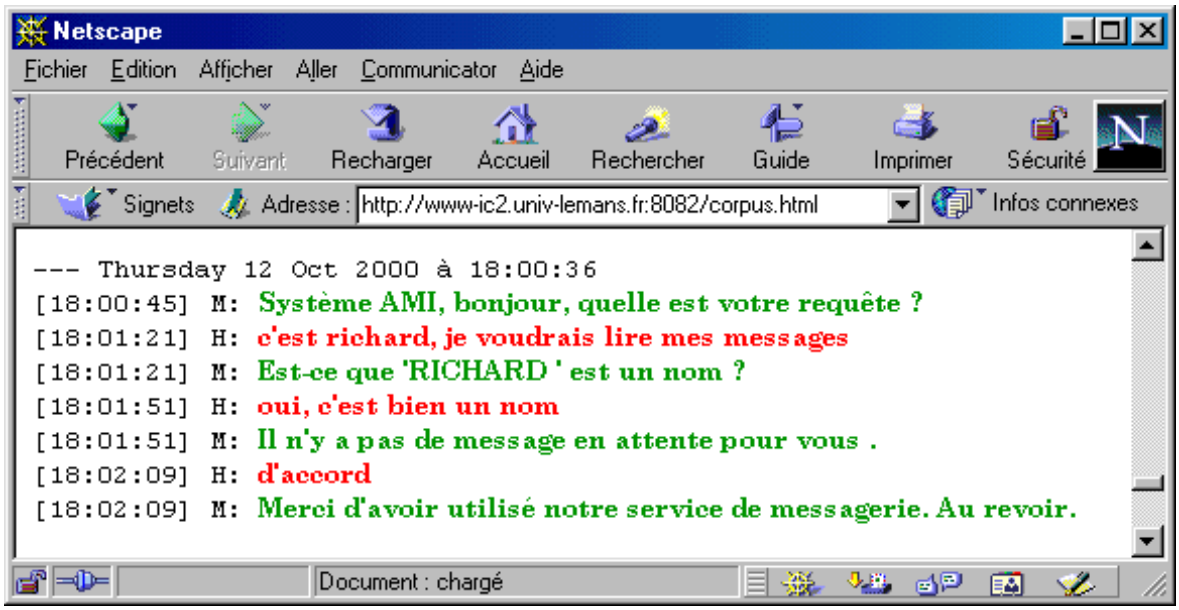


FIG. 4.22 – Gestion d'une ambiguïté lexicale.

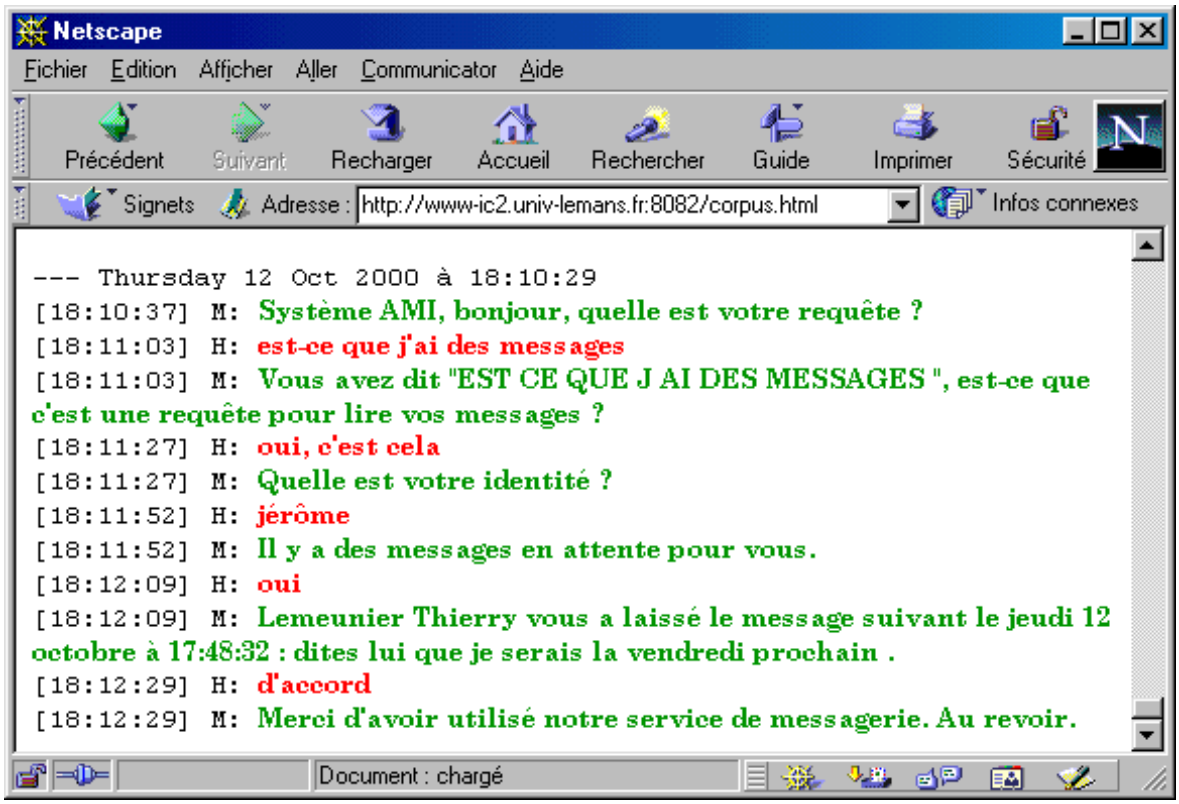


FIG. 4.23 – Exemple d'une incompréhension sémantique.

Chapitre 5

Bilan et perspectives

Bien que nous n'ayons pas encore de résultats de l'expérimentation du système AMI2, nous pouvons d'ores et déjà prétendre à un certain nombres d'apports de notre travail de thèse. Parallèlement, nous mettons en exergue les limites du modèle Génédic et les améliorations à y apporter qui nous semblent intéressantes pour tout système de dialogue en général. Nous finissons ce chapitre en ouvrant une discussion sur l'évaluation des systèmes de dialogue en langue naturelle.

5.1 Notre contribution

Notre travail porte sur la modélisation d'un agent artificiel et non réel. Nous n'avons aucune prétention quant à sa validité pour le fonctionnement et les capacités cognitives des utilisateurs humains. Nous nous sommes simplement placé dans la perspective de la co-construction du sens et de l'usage du terrain commun pour y parvenir.

Du point de vue conceptuel, on a pu voir que les modèles de dialogue fondés uniquement sur la notion de plan sont inadéquats à rendre compte de l'interaction homme-machine telle qu'elle devrait s'organiser, c'est-à-dire comme un processus commun de co-construction du sens s'appuyant sur la notion, centrale à nos yeux, de terrain commun (cf. Sect 2.3.2 p. 30).

À partir du modèle conceptuel des conversations langagières de Nicolle et Saint-Dizier De Almeida (cf. Sect. 2.3.2 p. 30), nous avons proposé pour rendre de compte de ce processus une modélisation des intentions communicatives s'appuyant sur la reconnaissance de configurations remarquables de structures d'éléments d'une mémoire de travail (cf. Sect 3.1.4 p. 56).

En reprenant la typologie des applications présentées dans l'introduction de ce document, nous pensons que le développement de modèles tels que le nôtre, est une étape nécessaire vers des applications du type *maître-maître*. Ce type d'application nécessite en effet des dialogues de négociation, dialogues dont il est difficile de rendre compte *simplement* avec les modèles par plans. Au contraire, cela est plus facile dès lors que l'on s'appuie sur l'idée de la co-construction du sens interactionnel.

Enfin, d'un point de vue informatique, après les thèses de Bricon-Souf [Bricon-Souf, 1994] et de Lehuen [Lehuen, 1997b], nous nous sommes efforcé de montrer qu'une architecture logicielle de type tableau noir était appropriée pour implémenter un système de dialogue homme-machine[1]. De plus, la méthode de développement que nous avons adoptée, identique à celle de Rouillard, semble être une voie intéressante pour l'élaboration des systèmes de dialogue accessibles à distance. Il faudra alors étudier les dialogues des corpus obtenus pour voir s'ils sont différents des dialogues obtenus avec les systèmes présentiels[2].

5.2 Limites et améliorations

Notre modèle ne peut pas prétendre à une grande universalité : seules des applications simples sont abordables avec le modèle des intentions tel qu'il est présenté dans ce document. Nous n'avons pas démontré son adaptabilité aux applications de type *maître-maître*.

De plus, nous n'avons pas pris en compte les aspects liés à la temporalité et le traitement des références dans les structures d'actions. En effet, nous devrons préciser, par exemple, la manière dont les éléments mnésiques actifs sont mis à jour par une interprétation contradictoire ou divergente. Dans le système AMI, les aspects référentiels sont gérés par l'expert de l'application. Il s'agit de références définies non-anaphoriques (les appelants et les appelés). Les autres types de références n'ont pas été abordés.

Les limites de notre travail que nous venons d'évoquer sont déjà autant de voies possibles d'amélioration. En voici d'autres « pêle-mêle » :

- Il sera intéressant de travailler sur les problèmes liés à l'oubli des structures mnésiques[3] : oubli au cours d'un dialogue et mémorisation/rappel des anciens dialogues avec un même utilisateur. Nous pourrons à cet égard nous appuyer sur la notion d'histoire interactionnelle développée dans [Kerbrat-Orecchioni & Plantin, 1995].

  L'idée centrale est la suivante : un système de dialogue doit-il être capable de ne rien oublier ou doit-il être doté d'une capacité d'oubli ? Par exemple, il n'est pas rare chez les humains qu'ils oublient ce qu'ils se sont dit entre deux conversations espacées ne serait-ce que d'une semaine. Une machine, à l'inverse, est capable de reprendre un dialogue interrompu depuis cinq minutes sans problème si elle est programmée pour le faire, ce qui ne sera pas toujours le cas de son utilisateur.

  Ici deux attitudes sont concevables. Si l'on décide que la machine doit être mise au niveau de l'utilisateur humain, alors ce n'est pas un oubli « sec » et définitif qu'il faut réaliser, mais un oubli progressif, où les souvenirs des anciens dialogues se transforment

[1] Un tel système peut être vu à un niveau macroscopique comme étant lui-même un agent artificiel fortement cognitif et dialoguant avec des agents humains.

[2] Dans sa thèse, Rouillard a effectivement obtenu un corpus de dialogues via le Web avec le système HALPIN (interactions orales et manipulation directe de l'interface) [Rouillard & Caelen, 1998]. Cependant, il n'a pas fait ce travail de comparaison, la mise à distance d'un tel système étant déjà à elle seule un travail important.

[3] Nous tenons particulièrement à cet aspect, Martial Vivet lui-même nous en avez suggéré l'intérêt.

pour ne former qu'un « noyau » ne retenant que les éléments cruciaux, à la manière des humains. Au contraire, si l'on pense que le dialogue homme-machine est un type de dialogue particulier, alors les singularités, qui ne manqueront pas d'apparaître avec une machine incapable d'oublier, doivent être acceptées par les utilisateurs humains.

- En ce qui concerne les arborescences, nous pouvons envisager d'ajouter d'autres types de liaisons entre fils et père. Par exemple, nous pourrions différencier la relation disjonctive inclusive de la relation disjonctive exclusive. On peut également imaginer ajouter une relation conjonctive *n parmi m* qui signifierait que n nœuds parmi les m nœuds fils sont obligatoires pour que l'arborescence soit considérée comme complète. Dans tous les cas, ce genre d'amélioration nécessitera une revue complète des configurations remarquables (mais pas nécessairement des règles dialogiques).
- En théorie, une représentation de l'espace interactionnel est nécessaire au système, afin qu'il puisse en avoir une représentation et agir s'il juge que l'interaction se déroule mal. En particulier, cela peut être un moyen de s'apercevoir d'une totale incompréhension avec l'utilisateur ou d'un quiproquo. Cela peut aussi aider à juger de l'état interactionnel selon la proportion du nombre d'actes satisfaits sur le nombre d'actes insatisfaits au cours et à la fin du dialogue. On peut alors envisager diverses interprétations : si cette proportion est de n/n c'est que l'interaction est équilibrée ; si elle est de n/m avec $n > m$ alors le locuteur doit être globalement satisfait du dialogue ; enfin, si la proportion est de m/n avec $m < n$ alors il doit être plutôt insatisfait.
  Considérons à nouveau le modèle COALA de Lehuen. Un dialogue se structure en UMI subjectives qui s'organisent selon les axes régissant et incident de Luzzati (cf. Sect. 2.4 p. 37). La structure peut différer si on considère le point de vue de la machine sur le dialogue ou si on considère celui de l'utilisateur. Il serait maintenant intéressant de considérer la structuration en UMI selon la satisfaction et la non-satisfaction des actes langagiers : alors que dans COALA une UMI est validée positivement lorsque les attentes de la machine liées à l'UMI sont satisfaites et négativement dans le cas contraire, on pourrait dire qu'une UMI est validée positivement lorsque le système juge que son acte est satisfait et négativement dans le cas contraire.
  Reprenons l'exemple de dialogue entre un petit-fils et sa grand-mère Fig. 2.9 p. 36. Selon notre proposition, un tel dialogue serait représenté par la structure Fig. 5.1.

  Il est intéressant d'observer l'évolution dans ce dialogue de la proportion du nombre d'actes satisfaits par rapport au nombre d'actes insatisfaits du petit-fils. Nous avons reporter ce rapport sur la figure : en (1) il est de 0/1 car PF1 n'est pas satisfait ; en (2) il est de 1/1 car PF3 semble satisfait ; en (3) il devient 1/2 car PF6 n'est pas satisfait ; en (4) GM9 satisfait PF8 ce qui satisfait par la même occasion PF6, le rapport devient donc 3/1 ; enfin GM9 entraîne également la satisfaction de l'acte initial PF1, le rapport

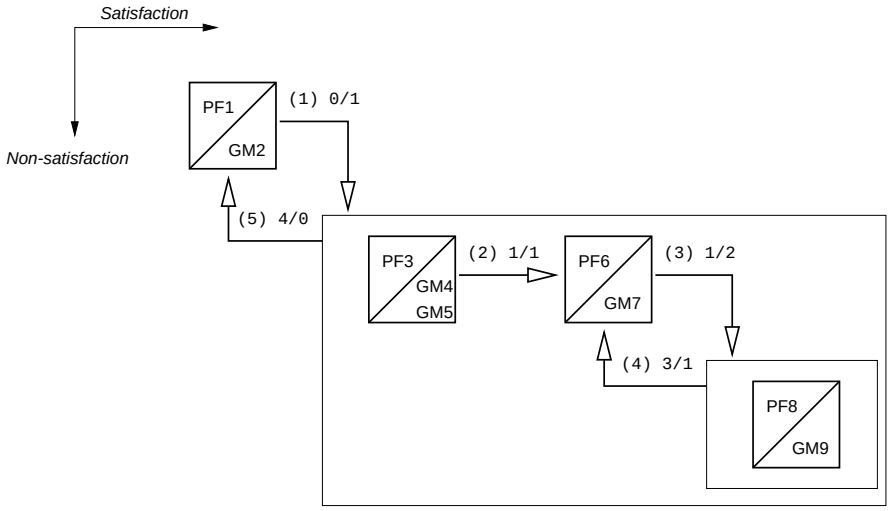


FIG. 5.1 – Structuration selon la satisfaction des actes du petit-fils.

devient donc finalement 4/0 en (5). Tous les actes du petit-fils ont été satisfaits. Il peut se sentir heureux d'avoir à la fois son bonbon et d'avoir convaincu sa grand-mère (qui s'est d'ailleurs laissée convaincre car elle n'est pas revenue sur l'argument GM4).

5.3 Discussion sur l'évaluation

Alors que l'action GRACE d'évaluation des étiqueteurs morphosyntaxiques pour le français a publié récemment les résultats de la confrontation de 13 analyseurs en compétition [Adda *et al.*, 1999], presque rien n'existe sur les systèmes de dialogue homme-machine, hormis les méthodes quantitatives mises en œuvre pour évaluer les systèmes de dialogues oraux du type ATIS (*Air Travel Information System*) du programme mise en place par l'ARPA (*Defense Advanced Research Projects Agency*) aux États-Unis.

Comme le fait remarquer Vilnat, l'évaluation d'un système de dialogue est beaucoup plus délicate que, par exemple, un système de reconnaissance de la parole : « Il existe des critères faciles à établir pour décider si un signal de parole a été bien décodé : on peut pour cela comparer les résultats avec les mots qui ont été prononcés. En revanche, il est moins évident de décider si une intervention dans un dialogue a été correctement interprétée. » [Vilnat, 1997, p. 138]. En effet, dans le cas des systèmes de dialogue, le problème du choix des critères d'évaluation ne peut pas être résolu en comparant les résultats des modules d'interprétation. En ce qui concerne la justesse et la pratique d'une telle méthode, il nous paraît peu réaliste de s'appuyer sur des critères liés au fonctionnement interne des systèmes, car il ne peut pas y avoir de critères communs entre des systèmes conçus et fonctionnant différemment.

La difficulté dans ce genre d'évaluation provient du fait que l'objet évalué n'est pas le bon : on cherche à évaluer le fonctionnement du système et non son comportement durant les

dialogues. Nous considérons au contraire que son **fonctionnement interne** n'est pas l'enjeu de l'évaluation. Ce que l'on veut évaluer ce sont ses capacités à dialoguer avec un utilisateur humain. Aussi, une méthode d'évaluation fondée sur des critères internes n'est valable que si l'on veut s'auto-évaluer car, dans ce cas, ce que l'on cherche à faire c'est aussi bien évaluer les interactions langagières qu'évaluer la manière dont fonctionne le système. C'est une telle méthode que nous avons utilisé pour évaluer AMI1 (cf. 2.4.2.1.3).

Nous devons donc nous appuyer sur des critères externes aux systèmes, c'est-à-dire uniquement interactionnels. Essayons de préciser cette idée en prenant l'exemple d'un système de dialogue développé par Luzzati. Cet auteur a évalué le système de renseignements SNCF Dialors selon deux critères [Luzzati, 1996]. La méthode utilisée consiste à comparer les dialogues pouvant être traités par le système à ceux d'un corpus obtenu par la méthode du Magicien d'Oz. Un *taux de compétence* mesure la capacité du système à traiter les dialogues du corpus obtenu. Un *taux d'efficacité* mesure sa capacité à récupérer les problèmes de compréhension. Premièrement, ces deux critères ne sont applicables que si l'on dispose d'un corpus de référence, ce qui pose le problème de sa définition. Deuxièmement, ces critères ne mesurent pas les capacités réelles du système (il n'a pas été confronté à l'usage des utilisateurs finaux), mais la possibilité de traiter ce type d'application par ordinateur. Nous pensons donc qu'ils ne répondent pas à l'objectif fixé dans l'évaluation d'un système. On ne vise pas à mesurer la faisabilité d'un système, mais à mesurer ses capacités dialogiques lorsqu'il est confronté à une utilisation réelle ou réaliste.

Une autre idée, proposée par Vilnat [op. cité], serait d'identifier un certain nombre de phénomènes linguistiques et/ou dialogiques et de les considérer comme autant de critères d'évaluation. Cette idée est malheureusement difficile à mettre en place actuellement. Comment en effet identifier clairement les phénomènes en question, alors qu'ils sont encore des sujets d'études. Par ailleurs, ces critères font abstraction d'un facteur important oublié jusqu'ici, et qui est l'utilisateur du système. Évaluer un système de dialogue homme-machine c'est selon nous prendre aussi et surtout en compte **l'appréciation des usagers** de ce système. Par voie de conséquence, contrairement aux propos pessimistes de Vilnat qui conclut en disant « qu'évaluer un système de dialogue, et comparer son efficacité à d'autres nous paraît actuellement une tâche à la fois impossible, irréaliste et peu utile » [op. cité, p. 140], nous considérons qu'il est possible (dans une certaine mesure) d'évaluer dès maintenant nos systèmes. Simplement, il faut provisoirement laisser de côté les critères objectifs au profit de critères subjectifs. Un questionnaire présenté aux utilisateurs paraît être un premier élément intéressant[4]. D'autres auteurs parviennent également aux mêmes conclusions en ce qui concerne les systèmes de dialogues oraux [Cole *et al.*, 1997, p. 212].

Dans une évaluation par questionnaire, l'un des risques est que les utilisateurs ne fassent pas suffisamment la différence entre, d'une part, les capacités dialogiques et langagières du

[4] Il existe des méthodes spécifiques pour établir ce genre de questionnaire. L'informaticien doit travailler avec des ergonomes pour présenter et formuler correctement les questions.

système que l'on cherche à évaluer, et d'autre part ses capacités applicatives qui ne doivent pas être considérées dans l'évaluation. Les questions doivent donc porter sur les capacités d'interaction ressenties par l'utilisateur et être posées de telle sorte qu'elles lui suggèrent de juger le système non par rapport à ce qu'il peut s'attendre d'une machine, mais par rapport à ce qu'il s'attendrait d'un humain, afin d'évacuer le plus possible l'aspect applicatif. Une note globale peut être ainsi calculée selon les réponses données à une grille d'évaluation. Par exemple, nous avons établi celle Tab. 5.1 pour le système AMI2. (Elle est ouverte à toute discussion.) Le lecteur pourra voir également le questionnaire de satisfaction du système Halpin-Documentaire établi par Rouillard assez proche du nôtre [Rouillard, 2000, p. 193].

	0	1	2	3	4	5	6	7	8	9	10
Le dialogue avec la machine vous paraît-il souple ? 0 : Pas du tout souple. 10 : Aussi souple qu'un dialogue humain.							X				
Pensez-vous que la machine à compris vos requêtes ? 0 : N'en a compris aucune. 10 : Les a toutes comprises.					X						
Avez-vous compris les énoncés de la machine ? 0 : Pas du tout compris. 10 : Compris tous les énoncés.											X
Trouvez-vous les énoncés de la machine pertinent ? 0 : Pas du tout pertinent. 10 : Tout à fait pertinent.									X		
Comment le dialogue avec la machine peut-il être comparé à un dialogue avec un humain ? 0 : Complètement différent. 10 : Tout à fait identique.						X					
Le dialogue avec la machine vous paraît-il naturel ? 0 : Pas du tout naturel. 10 : Tout à fait naturel.				X							
D'après vous, qui a mené le dialogue ? 0 : Vous uniquement. 10 : La machine uniquement.										X	
Quelle note globale attribuée au dialogue avec la machine ? 0 : Le dialogue est très mauvais. 10 : Le dialogue est très bon.							X				

TAB. 5.1 – Exemple d'une grille d'évaluation d'un système de dialogue.

5.4 Conclusion

Le dialogue homme-machine est un domaine de recherche qui posent de nombreuses questions fondamentales auxquels les chercheurs de la communauté ont bien du mal à répondre. Ainsi, malgré leurs efforts, il existe encore de nombreuses zones d'ombres qu'il faut éclairer.

Le modèle des intentions pour une machine que nous avons proposé n'apporte qu'un faible éclairage à ces zones d'ombres : l'étude du dialogue, plus que tout autre thème de recherche, pose le problème de « l'appréhension » du sens pour les agents artificiels. Quel sens peuvent avoir les mots pour une machine sans corps et sans « conscience » de ce corps ? D'où peut venir ce sens ? Est-il possible pour la machine de dialoguer véritablement sans qu'elle puisse attacher aux mots un sens qui ne soit pas préconçu ?

Malgré les nombreuses questions laissées en suspens, nous espérons que ce travail contribuera à une meilleure reconnaissance de ce domaine souvent entaché de fausses idées et de faux espoirs. Nous espérons plus généralement avoir contribué à une discipline scientifique qui est celle de l'intelligence artificielle.

Bibliographie

[Adda *et al.*, 1999] G. Adda, J. Mariani, P. Paroubek, M. Rajman, et J. Lecomte. Métrique et premiers résultats de l'évaluation GRACE des étiqueteurs morpho-syntaxiques pour le français. In *Actes de TALN'99*, p. 15–24, Cargèse, Corse, 12-17 Juillet 1999.

[Amalberti *et al.*, 1989] R. Amalberti, N. Carbonell, et P. Falzon. User's Representation Of The Machine In Human-Computer Speech Interaction. Rapport Technique 1125, INRIA, Rocquencourt, décembre 1989.

[Austin, 1962] J. L. Austin. *How to do Things with Words.* Oxford University Press, 1962.

[Bachimont, 1994] B. Bachimont. *Le contrôle dans les systèmes à base de connaissances, Contribution à l'épistémologie de l'intelligence artificielle.* Hermès, Paris, 2ème édition, 1994.

[Balkanski & Hurault-Plantet, 1997] C. Balkanski et M. Hurault-Plantet. Communicative Actions in a Dialogue Model for Cooperative Discourse: an initial report. In *AAAI'97 Fall Symposium on communication actions in humans an machines*, Cambridge, MA, 1997.

[Bange, 1992] P. Bange. *Analyse conversationnelle et théorie de l'action.* Hatier, Paris, 1992.

[Bassi Acuña, 1995] A. Bassi Acuña. *Un modèle dynamique de la compréhension de textes intégrant l'acquisition de connaissances.* Thèse de doctorat, LIMSI-CNRS, Orsay, 3 mars 1995.

[Beust, 1998] P. Beust. *Contribution à un modèle interactionniste du sens, Amorce d'une compétence interprétative pour les machines.* Thèse de doctorat, Université de Caen, 22 décembre 1998.

[Bilange, 1992] E. Bilange. *Dialogue personne-machine, modélisation et réalisation informatique.* Hermès, Paris, 1992.

[Brassac & Pesty, 1999] C. Brassac et S. Pesty. *Analyse et Simulation de Conversations, De la théorie des actes de discours aux systèmes multiagents*, chap. Simuler la conversation : un défi pour les systèmes multi-agents, p. 317–345. L'Interdisciplinaire, Limonest, 1999.

[Brassac, 1993] C. Brassac. Théorie des actes de langage et intelligence artificielle distribuée. In *Actes des Journées Systèmes Multi-Agents du PRC-IA*, Montpellier, 17 décembre 1993.

[Bricon-Souf, 1994] N. Bricon-Souf. *ARCICE, Architecture réflexive d'un Compèrobot interactif, Compréhension des énoncés.* Thèse de doctorat, université de Caen, 1994.

[Bruillard & Vivet, 1994] E. Bruillard et M. Vivet. *Didactique et intelligence artificielle*, chap. Concevoir des EIAO pour des situations scolaires : approche méthodologique, p. 273–302. La pensée sauvage éditions, 1994.

[Bunt, 1995] H. C. Bunt. Semantics and Pragmatics in the DELTA system. In *Topics, Proceedings of the Second Dialogue and Discourse Workshop*, éd. L. Dybkjaer, Dublin, avril 1995. CCI, Roskilde University.

[Caron, 1997] J. Caron. *Conversation, interaction et fonctionnement cognitif*, chap. Psychologie cognitive et interactions conversationnelles, p. 221–237. Presses Universitaires de Nancy, Nancy, 1997.

[Castaing, 1993] M.-F. Castaing. Corpus de dialogues enregistrés dans un standard téléphonique. Rapport Technique 93-29, LIMSI, Orsay, octobre 1993.

[Charnay, 1999] L. Charnay. *Dialogue et Explication dans les Systèmes à Base de Connaissances, ADex, un modèle informatique pour l'énonciation.* Thèse de doctorat, université de Paris XI, Orsay, 27 septembre 1999.

[Chicoisne & Pesty, 1999] G. Chicoisne et S. Pesty. Modèle de conversation & agents rationnels socialement corrects. In *Actes de l'atelier la langue dans l'interaction personnes-machines, TALN'99*, p. 91–104, Cargèse, Corse, 12-17 juillet 1999.

[Clark, 1992] H. H. Clark. *Arenas of Language Use.* The University of Chicago Press, Chicago, 1992.

[Cohen *et al.*, 1992] éds. P. Cohen, J. Morgan, et M. Pollack. *Intentions in Communication.* Bradford Books, MIT Press, Cambridge, Massachusetts, seconde édition, 1992.

[Cole *et al.*, 1997] éds. R. Cole, J. Mariani, H. Uszkoreit, G. B. Varile, A. Zaenen, A. Zampolli, et V. Zue. *Survey of the state of the art in human language technology.* Cambridge University Press, 1997.

[Coursil, 1992] J. Coursil. *Grammaire analytique du français contemporain, Essai d'intelligence artificielle et de linguistique générale.* Thèse de doctorat, université de Caen, 1992.

[Coursil, 1993] J. Coursil. Dialog, the semiology of transfert. In *2ème Congrès Européen de Systémique*, Prague, 1993. AFCET, CSCI.

[Dessalles, 1993] J.-L. Dessalles. *Modèle cognitif de la communication spontanée, appliqué à l'apprentissage des concepts.* Thèse de doctorat, Telecom-Paris 93 E 022, ENST, 1993.

[Dessalles, 1996] J.-L. Dessalles. *Psychologie du dialogue homme-machine en langage naturel*, chap. Des machines capables d'argumenter, p. 117–126. Europia Productions, Paris, 1996.

[Dreyfus, 1993] H. L. Dreyfus. La critique heideggerienne de l'approche husserlienne et searlienne de l'intentionnalité. *Intellectica*, 2(17):27–49, 1993.

[Dupont, 1996] M. Dupont. Le modèle des attentes du lecteur dans le calcul de la référence. In *Actes de RéciTAL'96*, p. 155–160, Courcelle, 25-27 septembre 1996.

[Durand, 1996] B. Durand. *Simulation multi-agents et épidémiologie opérationnelle, Étude d'épizooties de fièvre aphteuse.* Thèse de doctorat, Université de Caen, 20 juin 1996.

[Erceau & Ferber, 1994] J. Erceau et J. Ferber. *Intelligence collective*, chap. Intelligence artificielle distribuée et systèmes multi-agents, p. 157–179. Hermès, Paris, 1994.

[Ferber & Ghallab, 1988] J. Ferber et M. Ghallab. Problématique des univers multi-agents intelligents. In *Actes des 2èmes Journées du PRC-GRECO IA*, éd. D. Pastre, p. 295–320, Toulouse, 14-15 mars 1988. Teknea.

[Ferrari, 1997] S. Ferrari. *Méthode et outils informatiques pour le traitement des métaphores dans les documents écrits.* Thèse de doctorat, Orsay, 17 décembre 1997.

[Fraczak *et al.*, 1998] L. Fraczak, G. Lapalme, et M. Zock. Variation du contenu et de la forme dans la génération de descriptions d'itinéraires en métro. In *Actes de TALN'98*, p. 72–81, Paris, 10-12 juin 1998.

[Gaudel *et al.*, 1996] M.-C. Gaudel, B. Marre, F. Schlienger, et G. Bernot. *Précis de génie logiciel.* Masson, Paris, 1996.

[Gérard & Nicolle, 1998] F. Gérard et A. Nicolle. BISTRO : un modèle de dialogue intégrant la manipulation de concepts. In *Actes de RÉCITAL'98*, éd. J. Lehuen, p. 135–147, Le Mans, 8-9 septembre 1998.

[Gérard, 1999] F. Gérard. Un processus d'interprétation progressive dans le cadre de dialogues homme-machine finalisés. In *Actes de l'atelier La langue dans l'interaction personnes-machines, TALN'99*, p. 105–114, Cargèse, Corse, 12-17 juillet 1999.

[Gleizes *et al.*, 1994] M.-P. Gleizes, P. Glize, et S. Trouilhet. Étude des lois de la conversation entre agents autonomes. *Revue Internationale de Systémique*, 8(1), 1994.

[Goffman, 1974] E. Goffman. *Les rites d'interaction.* Éditions de Minuit, Paris, 1974.

[Grau *et al.*, 1994] B. Grau, G. Sabah, et A. Vilnat. Pragmatique et dialogue homme-machine. *Technique et science informatique*, 13(1):9–30, 1994.

[Grice, 1968] P. Grice. Utterer's Meaning, Sentence-Meaning, and Word-Meaning. *Foundations of Language*, (4):1–18, 1968.

[Grice, 1975] H. P. Grice. *Syntax and Semantics 3 : Speech Acts*, chap. Logic and conversation, p. 41–58. Academic Press, New-York, 1975.

[Grosz & Kraus, 1996] B. Grosz et S. Kraus. Collaborative Plans for Complex Group Action. *Artificial Intelligence*, 86(2):269–357, 1996.

[Grosz, 1996] B. Grosz. Collaborative Systems. *AI Magazine*, 2(17):67–85, 1996.

[Guyomard *et al.*, 1995] M. Guyomard, P. Nerzic, et J. Siroux. Plans, métaplans et dialogue. In *Actes de la Journée Communication parlée et dialogue*, ENST, avril 1995.

[Haton *et al.*, 1991] J.-P. Haton, N. Bourzid, F. Charpillet, M.-C. Haton, B. Lâasri, H. Lâasri, P. Marquis, T. Mondot, et A. Napoli. *Le raisonnement en intelligence artificielle*. InterÉditions, Paris, 1991.

[Hoc, 1987] J.-M. Hoc. *Psychologie cognitive de la planification*. PUG, Grenoble, 1987.

[INRIA, 1998] Projet Langue et Dialogue, rapport d'activité de l'INRIA-Nancy, 1998.

[Kerbrat-Orecchioni & Plantin, 1995] C. Kerbrat-Orecchioni et C. Plantin. *Le trilogue*. Presses Universitaires de Lyon, 1995.

[Lâasri & Maître, 1989] H. Lâasri et B. Maître. *Coopération dans un univers multi-agents basée sur le modèle du blackboard : études et réalisations*. Thèse de doctorat, université de Nancy I, février 1989.

[Lehuen & Luzzati, 1999] J. Lehuen et D. Luzzati. Acquisition coopérative d'une compétence langagière interprétative en dialogue homme-machine. In *Actes de TALN'99*, p. 357–366, Cargèse, Corse, 12-17 juillet 1999.

[Lehuen *et al.*, 1996] J. Lehuen, A. Nicolle, et D. Luzzati. Un modèle hypothético-expérimental dynamique pour la gestion des dialogues homme-machine. In *Actes du congrès RFIA'96*, p. 180–189, Rennes, 16-18 janvier 1996.

[Lehuen, 1997a] J. Lehuen. *Apprentissage par l'interaction*, chap. Un modèle de co-adaptation langagière pour le dialogue homme/machine, p. 117–141. Europia Productions, Paris, 1997.

[Lehuen, 1997b] J. Lehuen. *Un modèle de dialogue dynamique et générique intégrant l'acquisition de sa compétence linguistique, Le système COALA*. Thèse de doctorat, Université de Caen, juin 1997.

[Lemeunier & Lehuen, 1999] T. Lemeunier et J. Lehuen. Un modèle de génération des intentions de communication pour le dialogue homme-machine. In *Actes de l'atelier la langue dans l'interaction personnes-machines, TALN'99*, p. 115–124, Cargèse, Corse, 12-17 juillet 1999.

[Lemeunier, 1996] T. Lemeunier. L'usage de l'humour en informatique. Mémoire de stage 96-09-/06/DEA, LIUM, université du Maine, 1996.

[Lemeunier, 1998a] T. Lemeunier. AMI : un système de DHM expérimental qui apprend à s'adapter aux situations interactionnelles itératives. In *Actes de RÉCITAL'98*, éd. J. Lehuen, p. 117–126, université du Maine, 8-9 septembre 1998.

[Lemeunier, 1998b] T. Lemeunier. Corpus de dialogues homme-machine du système informatique AMI1. Disponible auprès de l'auteur, juin 1998.

[Lemeunier, 1999] T. Lemeunier. La modélisation des attentes dans le système AMI. In *Actes de RÉCITAL'99*, p. 439–443, Cargèse, Corse, 12-17 juillet 1999.

[Litman & Allen, 1992] D. Litman et J. Allen. *Intentions in Communication*, chap. Discourse Processing and Commonsense Plans, p. 365–388. The MIT Press, seconde édition, 1992.

[Luzzati & Taleb, 1995] D. Luzzati et L. Taleb. Finalized spoken dialogue modelling based on communication failure. In *Proceedings of the ESCA Workshop on Spoken Dialogue Systems*, Visgø, Danemark, 1995.

[Luzzati, 1995a] D. Luzzati. De l'erreur en DHM. *Cahiers de Linguistique Française*, (16):175–192, 1995.

[Luzzati, 1995b] D. Luzzati. *Le dialogue verbal homme-machine*. Masson, Paris, 1995.

[Luzzati, 1996] D. Luzzati. *Psychologie du dialogue homme-machine en langage naturel*, chap. Pour une typologie des tâches en dialogue homme-machine, p. 107–115. Europia Productions, Paris, 1996.

[Moeschler & Reboul, 1994] J. Moeschler et A. Reboul. *Dictionnaire Encyclopédique de Pragmatique*. Éditions du Seuil, Paris, 1994.

[Morin & Pierrel, 1987] P. Morin et J.-M. Pierrel. PARTNER : un système de dialogue oral homme-machine. In *Actes du congrès COGNITIVA*, p. 354–361, Paris, 1987.

[Nerzic, 1993] P. Nerzic. *Erreurs et échecs dans le dialogue oral homme-machine*. Thèse de doctorat, Univerité de Rennes 1, janvier 1993.

[Nicolle & Saint-Dizier De Almeida, 1999a] A. Nicolle et V. Saint-Dizier De Almeida. *Analyse et Simulation de Conversations, De la théorie des actes de discours aux systèmes multiagents*, chap. Vers un modèle des interactions langagières, p. 133–169. L'Interdisciplinaire, Limonest, 1999.

[Nicolle & Saint-Dizier De Almeida, 1999b] A. Nicolle et V. Saint-Dizier De Almeida. Un modèle des interactions langagières personnes/machine. In *Actes de l'atelier la langue dans l'interaction personnes-machines, TALN'99*, p. 125–136, Cargèse, Corse, 12-17 Juillet 1999.

[Peckham, 1991] J. Peckham. Speech Understanding and Dialogue over the telephone: an overview of the ESPRIT SUNDIAL project. In *Actes d'Eurospeech*, Genève, 1991.

[Pernel, 1994] D. Pernel. *Gestion des buts multiples de l'utilisateur dans un dialogue homme-machine de recherche d'informations*. Thèse de doctorat, LIMSI-CNRS, Orsay, juin 1994.

[Pierrel, 1987] J.-M. Pierrel. *Dialogue oral homme-machine*. Hermès, Paris, 1987.

[Pitrat, 1995] J. Pitrat. *De la machine à l'intelligence*. Hermès, Paris, 1995.

[Prince & Ferrari, 1996] V. Prince et S. Ferrari. A Textual Clues Approach for Generating Metaphors as Explanations by an Intelligent Tutoring System. In *Proceedings of TALC96, Conference on Teaching And Language Corpora*, p. 217–232, Lancaster University, 9-12 Août 1996.

[Prince, 1990] V. Prince. EDGAR: Model and Method for Ambiguity Representation in NL-Understanding Systems. In *Proceedings of AINN'90*, p. 209–215, 1990.

[Pynte, 1988] J. Pynte. *Psychologie cognitive, modèles et méthodes*, chap. Les modèles de la compréhension du langage, p. 87–110. PUG, Grenoble, 1988.

[Quignard, 2000] M. Quignard. *Modélisation cognitive de l'argumentation dialoguée, Étude de dialogues d'élèves en résolution de problème de sciences physiques.* Thèse de doctorat, Grenoble, 28 janvier 2000.

[Richard, 1990] J.-F. Richard. *Les activités mentales, Comprendre, raisonner, trouver des solutions.* Armand Colin Édition, Paris, 1990.

[Rouillard & Caelen, 1998] J. Rouillard et J. Caelen. Étude du dialogue Homme-Machine en langue naturelle sur le Web pour une recherche documentaire. In *Apprentissage Homme-Machine, Actes de CAPS'98*, éd. K. Zreik, p. 99–119, Paris, 1998. Europia Productions.

[Rouillard, 1998] J. Rouillard. Contribution à l'étude du dialogue Homme-Machine à travers le Web. In *Actes de RÉCITAL'98*, éd. J. Lehuen, p. 127–134, Le Mans, 8-9 septembre 1998.

[Rouillard, 2000] J. Rouillard. *Hyperdialogue sur Internet, Le système HALPIN.* Thèse de doctorat, CLIPS - IMAG, Université Joseph Fourier, Grenoble 1, 11 janvier 2000.

[Sabah *et al.*, 1997] G. Sabah, J. Vivier, A. Vilnat, J.-M. Pierrel, L. Romary, et A. Nicolle. *Machine, langage et dialogue.* L'Harmattan, Paris, 1997.

[Sabah, 1988] G. Sabah. *L'intelligence artificielle et le langage, Représentations des connaissances*, volume 1. Hermès, Paris, 1988.

[Sabah, 1989] G. Sabah. *L'intelligence artificielle et le langage, Processus de compréhension*, volume 2. Hermès, Paris, 1989.

[Sabah, 1993] G. Sabah. *Modèles et concepts pour la science cognitive, hommage à J.-F. Le Ny*, chap. Vers une conscience artificielle ?, p. 207–222. PUG, Grenoble, 1993.

[Searle & Vanderveken, 1985] J. R. Searle et D. Vanderveken. *Foundations of Illocutionary Logic.* Cambridge University Press, Cambridge, 1985.

[Searle, 1972] J. Searle. *Les actes de langage, Essai de philosophie du langage.* Hermann, 1972.

[Searle, 1985] J. R. Searle. *L'intentionalité, Essai de philosophie des états mentaux.* Les Éditions de Minuit, Paris, 1985.

[Sperber & Wilson, 1989] D. Sperber et D. Wilson. *La pertience, communication et cognition.* Les Éditions de Minuit, Paris, 1989.

[Steele, 1990] G. L. Steele. *Common Lisp, The Language, Second Edition.* Digital Press, 1990.

[Tiberghien, 1997] G. Tiberghien. *La mémoire oubliée.* Mardaga, Sprimont, Belgique, 1997.

[Trichet, 1998] F. Trichet. *DSTM : un environnement de modélisation et d'opérationalisation de la démarche de résolution de problèmes d'un Système à Base de Connaissances.* Thèse de doctorat, Université de Nantes, 26 novembre 1998.

[Trognon & Brassac, 1992] A. Trognon et C. Brassac. L'enchaînement conversationnel. *Cahiers de Linguistique Française*, 13:76–107, 1992.

[Trognon, 1995] A. Trognon. Structures interlocutoires. *Cahiers de Linguistique Française*, (17):79–98, 1995.

[Trognon, 1996] A. Trognon. *Psychologie du dialogue homme-machine en langage naturel*, chap. L'interlocution des conversations et des dialogues personne-machine, p. 17–34. Europia Productions, Paris, 1996.

[Trognon, 1997a] A. Trognon. Conditions pyscholinguistiques de l'échange parlé. In *Le dialogique, Colloque international sur les formes philosophiques, littéraires, linguistiques et cognitives du dialogue*, Paris, 1997. Peter Lang.

[Trognon, 1997b] A. Trognon. *Conversation, interaction et fonctionnement cognitif*, chap. Conversation et raisonnements, p. 253–282. Presses Universitaires de Nancy, Nancy, 1997.

[Vanderveken, 1990] D. Vanderveken. *Meaning and Speech Acts*. University Press, Cambridge, 1990.

[Vanderveken, 1999] D. Vanderveken. *Analyse et Simulation de Conversations, De la théorie des actes de discours aux systèmes multiagents*, chap. La structure logique des dialogues intelligents, p. 61–100. L'Interdisciplinaire, Limonest, 1999.

[Vernant, 1997a] D. Vernant. Dialectique, forme dialoguale et dialogique. In *Le dialogique, Colloque international sur les formes philosophiques, littéraires, linguistiques et cognitives du dialogue*, p. 11–26, Paris, 1997. Peter Lang.

[Vernant, 1997b] D. Vernant. *Du discours à l'action*. PUF, 1997.

[Villaseñor & Caelen, 1998] L. Villaseñor et J. Caelen. Une Logique pour le Dialogue Coopératif Homme-Machine. In *Apprentissage Homme-Machine, Actes de CAPS'98*, éd. K. Zreik, p. 85–98, Paris, 1998. Europia Productions.

[Vilnat, 1997] A. Vilnat. *Machine, langage et dialogue*, chap. Quels processus pour les dialogues homme-machine ?, p. 129–182. L'Harmattan, 1997.

[Vivier, 1996] J. Vivier. *Psychologie du dialogue homme-machine en langage naturel*. Europia Productions, Paris, 1996.

[Vivier, 1997] J. Vivier. *Machine, langage et dialogue*, chap. Pour une psychologie du dialogue homme-machine, p. 73–127. Figures de l'interaction. L'Harmattan, Paris, 1997.

[Winograd, 1993] T. Winograd. Heidegger et la conception des systèmes informatiques. *Intellectica*, 2(17):51–78, 1993.

Table des figures

Liste des tableaux

Annexe A - Analyse du corpus de dialogue homme-machine établi lors de l'expérimentation du système AMI1 de juin 1998

0.1 Analyse des deux premières séances

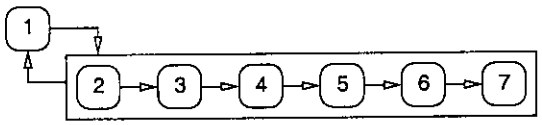
0.1.1 Première séance : analyses des dialogues D1-1 à D1-13

0.1.1.1 Dialogue D1-1

Dialogue

M1 : Service de messagerie, bonjour. Que puis-je faire pour vous ?
H1 : ben rien pourquoi ?
M2 : Venez-vous de me demander de lire les messages qui vous sont adressés ?
H2 : non
M3 : Venez-vous de me demander de laisser un message pour quelqu'un ?
H3 : non
M4 : Venez-vous de me demander de parler à une personne du laboratoire ?
H4 : non
M5 : Venez-vous de vous présenter ?
H5 : non
M6 : Venez-vous de désigner la personne que vous voulez contacter ?
H6 : non
M7 : Venez-vous de clore le dialogue ?
H7 : oui plutot ça

- Dialogue : complet et correct.
- Déroulement :
  - incompréhension totale initiale ;
  - mise en marche de la tactique dialogique n°2 ;
  - apprentissage d'un nouveau lexème ;
  - arrêt normal par le système.
- Structure :



- Commentaires :

1. Problème de la gestion du dialogue : les six questions pour parvenir à comprendre l'énoncé donnent un dialogue un peu lourd. Sans remettre en cause le principe des attentes, une solution consisterait à avoir des attentes générales et des sous-attentes. Par exemple, les attentes des tâches bien précises (lire les messages, laisser un message, contacter une personne) seraient des sous-attentes d'une attente plus générale (attente d'une demande de tâche parmi celles possibles). En cas d'incompréhension totale, les questions posées porteraient uniquement sur les attentes les plus générales.
2. Problème en compréhension : le système devrait dire qu'il n'a rien compris à l'énoncé initial de l'utilisateur pour que celui-ci se rende mieux compte de l'état dialogique courant.

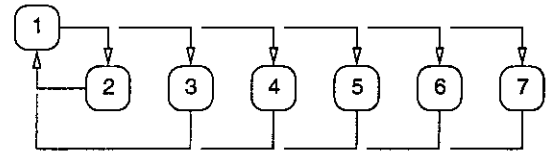
3. Erreur de conception : la SC Invalidation-Act (tactique dialogique n°2) doit avoir un intérêt supérieur à la SC Verification-Act (tactique dialogique n°2) afin de supprimer les possibilités correspondantes.

0.1.1.2 Dialogue D1-2

Dialogue

M1 : Service de messagerie, bonjour. Que puis-je faire pour vous ?
H1 : je voudrais le numéro de fax du secrétariat lium
M2 : Vous avez dit 'NUMéRO DE FAX DU SECRéTARIAT'.
M2 : Est-ce que "NuméRo De Fax Du SecréTariat" est un numéro de poste ?
H2 : oui, en quelque sorte
M3 : Vous avez dit 'NUMéRO DE FAX DU'.
M3 : Est-ce que "NuméRo De Fax Du" est un numéro de poste ?
H3 : non, je crois que vou avez mal compris
M4 : Vous avez dit 'FAX DU SECRéTARIAT LIUM'.
M4 : Est-ce que "Fax Du SecréTariat Lium" est un nom ?
H4 : non
M5 : Vous avez dit 'NUMéRO DE FAX'.
M5 : Est-ce que "NuméRo De Fax" est un numéro de poste ?
H5 : mais non !
M6 : Vous avez dit 'FAX DU SECRéTARIAT'.
M6 : Est-ce que "Fax Du SecréTariat" est un prénom ?
H6 : non, j'en ai marre. apparemment vous ne pouvez pas répondre à ma question !

- Dialogue : incomplet.
- Déroulement :
  - compréhension partielle en H1 ;
  - mise en marche de la tactique dialogique n°1 autant de fois qu'il y a des conflits d'interprétations ;
  - compréhension partielle en H6 ;
  - arrêt brutal par l'expérimentateur du fait du temps de réaction trop long du système.
- Structure :



- Commentaires :

1. Problème en compréhension : en H1 et en H6 il y a beaucoup trop d'interprétations hypothétiques de construites par l'analyseur (à cause des compréhensions partielles), d'où les temps de réactions importants et le grand nombre de questions. Ceci provient de l'analyseur qui découpe élément par élément tous les groupes lexicaux non reconnus de l'énoncé.

2. Problème de la gestion du dialogue : en H2 si l'expression « numéro de fax du secrétariat » est validé par l'utilisateur comme étant un numéro de poste, alors en M3 « numéro de fax du » est forcément un numéro de poste (idem en M5). Il s'agit donc de questions non cohérentes. La solution consiste à ce que lorsqu'un groupe est bien

identifié par l'utilisateur, aucune question ne doit porter sur des sous-groupes issus de ce groupe puisqu'il a été validé.

3. Problème de conception : les questions posées en M2, M3, M4, M5 et M6 ne sont pas très compréhensibles.

0.1.1.3 Dialogue D1-3

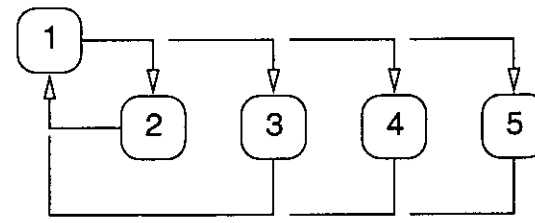
- Dialogue : non analysé en détail.
- Commentaires : problèmes identiques à ceux du dialogue D1-2.

0.1.1.4 Dialogue D1-4

Dialogue

M1: Service de messagerie, bonjour. Que puis-je faire pour vous ?
H1 : Me donner le numéro de téléphone de p.
M2 : Vous avez dit 'ME DONNER LE NUMéRO DE TéLéPHONE DE'.
M2 : Est-ce que "Me Donner Le NuméRo De TéLéPhone De" est un nom ?
H2 : non
M3 : Vous avez dit 'TéLéPHONE DE'.
M3 : Est-ce que "TéLéPhone De" est un nom ?
H3 : non
M4 : Vous avez dit 'TéLéPHONE'.
M4 : Est-ce que "TéLéPhone" est un prénom ?
H4 : non
M5 : Vous avez dit 'TéLéPHONE'.
M5 : Est-ce que "TéLéPhone" est un nom ?
H5 : non
M5 : Désolé, mais je ne connais pas cette personne. Au revoir.

- Dialogue : complet et incorrect.
- Déroulement :
  - compréhension partielle initiale ;
  - mise en marche de la tactique dialogique n°1 autant de fois qu'il y a de conflits d'interprétations ;
  - échec lors de la recherche dans l'annuaire de la personne appelée.
- Structure :



- Commentaires :

1. Problèmes de compréhension : « p. » a été compris comme désignant la personne appelante et la personne appelée a été comprise comme étant « téléphone de » (en posant une hypothèse) ; il s'agit d'un non-sens et d'une interprétation abusive. Pour éviter l'interprétation abusive, le système pourrait demander confirmation de la validité de l'hypothèse à l'utilisateur. Le non-sens est dû à une mauvaise amorce linguistique de l'identification de la personne appelante.

2. Problème de conception : en M5, il faudrait que le système indique l'identité de la personne qu'il ne connaît pas.

3. Problème de conception : les questions en M2, M3, M4, et M5 ne sont pas très compréhensibles.

0.1.1.5 Dialogue D1-5

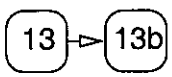
- Dialogue : identique au dialogue D1-4.

0.1.1.6 Dialogue D1-6

Dialogue

M13 : Service de messagerie, bonjour. Que puis-je faire pour vous ?
H13 : le poste de p.
M13 : Désolé, mais je ne connais pas cette personne. Au revoir.

- Dialogue : complet et incorrect.
- Déroulement :
  - compréhension partielle initiale ;
  - création d'une hypothèse ;
  - échec lors de la recherche dans l'annuaire de la personne appelée.
- Structure :



- Commentaires :

1. Problème de conception : des objets n'ont pas été supprimés du dialogue précédent, d'où la mise en marche intempestive de la SC Propagation-Validation et la suppression d'objets nouvellement créés (par la SC Propagation-Suppression).

2. Problème de compréhension : il y a un non-sens (« p. » a été compris comme la personne appelante) et une interprétation abusive au sujet de la personne appelée. Le non-sens est dû à une mauvaise amorce linguistique de l'identification de la personne appelante. L'hypothèse qui a conduit à l'interprétation abusive devrait être validée par l'utilisateur.

3. Problème de compréhension : en H13, il y a deux interprétations hypothétiques qui peuvent répondre à la même attente. Le choix est ici aléatoire. Pour remédier au problème, il faut favoriser l'interprétation la moins hypothétique et la plus couvrante. Si on ne peut pas faire de choix, il faut poser la question à l'utilisateur.

4. Problème de conception : en M13, il faudrait que le système indique l'identité de la personne qu'il ne connaît pas.

0.1.1.7 Dialogue D1-7

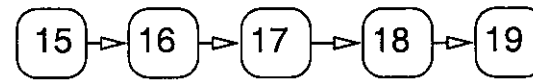
- Dialogue : complet et incorrect (identique au dialogue D1-6).

0.1.1.8 Dialogue D1-8

Dialogue

M15 : Service de messagerie, bonjour. Que puis-je faire pour vous ?
H15 : Je voudrais parler à M.
M16 : Pourrais-je connaître l'identité de votre correspondant ?
H16 : M. V.
M17 : Pourrais-je connaître l'identité de votre correspondant ?
H17 : M. V.
M18 : Pourrais-je connaître l'identité de votre correspondant ?
H18 : non
M19 : Pourrais-je connaître l'identité de votre correspondant ?
H19 : au revoir !

- Dialogue : complet et incorrect.
- Déroulement :
  - compréhension initiale ;
  - échec de la recherche dans l'annuaire de la personne appelante ;
  - interprétation incorrecte de H16 et H17 ;
  - arrêt de l'utilisateur.
- Structure :



- Commentaires :

1. Problème de conception : des objets n'ont pas été supprimés du dialogue précédent, ce qui a pour effet d'invalider une interprétation de l'énoncé initial.
2. Problème de compréhension : il y a un non-sens (« m. » est compris comme désignant le prénom de la personne appelante) et une interprétation abusive sur le nom de l'appelant (hypothèse portant sur le début de la phrase). Cette interprétation incorrecte est due à une mauvaise amorce linguistique de l'identification de la personne appelante. Les questions M16, M17, M18 et M19 sont aberrantes à cause du non-sens.
3. Problème de compréhension : les énoncés H16 et H17 ne sont pas compris, car dans l'amorce linguistique la suite <prénom> <nom> désigne la personne appelante et non la personne appelée, alors que c'est cette dernière qui est attendue par le modèle de la tâche à cette étape du déroulement normal.

0.1.1.9 Dialogue D1-9

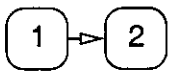
Dialogue

M1 : Service de messagerie, bonjour. Que puis-je faire pour vous ?
H1 : Je voudrais parler à M.
M2 : M. V. est actuellement en train de prendre son déjeuner. Désirez-vous que je lui tramsmette un message ?
H2 : non, je voulais lui parler en personne.

- Dialogue : incomplet.
- Déroulement :
  - compréhension initiale ;
  - succès de la recherche dans l'annuaire de la personne appelée ;

- échec de la recherche dans l'annuaire de la personne appelante ;
- arrêt sur erreur logicielle.

- Structure :



- Commentaires :

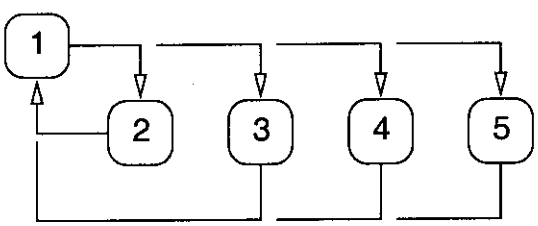
1. Problème de compréhension : interprétation abusive de la personne appelante à cause de la mauvaise amorce linguistique. Pour remédier à cela, il faut demander à l'utilisateur la validité de l'hypothèse.
2. Problème de gestion du dialogue : la demande en H2 n'a pas été prise en compte. Ceci est dû au fait qu'à ce moment-là, le modèle de la tâche n'a que deux attentes (une confirmation et une infirmation) et que toute offre ne correspondant pas à ces attentes n'est pas prise en compte. Ceci pose le problème de l'initiative de l'utilisateur. La solution non généralisable consiste à ajouter des attentes lors de cette étape dans le modèle de la tâche.
3. Erreur logicielle : la virgule dans l'énoncé H2 n'a pas été supprimée ce qui a provoqué cette erreur.

0.1.1.10 Dialogue D1-10

Dialogue

M1 : Service de messagerie, bonjour. Que puis-je faire pour vous ?
H1 : je voudrais savoir s'il y a des messages pour moi
M2 : Vous avez dit 'JE VOUDRAIS SAVOIR S IL Y A'.
M2 : Est-ce que "Je Voudrais Savoir S Il Y A" est de lire vos messages ?
H2 : oui
M3 : Vous avez dit 'JE VOUDRAIS SAVOIR S IL Y A'.
M3 : Est-ce que "Je Voudrais Savoir S Il Y A" est de lire vos messages ?
H3 : oui !
M4 : Vous avez dit 'S IL Y A'.
M4 : Est-ce que "S Il Y A" est de lire vos messages ?
H4 : oui
M5 : Vous avez dit 'Y A'.
M5 : Est-ce que "Y A" est de lire vos messages ?
H5 : non
M5 : Désolé, mais il n'y a aucun message en attente. Au revoir.

- Dialogue : complet et correct.
- Déroulement :
  - compréhension partielle initiale ;
  - mise en marche de la tactique dialogique n°1 autant de fois qu'il y a de conflits d'interprétations ;
  - arrêt normal par le système.
- Structure :



- Commentaires :

1. Problème de compréhension : on a deux fois la même question en M2 et M3 à cause de plusieurs interprétations concurrentes. Pour remédier à cela, la validation ou l'invalidation par l'utilisateur d'une hypothèse devrait être répercutée sur les autres interprétations et non seulement sur l'interprétation falsifiée.

2. Problème de compréhension : en H1 il y a plusieurs interprétations qui correspondent à une seule attente. Le choix est ici arbitraire (voir le troisième commentaire du dialogue D1-6).

3. Problème de gestion du dialogue : les questions M4 et M5 ne sont pas pertinentes en vue de la réponse en H2 (voir le deuxième commentaire du dialogue D1-2).

4. Problème de conception : les questions en M2, M3, M4, et M5 ne sont pas très compréhensibles.

0.1.1.11 Dialogue D1-11

Dialogue

M6 : Service de messagerie, bonjour. Que puis-je faire pour vous ?

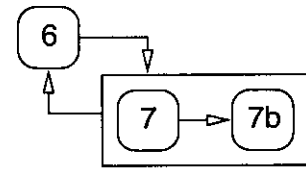
H6 : messages

M7 : Venez-vous de me demander de lire les messages qui vous sont adressés ?

H7 : oui

M7 : Désolé, mais il n'y a aucun message en attente. Au revoir.

- Dialogue : complet et correct.
- Déroulement :
  - incompréhension totale initiale ;
  - mise en marche de la tactique dialogique n°2 ;
  - apprentissage d'un nouveau lexème ;
  - arrêt normal par le système.
- Structure :



- Commentaires :

1. Problème de la gestion du dialogue : voir les deux premiers commentaires du dialogue D1-1.

0.1.1.12 Dialogue D1-12

Dialogue

M1 : Service de messagerie, bonjour. Que puis-je faire pour vous ?

H1 : je veux parler à P.

M2 : P. T. est actuellement en train de prendre son déjeuner. Désirez-vous que je lui transmette un message ?

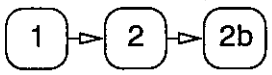
H2 : non, je voulais parler à P. D.

M2 : Je suis désolé mais il n'y a personne que je puisse contacter. Au revoir.

- Dialogue : complet et incorrect.
- Déroulement :

- compréhension initiale ;
- succès de la recherche dans l'annuaire de la personne appelée ;
- échec de la recherche dans l'annuaire de la personne appelante ;
- arrêt normal par le système.

- Structure :



- Commentaires :

1. Problème de compréhension : en H1 l'analyseur a produit une interprétation incorrecte sur l'identité de la personne appelante à cause d'une mauvaise amorce linguistique. Ce problème pourrait être résolu si toute hypothèse était validée par l'utilisateur avant d'être utilisée par le modèle de la tâche.
2. Problème de gestion du dialogue : la demande en H2 n'a pas été prise en compte (voir le deuxième commentaire du dialogue D1-9).

0.1.1.13 Dialogue D1-13

- Dialogue : complet et incorrect (identique au dialogue D1-12).

0.1.2 Seconde séance : analyses des dialogues D2-1 à D2-15

0.1.2.1 Dialogue D2-1

Dialogue

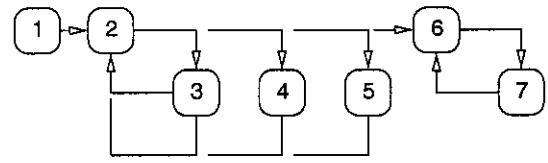
M1 : Service de messagerie, bonjour. Que puis-je faire pour vous ?
H1 : je souhaiterais laisser un message au responsable du laboratoire
M2 : Pourrais-je savoir qui vous êtes ?
H2 : je suis j. C.
M3 : Vous avez dit 'J. C.'.
M3 : Est-ce que "J. C." est un prénom ?
H3 : non
M4 : Vous avez dit 'J.'.
M4 : Est-ce que "J." est un nom ?
H4 : non
M5 : Vous avez dit 'J.'.
M5 : Est-ce que "J." est un prénom ?
H5 : oui
M6 : M. V. est actuellement en cours. Désirez-vous que je lui tramsmette un message ?
H6 : savez-vous quand pourrais-je le joindre ?
M7 : Vous avez dit 'JOINDRE'.
M7 : Est-ce que "Joindre" est un nom ?
H7 : non

- Dialogue : incomplet.
- Déroulement :

- compréhension initiale correcte en H1 ;
- incompréhension partielle en H2 ;
- mise en marche de la tactique dialogique n°1 autant de fois qu'il y a de conflits ;
- apprentissage du terme lexical « j. » comme <prenom> ;
- échec de la recherche dans l'annuaire de la personne appelante ;
- compréhension partielle en H6 ;

- mise en marche de la tactique dialogique n°1 ;
- arrêt du système, car il n'y a plus de SC instanciée dans l'agenda.

- Structure :



- Commentaires :

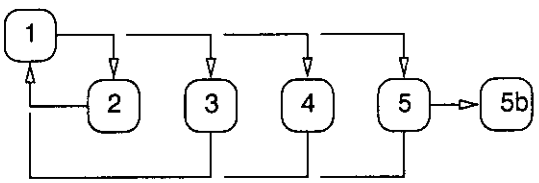
1. Problème de conception : l'apprentissage du lexème « c. » comme <nom> n'a pas marché (car l'attente a été satisfaite avec une interprétation incomplète ne prenant pas en compte le dernier terme de l'énoncé). Une solution consiste à dire que quand une attente peut être satisfaite à la fois par deux ou même plusieurs offres, il faut utiliser l'offre appartenant à l'interprétation la moins hypothétique et la plus couvrante. Il faut remarquer que ce problème existe très souvent mais n'apparaît que dans certains cas.
2. Problème de gestion du dialogue : la demande en H6 n'a pas été prise en compte (voir le deuxième commentaire du dialogue D1-9).
3. Problème de conception : les questions en M3, M4, M5, et M6 ne sont pas très compréhensibles.
4. Problème de conception : le système s'arrête car il n'y a aucune attente de satisfaite et donc aucune SC ne peut être lancée.

0.1.2.2 Dialogue D2-2

Dialogue

M1 : Service de messagerie, bonjour. Que puis-je faire pour vous ?
H1 : je voudrais contacter un thésard
M2 : Vous avez dit 'UN THéSARD'.
M2 : Est-ce que "Un ThéSard" est un nom ?
H2 : non
M3 : Vous avez dit 'UN'.
M3 : Est-ce que "Un" est un prénom ?
H3 : non
M4 : Vous avez dit 'UN'.
M4 : Est-ce que "Un" est un nom ?
H4 : non
M5 : Vous avez dit 'UN'.
M5 : Est-ce que "Un" est avant le nom ?
H5 : non
M5 : Désolé, mais je ne connais pas cette personne. Au revoir.

- Dialogue : complet et incorrect.
- Déroulement :
  - compréhension initiale partielle ;
  - mise en marche de la tactique dialogique n°1 autant de fois qu'il y a de conflits ;
  - échec de la recherche dans l'annuaire de la personne appelée ;
  - arrêt normal par le système.
- Structure :



- Commentaires :

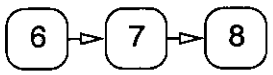
1. Problème de compréhension : l'expression « un thésard » a été comprise comme étant un prénom. Pour éviter ce problème, il faut demander à l'utilisateur de valider l'hypothèse.
2. Problème de conception : les questions en M2, M3, M4, et M5 ne sont pas très compréhensibles.
3. Problème de conception : en M5, il faudrait que le système indique l'identité de la personne qu'il ne connaît pas.

0.1.2.3 Dialogue D2-3

Dialogue

M6 : Service de messagerie, bonjour. Que puis-je faire pour vous ?
H6 : salut, pourrais parler à P.
M7 : P. L. est actuellement en cours. Désirez-vous que je lui tramsmette un message ?
H7 : oui, je serais gentil
M8 : Veuillez maintenant taper votre message.
H8 : bonjour P., pourrais-tu me rappeler rapidement, j'ai besoin de te parler.

- Dialogue : complet et correct.
- Déroulement :
  - compréhension initiale correcte ;
  - succès de la recherche dans l'annuaire de la personne appelée ;
  - échec de la recherche dans l'annuaire de la personne appelante ;
  - arrêt normal par le système.
- Structure :



- Commentaires :

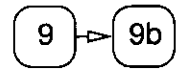
1. Problème de compréhension : une interprétation hypothétique a été faite sur l'identité de la personne appelante. Ceci est dû à une mauvaise amorce en ce qui concerne cette identification. Ce type de problème devrait être supprimé en demandant systématiquement la validité de l'hypothèse à l'utilisateur.
2. Problème de conception : la clôture du dialogue est trop sèche. Il faut modifier le modèle de la tâche pour ajouter une dernière étape de clôture après la prise d'un message.

0.1.2.4 Dialogue D2-4

Dialogue

M9 : Service de messagerie, bonjour. Que puis-je faire pour vous ?
H9 : bonjour, je voudrais lire les messages qui me sont destinés
M9 : Désolé, mais il n'y a aucun message pour vous. Au revoir.

- Dialogue : complet et incorrect.
- Déroulement :
  - compréhension initiale partielle ;
  - échec de la recherche dans l'annuaire de la personne appelante ;
  - échec de la recherche de messages ;
  - arrêt normal par le système.
- Structure :



- Commentaires :

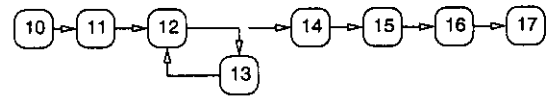
1. Problème de conception : la clôture du dialogue est trop sèche (voir le deuxième commentaire du dialogue D2-3).
2. Problème de compréhension : l'échec de la recherche de messages est dû aux faits qu'il y a une mauvaise interprétation en H9 (portant sur l'identité de l'appelant) et que le message créé durant le dialogue D2-3 possède un champ origine vide (car la personne appelante n'avait pas été identifiée correctement). Ce type de problème devrait être supprimé en demandant systématiquement la validité d'une hypothèse à l'utilisateur.

0.1.2.5 Dialogue D2-5

Dialogue

M10 : Service de messagerie, bonjour. Que puis-je faire pour vous ?
H10 : je suis J. M.
M11 : Pourrais-je connaître l'identité de votre correspondant ?
H11 : il s'appelle c. p.
M12 : Pourrais-je connaître l'identité de votre correspondant ?
H12 : c.
M13 : Venez-vous de désigner la personne que vous voulez contacter ?
H13 : oui
M14 : Pourrais-je connaître l'identité de votre correspondant ?
H14 : p. b.
M15 : Pourrais-je connaître l'identité de votre correspondant ?
H15 : non
M16 : Pourrais-je connaître l'identité de votre correspondant ?
H16 : oui
M17 : Pourrais-je connaître l'identité de votre correspondant ?
H17 : au revoir

- Dialogue : complet et incorrect.
- Déroulement :
  - compréhension initiale correcte ;
  - mauvaises interprétations en H11, H12 et H14 ;
  - arrêt par l'utilisateur.
- Structure :



- Commentaires :

1. Problème de programmation : en H10 deux interprétations sont identiques.

2. Problème de compréhension : en H10, trois interprétations peuvent répondre à une même attente. Le choix est ici fait de façon arbitraire. Une solution consiste à dire que quand une attente peut être satisfaite à la fois par deux ou même plusieurs offres, il faut utiliser l'offre appartenant à l'interprétation la moins hypothétique et la plus couvrante. S'il y a équivalence, il faut demander à l'utilisateur.

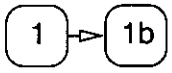
3. Problème de compréhension : les mauvaises interprétations en H11, H12 et H14 ainsi que l'apprentissage incorrect du terme « c. » comme acte de langage <ident-appele> (avec l'UMI incidente n°13), sont dues à une mauvaise amorce linguistique (voir le troisième commentaire du dialogue D1-8).

0.1.2.6 Dialogue D2-6

Dialogue

M1 : Service de messagerie, bonjour. Que puis-je faire pour vous ?
H1 : je voudrais consulter ma messagerie
M1 : Désolé, mais je ne connais pas cette personne. Au revoir.

- Dialogue : complet et incorrect.
- Déroulement :
  - mauvaise interprétation initiale ;
  - échec de la recherche dans l'annuaire de la personne appelée ;
  - arrêt normal par le système.
- Structure :



- Commentaires :

1. Problème de conception : en M1, il faudrait que le système indique l'identité de la personne qu'il ne connaît pas.

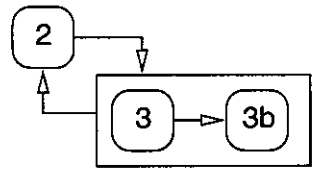
2. Problème de compréhension : l'interprétation faite en H1 est incorrecte (elle porte sur le nom de la personne appelée). Pour remédier à cela, il faut demander à l'utilisateur de valider l'hypothèse.

0.1.2.7 Dialogue D2-7

Dialogue

M2 : Service de messagerie, bonjour. Que puis-je faire pour vous ?
H2 : bonjour, j'aimerais contacter M. T.
[« M. » est une abréviation de « monsieur »]
M3 : Vous avez dit 'M'.
M3 : Est-ce que "M" est un nom ?
H3 : non
M3 : Désolé, mais je ne connais pas cette personne. Au revoir.

- Dialogue : complet et incorrect.
- Déroulement :
  - compréhension partielle initiale ;
  - mise en marche de la tactique dialogique n°1 ;
  - mauvaise interprétation du terme « M » ;
  - échec de la recherche dans l'annuaire de la personne appelée ;
  - arrêt normal par le système.
- Structure :



- Commentaires :

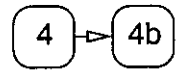
1. Problème de compréhension : l'interprétation faite est incorrecte (« M » est compris comme étant un prénom). Il faudrait faire une demande de confirmation de l'hypothèse à l'utilisateur. L'unité lexicale « M. » doit être également ajoutée à l'amorce.
2. Problème de conception : en M3, il faudrait que le système indique l'identité de la personne qu'il ne connaît pas.
3. Problème de conception : la question posée en M3 n'est pas très compréhensible.

0.1.2.8 Dialogue D2-8

Dialogue

M4 : Service de messagerie, bonjour. Que puis-je faire pour vous ?
H4 : je voudrais lire les messages qui me sont adressés
M4 : Désolé, mais il n'y a aucun message en attente. Au revoir.

- Dialogue : complet et correct.
- Déroulement :
  - compréhension initiale correcte ;
  - arrêt normal par le système.
- Structure :



- Commentaires :

1. Problème de compréhension : en H4 deux interprétations peuvent répondre à la même attente (voir le deuxième commentaire du dialogue D2-5).

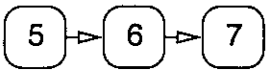
0.1.2.9 Dialogue D2-9

Dialogue

M5 : Service de messagerie, bonjour. Que puis-je faire pour vous ?
H5 : est-il possible de parler à P. ?
M6 : P. J. est actuellement en cours. Désirez-vous que je lui tramsmette un message ?

H6 : oui, pourquoi pas ?
M7 : Veuillez maintenant taper votre message.
H7 : bon, pierre rappelle moi dès que possible, c'est j.

- Dialogue : complet et correct.
- Déroulement :
  - compréhension initiale correcte ;
  - arrêt normal par le système.
- Structure :



- Commentaires :

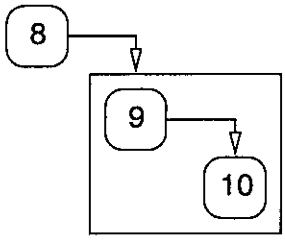
1. Problème de conception : la clôture du dialogue est trop sèche (voir le deuxième commentaire du dialogue D2-3).
2. Problème de compréhension : le message créé possède un champ origine vide (l'identification de la personne appelante est issue d'une interprétation hypothétique incorrecte). Ce problème a pour origine une mauvaise amorce en ce qui concerne l'identification de la personne appelante. De manière générale, ce type de problème peut être résolu si le système demande une confirmation de la validité de l'hypothèse à l'utilisateur.

0.1.2.10 Dialogue D2-10

Dialogue

M8 : Service de messagerie, bonjour. Que puis-je faire pour vous ?
H8 : un message pour s.
M9 : Vous avez dit 'UN MESSAGE POUR'.
M9 : Est-ce que "Un Message Pour" est une présentation ?
H9 : pourquoi ?
M10 : Venez-vous de me répondre par l'affirmative ? (oui/non)
H10 : non

- Dialogue : incomplet et incorrect.
- Déroulement :
  - compréhension initiale partielle ;
  - mise en marche de la tactique dialogique n°1 ;
  - arrêt sur erreur logicielle.
- Structure :



- Commentaires :

1. Problème de programmation trouvé (un granule de compréhension créé par la SC INVALIDATION-CONFIRMATION n'a pas de lien).
2. Problème de compréhension : il y a trop d'interprétations hypothétiques qui sont faites en H8 (voir le premier commentaire du dialogue D1-2).

3. Problème de conception : la question posée en M9 n'est pas très compréhensible.

0.1.2.11 Dialogue D2-11

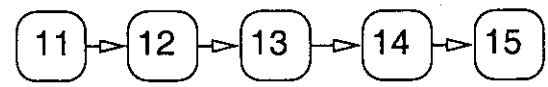
- Dialogue : incomplet (identique au dialogue D1-2).

0.1.2.12 Dialogue D2-12

Dialogue

M11 : Service de messagerie, bonjour. Que puis-je faire pour vous ?
H11 : est-ce que M. A. appartient bien à ce laboratoire
M12 : Pourrais-je connaître l'identité de votre correspondant ?
H12 : M. A.
M13 : Pourrais-je connaître l'identité de votre correspondant ?
H13 : A. R.
M14 : Pourrais-je connaître l'identité de votre correspondant ?
H14 : oui bien sur
M15 : Pourrais-je connaître l'identité de votre correspondant ?
H15 : au revoir

- Dialogue : complet et incorrect.
- Déroulement :
  - compréhension partielle initiale ;
  - mauvaises interprétations en H12 et H13 ;
  - arrêt par l'utilisateur.
- Structure :



- Commentaires :

1. Erreur de programmation : des objets d'un dialogue précédent n'ont pas été supprimés, ce qui provoque la suppression d'une des interprétations faites en H11. Cette suppression a pour conséquence de ne pas poser le problème du choix d'une interprétation parmi celles possibles (deux interprétations concurrentes sont faites en H11).

2. Problème de compréhension : en H11, le système fait une mauvaise interprétation de l'identité de la personne appelante. Il faudrait que celle-ci soit confirmée par l'utilisateur. Ceci provient d'une mauvaise amorce linguistique en ce qui concerne l'identification de la personne appelante.

3. Problème de compréhension : les énoncés H12, H13 et H14 ne sont pas compris (voir le troisième commentaire du dialogue D1-8).

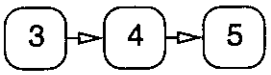
0.1.2.13 Dialogue D2-13

- Dialogue : incomplet (identique au dialogue D2-12).

0.1.2.14 Dialogue D2-14

- Dialogue : complet et correct (identique au dialogue D1-11).

0.1.2.15 Dialogue D2-15

Dialogue	
M3 : Service de messagerie, bonjour. Que puis-je faire pour vous ? H3 : je voudrais laisser un message M4 : Pourrais-je connaître l'identité de votre correspondant ? H4 : bien sur, P. L. M5 : Pourrais-je connaître l'identité de votre correspondant ? H5 : au revoir	- Dialogue : complet et incorrect. - Déroulement : • compréhension initiale correcte ; • mauvaise interprétation en H4 ; • arrêt par l'utilisateur. - Structure : 

- Commentaires :

1. Problème de compréhension : l'énoncé H4 n'est pas compris, car l'attente provenant du modèle de la tâche ne correspond pas à l'offre issue de l'analyseur (voir le troisième commentaire du dialogue D1-8).

0.2 Synthèse

0.2.1 Typologie des problèmes rencontrés

Le critère de classification des problèmes est celui de l'origine d'un dysfonctionnement et non les conséquences apparentes de ce dysfonctionnement. La typologie des problèmes recensés est la suivante :

- les problèmes purement informatiques : en général ceux-ci provoquent un arrêt anormal et brutal du système (ils ne sont pas repris dans la synthèse) ;
- les problèmes de conception : ceux-ci induisent un fonctionnement incorrect du système ;
- les problèmes conceptuels (mauvaises modélisations des phénomènes étudiés) :
  - en compréhension (analyse pragmatique des énoncés) : les énoncés de l'utilisateur ne sont pas interprétés correctement ;
  - en gestion du dialogue (tactiques dialogiques) : le dialogue n'est pas assez souple et naturel.

0.2.2 Récapitulatif des problèmes rencontrés

0.2.2.1 Problèmes de conception

- Les questions posées ne sont pas toujours claires, notamment dès qu'il s'agit des questions posées sur les granules de compréhension (voir les dialogues D1-2, D1-4, D1-10, D2-1, D2-2, D2-7 et D2-10). Cette partie de la génération peut être améliorée.

- Le système doit indiquer l'identité de la personne qu'il ne connaît pas (voir les dialogues D1-4, D2-2, D2-6 et D2-7).

- Après la prise d'un message, la clôture du dialogue est trop brutale (voir les dialogues D2-3, D2-4 et D2-9). Il faut ajouter une étape de clôture dans le graphe de la tâche.

- Certains objets créés à un certain moment par l'interpréteur n'ont pas été tous détruits. Ils ont donc interféré avec d'autres objets créés après, en provoquant des suppressions d'interprétations plus récentes (voir les dialogues D1-6, D1-8 et D2-12).

0.2.2.2 Problèmes en compréhension

- Dans le cas d'une incompréhension totale, le système doit indiquer qu'il n'a rien compris.

- En cas de compréhension partielle, l'analyseur peut être amené à construire beaucoup d'interprétations hypothétiques (voir les dialogues D1-2 et D2-10). Ceci provient de ce qu'il découpe élément par élément tous les groupes lexicaux non reconnus de l'énoncé. Un autre choix peut consister au contraire à faire ce découpage qu'en dernier ressort, lorsque ces groupes n'ont pas été reconnus par l'utilisateur.

- Le problème de la cohérence des questions portant sur les sous-groupes d'un groupe lexical non reconnu (voir les dialogues D1-2 et D1-10) n'existe pas s'il n'y a pas de découpage de ce groupe. Si le groupe doit quand même être découpé, il faut propager la validation dun groupe parmi ses sous-groupes. Ceci n'est valable que dans le cas d'une validation. Dans le cas contraire, il y aura autant de questions de posées.

- L'identification de la personne appelante a posé beaucoup de problèmes (voir les dialogues D1-4, D1-6, D1-8, D1-9, D1-12, D2-3, D2-5, D2-9 et D2-12). L'amorce linguistique doit être modifiée en conséquence.

- Dès que le système fait une hypothèse, il faut qu'elle soit validée par l'utilisateur avant d'être rendue utilisable par le modèle de la tâche (voir les dialogues D1-4, D1-6, D1-9, D1-12, D2-2, D2-3, D2-4, D2-6, D2-7, D2-9 et D2-12). La validation ou l'invalidation d'une hypothèse doit être répercutée sur les autres interprétations et non seulement sur l'interprétation falsifiée (voir le dialogue D1-10).

- Lorsque l'analyseur a construit plusieurs offres qui peuvent répondre à une même attente, il faut faire un choix (voir les dialogues D1-6, D1-10, D2-1, D2-5 et D2-8). Ce problème est celui de la polysémie. À notre sens, il faut favoriser l'interprétation qui est à la fois la moins hypothétique et celle qui prend en compte le plus grand nombre de termes lexicaux. En cas d'égalité, il faut poser la question à l'utilisateur.

0.2.2.3 Problèmes en gestion du dialogue

- Dans le cas d'une incompréhension totale, le nombre de questions posées dépend du nombre d'actes de dialogue attendus (voir les dialogues D1-1 et D1-11). Cela peut engendrer un dialogue lourd (comme le dialogue D1-1). Une solution consiste à avoir des hiérarchies d'attentes. Cela permettrait non pas de limiter le nombre d'attentes mais de limiter le nombre de questions posées en imposant un ordre d'intérêt. En cas d'incompréhension, les questions porteraient d'abord sur les attentes les plus générales. Par exemple, les attentes d'une tâche à effectuer (lire les messages, laisser un message, contacter une personne) seraient des sous-attentes d'une attente plus générale (attente d'une demande de tâche parmi celles qui sont possibles).

- Le problème de la prise en compte des offres non attendues a été posé (voir les dialogues D1-9, D1-12, D2-1 et D2-15). Cela survient lorsque l'énoncé de l'utilisateur n'a pas été prévu à ce moment dans le modèle de la tâche. Une solution consiste à ajouter des attentes lors de ces étapes dans le modèle de la tâche. Cette solution n'est pas entièrement satisfaisante puisqu'elle n'est pas généralisable.

Annexe B - Listings des fichiers sources de AMI2

Listing – utiles/charger.lsp

```
;; ###############################################################################################################
;; ### AMI - CHARGER.LSP - Lemeunier 25/02/99 (Thierry.Lemeunier@lium.univ-lemans.fr)
;; ###############################################################################################################

;; ---------------------------------------------------------------------------------------------------------------
;; Initialisation des variables spéciales conditionnant le comportement au chargement

;; "Commentez / décommentez" la ligne suivante selon le mode de fonctionnement (mode serveur ou mode interactif)
;; (setf *features* (cons ':SERVER *features*))

(defvar *niveau-debug* 1 "Niveau de débogage : permet d'afficher plus ou moins de messages au chargement")

(case *niveau-debug*

  (1 (setf *load-verbose* t             ;; Des messages aux chargements des fichiers
           *compile-verbose* t))        ;; Des messages à la compilation des fichiers

  (otherwise (setf *load-verbose* nil   ;; Pas de messages de chargement
                   *compile-verbose* nil))) ;; Pas de messages de compilation

;; ---------------------------------------------------------------------------------------------------------------
;; Chargement des fichiers

(load "utiles/fonctions.lsp")           ;; Fonctions utiles
(load "utiles/date.lsp")          ;; Chargement des fonctions liées au calcul des dates
(load "utiles/unites.lsp")              ;; Les objets nommés
(load "utiles/noms-fichiers.lsp")       ;; Les noms des fichiers de communication

#+(and clisp (not server))
(load "utiles/xterm.lsp")               ;; Le package pour manipuler des terminaux textes sous Unix

(load "utiles/blackboard/black.lsp")          ;; Le blackboard de base
(load "utiles/blackboard/black-rec.lsp")      ;; Le blackboard récursif
(load "utiles/blackboard/gramm.lsp")          ;; La grammaire d'écriture des KSOURCES
(load "utiles/blackboard/gramm-rec.lsp")      ;; La grammaire d'écriture des KSOURCES pour le blackboard récursif

(load "activite-dialogique/dialogueur.lsp")   ;; Implémentation du moteur de dialogue
(load "activite-dialogique/umm.lsp")          ;; Définition de l'umm et d'une arborescence
(load "activite-dialogique/umms-dial.lsp") ;; Chargement des umms liées à l'activité dialogique
(load "activite-dialogique/tache-dialogique.lsp") ;; Chargement de l'expertise dialogique

(load "activite-langagiere/umms-lang.lsp")    ;; Chargement des umms liées à l'activité langagière
(load "activite-langagiere/analyseur.lsp")    ;; Chargement de l'expertise d'analyse
(load "activite-langagiere/conflit-lexeme.lsp") ;; Chargement de l'expertise d'analyse (ambiguites lexicales)
(load "activite-langagiere/gestion-incomprehension.lsp") ;; Chargement de l'expertise d'analyse (incompréhension sémantique)
(load "activite-langagiere/interpreteur.lsp") ;; Chargement de l'expertise d'interprétation

(load "activite-langagiere/generateur.lsp")   ;; Chargement de l'expertise de génération
(load "activite-dialogique/intention.lsp")    ;; Chargement de l'expertise en intentions de communication

(load "activite-applicative/umms-appli.lsp")  ;; Chargement des umms liées à l'application
(load "activite-applicative/tache0.lsp")      ;; Chargement de l'expertise de l'application
(load "activite-applicative/gestion-message.lsp") ;; Chargement de fonctions de chargement et de sauvegarde des messages

;; ---------------------------------------------------------------------------------------------------------------
```

Listing: utiles/fonctions.lsp

```
;; ##########################################################################################################################
;; ### AMI - FONCTIONS.LSP - Lemeunier 02/12/99 (Thierry.Lemeunier@lium.univ-lemans.fr)
;; ##########################################################################################################################

;; --------------------------------------------------------------------------------------------------------------------------

(defun ?? (objet)
  "Retourner sur la sortie standart la description d'un objet"
  (describe objet))

;; --------------------------------------------------------------------------------------------------------------------------

(defun separation (taille)
  "Retourner une chaine de caractères de longueur taille constitué du caractère '-'"
  (make-string taille :initial-element #\-))

;; --------------------------------------------------------------------------------------------------------------------------

(defun rechercher-positions (sous-groupe groupe &optional (pos 0))
  "Retourner les positions d'un sous-groupe dans un groupe"
  (let ((posi (search sous-groupe groupe :test #'equalp)))
    (cond ((null groupe) nil)
    (posi (cons (+ pos posi)
          (rechercher-positions sous-groupe
             (subseq groupe (+ posi (length sous-groupe)))
             (+ pos (length sous-groupe) posi)))))))

;; --------------------------------------------------------------------------------------------------------------------------

(defun my-sort (sequence predicat &key key)
  "Trier par bulles une liste d'éléments selon un certain prédicat sur une cle"
  (loop for j from (length sequence) downto 1 do
  (loop for i from 0 to (- j 2) do
       (let ((premier (if key (funcall key (elt sequence i)) (elt sequence i)))
       (second (if key (funcall key (elt sequence (+ i 1))) (elt sequence (+ i 1)))))
    (unless (apply predicat (list premier second))
      (setf sequence (append
          (subseq sequence 0 i)
          (list (elt sequence (+ i 1)) (elt sequence i))
          (subseq sequence (+ i 2))))))))
  sequence)

;; --------------------------------------------------------------------------------------------------------------------------

(defun my-find-if (predicat sequence &key key)
  "Retourner la liste de tous les éléments de la séquence vérifiant un prédicat donné sur une clé"
  (cond ((null sequence) nil)
  ((funcall predicat (if key (funcall key (car sequence)) (car sequence)))
        (cons (car sequence) (my-find-if predicat (cdr sequence) :key key)))
  (t (my-find-if predicat (cdr sequence) :key key))))

;; --------------------------------------------------------------------------------------------------------------------------

(defun my-remove (liste-items sequence)
  "Enlever les items d'une séquence"
  (loop for item in liste-items do (setq sequence (remove item sequence))) sequence)

;; --------------------------------------------------------------------------------------------------------------------------

(defun partitionne-liste (liste &key key)
  "Partitionner une liste en sous-listes d'éléments vérifiant la même clé"
  (let ((liste-freres
   (my-find-if #'(lambda (x)
      (equal (if key (funcall key x) x) (if key (funcall key (car liste)) (car liste))))
      liste)))
    (cond ((null liste) nil)
    (liste-freres (cons liste-freres (partitionne-liste (my-remove liste-freres liste) :key key)))
    (t (cons (list (car liste)) (partitionne-liste (cdr liste) :key key))))))

;; --------------------------------------------------------------------------------------------------------------------------

(defun construis (-code)
  "Construire la struture en UMI représentant un dialogue"
  (if (or (null -code) (equal 'fin (car -code))) nil
     (if (numberp (car -code))
        (cons (car -code) (construis (cdr -code)))
        (cons (construis (sequence-incidente (cdr -code) 1))
              (construis (sequence-du-dialogue (cdr -code) 1))))))

(defun sequence-incidente (-code -rang)
  (if (or (zerop -rang) (null -code)) nil
     (case (car -code)
       (deb (cons (car -code) (sequence-incidente (cdr -code) (1+ -rang))))
       (fin (cons (car -code) (sequence-incidente (cdr -code) (1- -rang))))
       (t (cons (car -code) (sequence-incidente (cdr -code) -rang))))))

(defun reste-du-dialogue (-code -rang)
  (if (or (zerop -rang) (null -code)) -code
     (case (car -code)
```

```
        (deb (reste-du-dialogue (cdr -code) (1+ -rang)))
        (fin (reste-du-dialogue (cdr -code) (1- -rang)))
        (t (reste-du-dialogue (cdr -code) -rang)))))

;; ------------------------------------------------------------------------------------------------------------------------

(set-macro-character
  ;; Accéder aux attributs propres d'une instance : $attribut devient (attribut oself)
  #\$
  #'(lambda (stream char)
      (declare (ignore char))
      `(,(intern (string (read stream t nil t)) :user) oself)))

;; ------------------------------------------------------------------------------------------------------------------------

(defun str2cons (chaine)
  "Convertir une chaine de caractères en liste plate dont les éléments sont les mots de la chaine"
  (let ((mot nil) (liste nil))
    (cond ((zerop (length chaine)) nil)
          (t (loop for i from 0 to (1- (length chaine)) do
                (cond ((not (equal #\space (char chaine i))) (push (char chaine i) mot))
                      (t (push (coerce (reverse mot) 'string) liste) (setf mot nil)))
                finally (return (push (coerce (reverse mot) 'string) liste)))
             (reverse liste)))))

(defun cons2str (liste-mots)
  "Concatener une liste de mots dans une chaine de caractères"
  (cond ((null liste-mots) nil)
  (t (concatenate 'string (car liste-mots) " " (cons2str (cdr liste-mots))))))

(defun constostr (liste-symbol)
  "Convertir une liste de symboles en chaîne de caractères"
  (cond ((null liste-symbol) "")
  (t (concatenate 'string (format nil "~A" (car liste-symbol)) " " (constostr (cdr liste-symbol))))))

(defun supprime-blanc-intern (chaine)
   "Supprimer les caractères espace en trop dans une chaine"
   (cond ((<= (length chaine) 1) chaine)
         ((and (equal #\space (char chaine 0)) (equal #\space (char chaine 1)))
          (supprime-blanc-intern (subseq chaine 1)))
         (t (concatenate 'string (string (char chaine 0)) (supprime-blanc-intern (subseq chaine 1))))))

(defun mise-en-forme (chaine)
  "Mettre en forme d'une chaine de caratères pour l'analyser"
  (supprime-blanc-intern (enleve-accent (enleve-ponctuation (string-downcase chaine)))))

(defun enleve-accent (chaine)
  "Remplacer les caractères accentués par des caractères non-accentués"
  (substitute #\e #\é (substitute #\e #\è (substitute #\e #\ê (substitute #\e #\ë
  (substitute #\i #\î (substitute #\i #\ï (substitute #\i #\ì (substitute #\i #\í
  (substitute #\a #\â (substitute #\a #\ä (substitute #\a #\à (substitute #\a #\á
  (substitute #\a #\á (substitute #\a #\å (substitute #\o #\ö (substitute #\o #\ô
  (substitute #\o #\ó (substitute #\o #\ò (substitute #\o #\õ (substitute #\u #\ù
  (substitute #\u #\û (substitute #\u #\ü (substitute #\u #\ú (substitute #\n #\ñ
  (substitute #\c #\ç
  chaine))))))))))))))))))))))))))

(defun enleve-ponctuation (chaine)
  "Enlever les caractères de ponctuation d'une chaîne"
  (substitute #\Space #\. (substitute #\Space #\: (substitute #\Space #\! (substitute #\Space #\'
  (substitute #\Space #\, (substitute #\Space #\; (substitute #\Space #\? (substitute #\Space #\'
  (substitute #\Space #\" (substitute #\Space #\-
  chaine))))))))))

;; ------------------------------------------------------------------------------------------------------------------------
;; Interaction par fichier

#+server
(defvar *timeout* 120)

#+server
(defun lock ()
  (shell (format nil "touch ~A" *fichier-lock*)))

#+server
(defun unlock ()
  (delete-file *fichier-lock*))

#+server
(defun signalFin ()
  (shell (format nil "touch ~A" *fichier-fin*)))

#+server
(defun saisir (prompt &optional (stream t))
  (declare (ignore stream))
  (let ((compteur 0) (entree nil))
    (loop while (and (not (probe-file *fichier-input-flag*))
        (< compteur *timeout*)) do
    (setf compteur (1+ compteur))
    (shell "sleep 1"))
```

```
    (cond ((< compteur *timeout*)
     (delete-file *fichier-input-flag*)
     (with-open-file (stream *fichier-input* :direction :input)
         (setf entree (read-line stream)))
     (delete-file *fichier-input*)
     (mise-en-forme entree))
    (t (unlock) (quit)))))

#+server
(defun affiche (chaine &optional (stream t))
  (declare (ignore stream))
  (affiche-sans-fin chaine)
  (shell (format nil "touch ~A" *fichier-output-flag*)))

#+server
(defun affiche-sans-fin (chaine &optional (stream t))
  (with-open-file (stream *fichier-output*
        :direction :output
        :if-does-not-exist :create
        :if-exists :append)
      (write-string (format nil "~A " chaine) stream)))

;; ------------------------------------------------------------------------------------------------------------------------
;; Interaction à partir du clavier

#-server
(defun saisir (prompt &optional (stream t))
  "Afficher un prompt et saisie d'une chaîne de caractères"
  (format stream prompt)
  (let ((string (string-trim '(#\space #\tab #\newline) (read-line stream nil t))))
    (cond ((zerop (length string)) (saisir prompt stream))
    ((equal "e" string) (e) (saisir prompt stream))
          ((equal "l" string) (l) (saisir prompt stream))
    ((equal "a" string) (a) (saisir prompt stream))
          ((equal "ai" string) (ai) (saisir prompt stream))
    ((equal #\( (char string 0)) (format t "~%~S~%" (eval (read-from-string string))) (saisir prompt stream))
    (t (mise-en-forme string)))))

#-server
(defun affiche (chaine &optional (stream t)) (format stream "~%~A" chaine))

#-server
(defun affiche-sans-fin (chaine &optional (stream t)) (format stream "~A" chaine))

;; ------------------------------------------------------------------------------------------------------------------------

(defun save ()
  "Sauvegarde de l'image mémoire"
  (compile-ksources-dialogueur)
  (saveinitmem "ami.mem" :quiet t :init-function #'(lambda () (lancer))))

;; ------------------------------------------------------------------------------------------------------------------------
```

Listing: utiles/date.lsp

```
; ##############################################################################################################################
; ### AMI - DATE.LSP - LEMEUNIER (version du 10/12/1996) Thierry.Lemeunier@lium.univ-lemans.fr ########################
; ##############################################################################################################################

; ------------------------------------------------------------------------------------------------------------------------------
; Pour obtenir des formes concises de la date et de l'heure

(defun jour ()
  "Donner le jur en chiffre (lundi 0, mardi 1, ...)"
  (multiple-value-bind (-seconde -minute -heure -date -mois -annee -jour) (get-decoded-time)
    (declare (ignore -seconde -minute -heure -date -mois -annee))
    -jour))

(defun lheure ()
  "Donner uniquement l'heure (sur 24 heures)"
  (multiple-value-bind (-seconde -minute -heure -date -mois -annee -jour) (get-decoded-time)
    (declare (ignore -seconde -minute -date -mois -annee -jour))
    -heure))

(defun date ()
  "Donner la date sous la forme j/m/aaaa"
  (multiple-value-bind (-seconde -minute -heure -date -mois -annee -jour) (get-decoded-time)
    (declare (ignore -seconde -minute -heure -jour))
    (format nil "~A/~A/~A" -date -mois -annee)))

(defun heure ()
  "Donner l'heure sous la forme HHhMM"
  (multiple-value-bind (-seconde -minute -heure -date -mois -annee -jour) (get-decoded-time)
    (declare (ignore -seconde -date -mois -annee -jour))
    (format nil "~Dh~D" -heure -minute)))

; ------------------------------------------------------------------------------------------------------------------------------
; Pour obtenir la date avec le jour et le mois en clair (on peut ajouter un offset en nombre de jours)

(defun aujourdhui (&optional (-decalage 0))
  (multiple-value-bind (-seconde -minute -heure -date -mois -annee -jour)
        (decode-universal-time (+ (* 86400 -decalage) (get-universal-time)))
    (declare (ignore -seconde -minute -heure -annee))
    (format nil "~A ~A ~A"
      (nth -jour '("lundi" "mardi" "mercredi" "jeudi" "vendredi" "samedi" "dimanche"))
      -date
      (nth (1- -mois) '("janvier" "février" "mars" "avril" "mai" "juin"
            "juillet" "août" "septembre" "octobre" "novembre" "décembre")))))

; ------------------------------------------------------------------------------------------------------------------------------
```

Listing: utiles/unites.lsp

```
;; ###############################################################################################################################
;; ### AMI - UNITES.LSP - Lemeunier 25/02/99 (Thierry.Lemeunier@lium.univ-lemans.fr)
;; ###############################################################################################################################

(defclass UNITE () ((nom :accessor nom :initarg :nom)))

(defmethod print-object ((oself UNITE) stream) (format stream "<~A>" (nom oself)))

(let ((table-des-unites (make-hash-table)))

  (defun cree-unite (nom objet) (setf (gethash nom table-des-unites) objet))

  (defun unite (nom) (gethash nom table-des-unites))

  (defun lister-unites (&optional (domaine t))
    (let ((resultat nil))
      (maphash #'(lambda (key val)
       (declare (ignore key))
       (if (typep val domaine) (push val resultat)))
         table-des-unites)
      resultat))

  (defun reset-unites (&rest domaines)
    (if domaines (maphash #'(lambda (key val)
           (if (loop for domaine in domaines thereis (typep val domaine))
         (remhash key table-des-unites)))
       table-des-unites)
      (setf table-des-unites (make-hash-table))))
)

(defmethod initialize-instance :after ((oself UNITE) &rest initargs)
  (declare (ignore initargs))
  (unless (slot-boundp oself 'nom) (setf (nom oself) (gentemp (format nil "~S" (type-of oself)))))
  (cree-unite (nom oself) oself))

(set-macro-character #\!
  #'(lambda (stream char)
      (declare (ignore char))
      `(unite ',(read stream t nil t))))

; ------------------------------------------------------------------------------------------------------------------------------
```

Listing: utiles/noms-fichiers.lsp

```
;; ###############################################################################################################################
;; ### AMI - NOMS-FICHIERS.LSP - Lemeunier 01/08/00 (Thierry.Lemeunier@lium.univ-lemans.fr) ########################################
;; ###############################################################################################################################

;; J'ai concentré ici les noms de fichiers pour facilité la maintenance

;; -------------------------------------------------------------------------------------------------------------------------------
;; Noms des fichiers pour la communication entre la CGI et AMI

#+server
(defvar *fichier-lock* "/lium/buster1/lemeunier/public_html/serveur-ami/lock")
#+server
(defvar *fichier-fin* "/lium/buster1/lemeunier/public_html/serveur-ami/fin")
#+server
(defvar *fichier-input-flag* "/lium/buster1/lemeunier/public_html/serveur-ami/input_flag")
#+server
(defvar *fichier-input* "/lium/buster1/lemeunier/public_html/serveur-ami/input")
#+server
(defvar *fichier-output-flag* "/lium/buster1/lemeunier/public_html/serveur-ami/output_flag")
#+server
(defvar *fichier-output* "/lium/buster1/lemeunier/public_html/serveur-ami/output")

;; -------------------------------------------------------------------------------------------------------------------------------
;; Noms des fichiers pour le gestionnaire de messagerie

#+(and clisp (not pc386) (not server))
(defvar *fichier-message* "/lium/buster1/lemeunier/recherche/commonlisp/ami/activite-applicative/message.txt")
#+(and clisp pc386 (not server))
(defvar *fichier-message* "/home/thierry/recherche/commonlisp/ami/activite-applicative/message.txt")

#+server
(defvar *fichier-message* "/lium/buster1/lemeunier/public_html/serveur-ami/message.txt")

;; -------------------------------------------------------------------------------------------------------------------------------
;; Noms des fichiers pour la communication entre l'analyseur CLIPS et CLisp

#+(and clisp (not pc386) (not server))
(defvar *fichier-phrase* "/lium/buster1/lemeunier/recherche/commonlisp/ami/activite-langagiere/phrase.clp")
#+(and clisp (not pc386) (not server))
(defvar *fichier-analyse* "/lium/buster1/lemeunier/recherche/commonlisp/ami/activite-langagiere/analyseur.clp")
#+(and clisp (not pc386) (not server))
(defvar *fichier-sortie* "/lium/buster1/lemeunier/recherche/commonlisp/ami/activite-langagiere/sortie-analyse.clp")
#+(and clisp (not pc386) (not server))
(defvar *fichier-liste-facts* "/lium/buster1/lemeunier/recherche/commonlisp/ami/activite-langagiere/liste-facts.clp")

#+(and clisp pc386 (not server))
(defvar *fichier-phrase* "/home/thierry/recherche/commonlisp/ami/activite-langagiere/phrase.clp")
#+(and clisp pc386 (not server))
(defvar *fichier-analyse* "/home/thierry/recherche/commonlisp/ami/activite-langagiere/analyseur.clp")
#+(and clisp pc386 (not server))
(defvar *fichier-sortie* "/home/thierry/recherche/commonlisp/ami/activite-langagiere/sortie-analyse.clp")
#+(and clisp pc386 (not server))
(defvar *fichier-liste-facts* "/home/thierry/recherche/commonlisp/ami/activite-langagiere/liste-facts.clp")

#+server
(defvar *fichier-phrase* "/lium/buster1/lemeunier/public_html/serveur-ami/phrase.clp")
#+server
(defvar *fichier-analyse* "/lium/buster1/lemeunier/public_html/serveur-ami/analyseur.clp")
#+server
(defvar *fichier-sortie* "/lium/buster1/lemeunier/public_html/serveur-ami/sortie-analyse.clp")
#+server
(defvar *fichier-liste-facts* "/lium/buster1/lemeunier/public_html/serveur-ami/liste-facts.clp")
```

Listing: utiles/xterm.lsp

```
;;; XTERM --- connect to an xterm X window

;;; Bruce Krulwich <krulwich@ils.nwu.edu> 30.12.1991
;;; Bruno Haible <haible@ma2s2.mathematik.uni-karlsruhe.de> 4.1.1994

;; (XTERM:OPEN name &key display geometry title) creates an XTERM window.
;; arguments:
;;      NAME -- a symbol naming the window
;; keywords:
;;      DISPLAY -- X display for the window
;;      GEOMETRY -- X geometry for the window
;;      TITLE -- X title displayed when the window is created
;;               (should be able to have titlebars somehow)
;; Return value:
;;      a bidirectional stream to the XTERM window.

;; (XTERM:FIND name)
;; arguments:
;;      NAME -- a symbol naming the window
;; Return value:
;;      the bidirectional stream to the XTERM window that has this name,
;;      or NIL.

;; (XTERM:NAME window)
;; arguments:
;;      WINDOW -- stream to an XTERM window.
;; Return value:
;;      its name or NIL

;; (XTERM:KILL window) kills an XTERM window
;; argument:
;;      WINDOW -- stream to an XTERM window.

;; (XTERM:LIST-ALL-WINDOWS)
;; Return value:
;;      list of current XTERM windows

;; (XTERM:BROADCAST format-string &rest args)
;; broadcasts a message to every XTERM window.

(defpackage "XTERM"
  (:export "OPEN" "FIND" "NAME" "KILL" "LIST-ALL-WINDOWS" "BROADCAST")
  (:shadow "OPEN" "FIND")
)

(in-package "XTERM")

; A list of triples (name window-stream pid) of all XTERM windows.
(defvar *xterm-list* '())

(defun open (name &key (display nil) (geometry nil) (title nil) (font nil)
                       (xterm-args nil)
            )
  (when (lisp:find name *xterm-list* :key #'first :test #'equal)
    (error "~S: There is already an XTERM window named ~S." 'open name)
  )
  ; generate a unique filename
  (let ((filename (gen-tmp-filename)))
    ; start xterm
    (multiple-value-bind (bi-stream in-stream out-stream)
        (run-program
          "xterm"
          :arguments
            `(,@(if display `("-display" ,(string-downcase display)))
              ,@(if geometry `("-geometry" ,(string-downcase display)))
              ,@(if title `("-title" ,(string title)) `("-title" ,(string name)))
              ,@(if font `("-font" ,(string-downcase font)))
              "-si"
              "-sl" "200"
        ,@xterm-args
             "-e" "utiles/xterm_idle" ,filename
            )
          ; We don't communicate directly with the xterm process but we
          ; specify :stream to let it run concurrently.
          :input :stream :output :stream
        )
      (let ((pid
             (parse-integer
               (remove-if-not #'digit-char-p
                 (write-to-string in-stream :base 10 :radix nil :readably nil)
      )) ) )
        (close in-stream)
        (close out-stream)
        (close bi-stream)
        ; Now wait until filename exists and contains an entire line:
        (let ((ttyname
               (loop
                 (when (probe-file filename)
                   (with-open-file (stream filename :direction :input)
```

```
                    (multiple-value-bind (first-line eof-p)
                        (read-line stream nil nil)
                      (when (and first-line (not eof-p))
                        (return first-line)
            )) ) ) ) ) )
          ; We can now delete the file.
          (delete-file filename)
          ; Create a bidirectional stream to the XTERM window via the tty:
          (let ((stream (lisp:open ttyname :direction :io)))
            (push (list name stream pid) *xterm-list*)
            stream ; That's it!
) ) ) ) ) )

(defun gen-tmp-filename (&optional (prefix "/tmp/clisp_") (length 8))
  (loop
    (let ((filename
           (let ((l '()))
             (dotimes (i length) (push (+ (char-code #\a) (random 26)) l))
             (concatenate 'string prefix (map 'string #'code-char l))
         )) )
      (unless (probe-file filename) (return filename))
) ) )

(defun xterm:find (name)
  (second (lisp:find name *xterm-list* :key #'first :test #'equal))
)

(defun xterm:name (stream)
  (first (lisp:find stream *xterm-list* :key #'second))
)

(defun xterm:kill (stream)
  (let ((info (lisp:find stream *xterm-list* :key #'second)))
    (when info
      (run-shell-command (format nil "kill -9 ~D" (third info)))
      (setq *xterm-list* (delete stream *xterm-list* :key #'second))
) ) )

(defun xterm:list-all-windows ()
  (mapcar #'second *xterm-list*)
)

(defun xterm:broadcast (format-string &rest args)
  (apply #'format (apply #'make-broadcast-stream (xterm:list-all-windows))
                 format-string args
) )
```

Listing: utiles/xterm_idle

```
#!/bin/sh
# Bruce Krulwich <krulwich@ils.nwu.edu> 30.12.1991
# Bruno Haible <haible@ma2s2.mathematik.uni-karlsruhe.de> 4.1.1994
#
# Usage:
#   xterm_idle filename

# This is meant to be executed from an xterm. The xterm process allocates
# the window and the pseudo tty. This script then passes the name of the
# tty to the caller, via filename.

tty > $1

# While this script places the xterm idle, anyone can write to the xterm
# window or read from it - wonderful!

while true
  do
    sleep 1000000
  done
```

Listing: utiles/arbre.lsp

```
;; ##########################################################################################################################
;; ### AMI2 - ARBRE.LSP - Lemeunier 25/01/99 (Thierry.Lemeunier@lium.univ-lemans.fr) ########################################
;; ##########################################################################################################################

;; --------------------------------------------------------------------------------------------------------------------------
;; Définition d'une hiérarchie (les arbres n-aires sans étiquette)

(defclass ARBRE ()
  ((pere :accessor pere :initarg :pere :initform nil)
   (fils :accessor fils :initarg :fils :initform nil)
   (pere-stat :accessor pere-stat :initarg :pere-stat :initform '(ou))
   (fils-stat :accessor fils-stat :initarg :fils-stat :initform '(ou)))
  (:documentation "Une représentation des arbre n-aires"))

;; --------------------------------------------------------------------------------------------------------------------------
;; Méthode de copie profonde

(defmethod copie-noeud ((noeud ARBRE))
  "Créer un noeud à partir d'un autre noeud"
  (cree-noeud :pere-stat (pere-stat noeud)
       :fils-stat (fils-stat noeud)))

(defmethod copie-profonde ((arbre ARBRE) &optional (pere nil))
   "Copier de façon profonde un arbre naires"
   (let ((racine (copie-noeud arbre)))
      (setf (pere racine) pere)
      (setf (fils racine)
            (loop for fils in (fils arbre)
              collect (copie-profonde fils racine)))
      racine))

;; --------------------------------------------------------------------------------------------------------------------------
;; Méthodes de consultations

(defmethod hierarchie-et ((oself ARBRE)) (equal 'et (car $fils-stat)))

(defmethod hierarchie-ou ((oself ARBRE)) (equal 'ou (car $fils-stat)))

(defmethod completp ((oself ARBRE))
  (case (car $fils-stat)
    (ou (>= (length (cdr $fils-stat)) (length $fils)))
    (et (= (length (cdr $fils-stat)) (length $fils)))))

(defmethod appartient-meme-arbre-p ((arbre1 ARBRE) (arbre2 ARBRE))
  "Tester si arbre1 et arbre2 sont du même arbre (cad s'ils ont la même racine)"
  (equal (racine arbre1) (racine arbre2)))

(defmethod frere ((arbre1 ARBRE) (arbre2 ARBRE))
  "Tester si arbre1 et arbre2 sont freres directs"
  (equal (pere arbre1) (pere arbre2)))

(defmethod descendantp ((oself ARBRE) (arbre ARBRE))
  "Tester si oself est un sous-arbre de arbre"
  (or (equal arbre $pere)
      (and $pere (pere $pere) (descendantp $pere arbre))))

(defmethod terminale ((oself ARBRE)) (null (cdr $fils-stat)))

(defmethod nombre-max-fils ((oself ARBRE))
  "Retourner le nombre maximum de fils qu'un noeud peut avoir dans l'arbre"
  (if $fils (funcall #'max (length $fils) (car (mapcar #'nombre-max-fils $fils))) 0))

(defun liste-en-hierarchie (noeuds-classes)
  "Retourner la liste des listes des hiérarchies"
  (let* ((nouvelle-liste (cdr noeuds-classes))
   (resul (cons (car noeuds-classes)
          (loop for noeud in (cdr noeuds-classes)
        when (descendantp noeud (car noeuds-classes))
        collect (progn (setf nouvelle-liste (remove noeud nouvelle-liste)) noeud)))))
    (if nouvelle-liste
  (cons resul (liste-en-hierarchie nouvelle-liste))
      (list resul))))

;; --------------------------------------------------------------------------------------------------------------------------
;; Méthodes de recherches

(defmethod racine ((oself ARBRE))
  "Rechercher la racine d'un noeud"
  (if $pere (racine $pere) oself))

(defmethod premier-fils ((oself ARBRE) &key predicat)
  "Retourner le premier fils vérifiant un prédicat"
  (loop for fils in $fils
  when (if predicat (funcall predicat fils) fils)
  do (return fils)
  finally (return nil)))

(defmethod second-fils ((oself ARBRE) &key predicat)
  "Retourner le second fils vérifiant un prédicat"
```

```
    (loop for fils in $fils
    when (and (if predicat (funcall predicat fils) fils)
        (not (equal fils $premier-fils)))
    do (return fils)
    finally (return nil)))

(defmethod fils-meme-predicat ((oself ARBRE) &key predicat)
  "Retourner la liste des fils vérifiant un prédicat"
  (loop for fils in $fils
    when (funcall predicat fils)
    collect fils into resultat
    finally (return resultat)))

;; ------------------------------------------------------------------------------------------------------------------------
;; Méthodes de modifications

(defmethod ajoute-fils ((oself ARBRE) &rest liste-fils)
   "Ajouter des fils à la liste des fils de l'arbre receveur"
   (loop for un-fils in (if (consp (car liste-fils)) (car liste-fils) liste-fils) do
     (pushnew un-fils $fils)
     (setf (pere un-fils) oself))
   (setf $fils (my-sort $fils #'> :key #'interet))
   oself)

(defmethod enleve-fils ((oself ARBRE) &rest liste-fils)
   "Enlever des fils de la liste des fils de l'arbre receveur"
   (loop for un-fils in (if (consp (car liste-fils)) (car liste-fils) liste-fils)
     when (member un-fils $fils) do
     (setf $fils (remove un-fils $fils))
     (setf (pere un-fils) nil))
   oself)

(defmethod applique-statut ((oself ARBRE) val-statut)
  "Appliquer à l'arbre entier la valeur du statut"
  (setf $statut val-statut)
  (mapcar #'(lambda (x) (applique-statut x val-statut)) $fils)
  oself)

(defmethod applique-statut-fils ((oself ARBRE) val-statut)
  "Appliquer aux fils de l'arbre la valeur du statut"
  (loop for fils in $fils do (applique-statut fils val-statut)))

;; ------------------------------------------------------------------------------------------------------------------------
```

Listing: utiles/blackboard/black.lsp

```
;; ##########################################################################################################################
;; ### LA CLASSE DES TABLEAUX NOIRS 2.1 - LEHUEN 27/10/98 (lehuen@lium.univ-lemans.fr)
;; ##########################################################################################################################

(defgeneric meme (objet1 objet2) (:method (objet1 objet2) (equal objet1 objet2)))

(defun vrai (&rest x) (declare (ignore x)) t)

(defconstant ERRQUANT "Le quantificateur ~S n'est pas défini")
(defconstant ERRNIV "Le niveau ~S n'est pas défini")

;; --------------------------------------------------------------------------------------------------------------------------
;; Déclaration des classes de base
;; --------------------------------------------------------------------------------------------------------------------------

(defclass PRECONDITION (unite)
  ((niveau :reader niveau :initarg :niveau)
   (pile :accessor pile :initform (list t))
   (predicat :accessor predicat :initarg :predicat :initform #'vrai)))

(defclass EPRECOND (PRECONDITION)
  ((variable :reader variable :initarg :variable)))

(defclass UPRECOND (PRECONDITION)
  ((quantificateur :reader quantificateur :initarg :quantificateur)
   (variable :reader variable :initarg :variable :initform nil)))

(defclass KSOURCE (unite)
  ((nom :reader nom :initarg :nom)
   (tableau :reader tableau :initarg :tableau)
   (doc :reader doc :initarg :doc)
   (preconditions :accessor preconditions :initarg :preconditions :initform nil)
   (condition :accessor condition :initarg :condition :initform #'vrai)
   (action :accessor action :initarg :action :initform #'vrai)
   (interet :accessor interet :initarg :interet :initform 0)))

(defclass KSAR (unite)
  ((ksource :accessor ksource :initarg :ksource)
   (elements :accessor elements :initarg :elements)
   (interet :accessor interet :initarg :interet)))

(defclass ELEMENT ()
  ((nom :accessor nom :initarg :nom)
   (tableau :reader tableau :initform nil)
   (niveau :allocation :class :reader niveau :initform nil)
   (statut :accessor statut :initarg :statut :initform 'disponible)
   (interet :accessor interet :initarg :interet :initform 1)
   (modif-flag :accessor modif-flag :initform nil)))

(defclass TABLEAU (UNITE)
  ((ksources :accessor ksources :initform nil)
   (historique :accessor historique :initform nil)
   (evenements :accessor -evenements)
   (agenda :accessor -agenda)
   (mode-pas-a-pas :accessor mode-pas-a-pas :initarg :mode-pas-a-pas :initform nil)
   (trace-avec-agenda :accessor trace-avec-agenda :initarg :trace-avec-agenda :initform nil)
   (trace-stream :accessor trace-stream :initform nil)))

;; --------------------------------------------------------------------------------------------------------------------------
;; Macros d'instanciation
;; --------------------------------------------------------------------------------------------------------------------------

(defmacro cree-eprecond (&rest initargs) `(make-instance 'EPRECOND ,@initargs))
(defmacro cree-uprecond (&rest initargs) `(make-instance 'UPRECOND ,@initargs))
(defmacro cree-ksource (&rest initargs) `(make-instance 'KSOURCE ,@initargs))
(defmacro cree-ksar (&rest initargs) `(make-instance 'KSAR ,@initargs))

;; --------------------------------------------------------------------------------------------------------------------------
;; Méthodes des préconditions génériques PRECONDITION
;; --------------------------------------------------------------------------------------------------------------------------

(defmethod elements ((oself PRECONDITION)) (cdr $pile))

(defmethod reinitialiser ((oself PRECONDITION)) (setf $pile (list t)))

(defmethod enleve ((oself PRECONDITION) (element ELEMENT)) (delete element $pile))

(defmethod ajoute ((oself PRECONDITION) (element ELEMENT))
  (if $elements
      (setf (cdr $pile) (cons element (cdr $pile)))
      (setf (cdr $pile) (list element))))

(defmethod compiler ((oself PRECONDITION)) (setf $predicat (or (compile nil $predicat) $predicat)))

;; --------------------------------------------------------------------------------------------------------------------------
;; Méthodes des préconditions existancielles EPRECOND
;; --------------------------------------------------------------------------------------------------------------------------

(defmethod universelle ((oself EPRECOND)) nil)
```

```
90
(defmethod print-object ((oself EPRECOND) stream) (format stream "~A:~A" $variable $elements))

(defmethod satisfait ((oself EPRECOND)) $elements)

95 #|
(defmethod accept ((oself EPRECOND) (element ELEMENT))
  "Regarde si la précondition est validée ou invalidée par un élément"
  (if (member element $elements)
      (unless (funcall $predicat element) (enleve oself element))
100   (when (funcall $predicat element) (ajoute oself element))))
|#

(defmethod accept ((oself EPRECOND) (element ELEMENT))
  "Regarde si la précondition est validée ou invalidée par un élément"
105 (if (member element $elements)
      (if (funcall $predicat element) (modif-flag element) (enleve oself element))
      (if (funcall $predicat element) (ajoute oself element))))

;; ------------------------------------------------------------------------------------------------------------------------
110 ;; Méthodes des préconditions universelles UPRECOND
;; ------------------------------------------------------------------------------------------------------------------------

(defmethod universelle ((oself UPRECOND)) t)

115 (defmethod print-object ((oself UPRECOND) stream) (format stream "~A:~A" $quantificateur $elements))

(defmethod satisfait ((oself UPRECOND))
  (case $quantificateur
  (collect t)
120 (existe $elements)
  (aucun (null $elements))
  (tous (null $elements))
  (un (= 1 (length $elements)))
  (t (error ERRQUANT $quantificateur))))
125
(defmethod accept ((oself UPRECOND) (element ELEMENT))
  "Regarde si la précondition est validée ou invalidée un élément"
  (if (member element $elements)
      (case $quantificateur
130   (collect (unless (funcall $predicat element) (enleve oself element) t))
      (existe (unless (funcall $predicat element) (enleve oself element) $elements))
           (aucun (unless (funcall $predicat element) (enleve oself element) (null $elements)))
      (tous (when (funcall $predicat element) (enleve oself element) (null $elements)))
      (un (unless (funcall $predicat element) (enleve oself element) (= 1 (length $elements))))
135   (t (error ERRQUANT $quantificateur)))
      (case $quantificateur
      (collect (when (funcall $predicat element) (ajoute oself element) t))
      (existe (when (funcall $predicat element) (ajoute oself element) $elements))
      (aucun (when (funcall $predicat element) (ajoute oself element) nil))
140   (tous (unless (funcall $predicat element) (ajoute oself element) nil))
      (un (when (funcall $predicat element) (ajoute oself element) (= 1 (length $elements))))
      (t (error ERRQUANT $quantificateur)))))

;; ------------------------------------------------------------------------------------------------------------------------
145 ;; Méthodes des sources de connaissance KSOURCE (knowledge sources)
;; ------------------------------------------------------------------------------------------------------------------------

(defmethod print-object ((oself KSOURCE) stream) (format stream "<~A [~A] [~A]>" $nom $interet $doc))

150 (defmethod meme ((ksource1 KSOURCE) (ksource2 KSOURCE)) (equal (nom ksource1) (nom ksource2)))

(defmethod executable ((oself KSOURCE)) (loop for precondition in $preconditions always (satisfait precondition)))

(defmethod reinitialiser ((oself KSOURCE)) (loop for precondition in $preconditions do (reinitialiser precondition)))
155
(defmethod initialize-instance :after ((oself KSOURCE) &rest initargs)
  "Pour que chaque tableau et chaque niveau connaisse les ksources qui le concernent"
  (declare (ignore initargs))
  (unless $interet (setf $interet (length $preconditions)))
160 (when *niveau-debug* (format t "~%;; Lecture de la ksource ~A [~A]" $nom $interet))
  (push oself (ksources $tableau))
  (loop for precondition in $preconditions do
  (if (slot-exists-p $tableau (niveau precondition))
      (pushnew oself (slot-value $tableau (niveau precondition)) :test 'meme)
165   (error ERRNIV (niveau precondition)))))

(defmethod (setf interet) :after (valeur (oself KSOURCE))
  (declare (ignore valeur))
  (tracer $tableau (format nil "Modification de l'intérêt de ~A -> ~A" oself $interet))
170 (if (cherche-ksar $tableau oself)
      (initialize-instance (cherche-ksar $tableau oself))))

;; La méthode de la classe KSOURCE qui demande le plus de temps de calcul
(defmethod construis-ksars ((oself KSOURCE))
175 "Pour construire les ksars en combinant les préconditions avec les éléments - Version récursive"
  (labels ((construis (preconditions elements)
         (cond ((null preconditions)
          (when (apply $condition elements) (cree-ksar :ksource oself :elements elements)))
         ((universelle (car preconditions))
```

```
          (if (variable (car preconditions))
        (construis (cdr preconditions) (cons (elements (car preconditions)) elements))
           (construis (cdr preconditions) elements)))
         (t (loop for element in (elements (car preconditions)) do
        (construis (cdr preconditions) (cons element elements)))))))
    (construis (reverse $preconditions) nil)))

(defmethod compiler ((oself KSOURCE))
  (when *niveau-debug* (format t "~%;; Compilation de la ksource ~A" $nom))
  (setf $condition (or (compile nil $condition) $condition))
  (setf $action (or (compile nil $action) $action))
  (loop for precondition in $preconditions do (compiler precondition)))

;; ------------------------------------------------------------------------------------------------------------------
;; Méthodes des sources de connaissance activées KSAR (knowledge source activation records)
;; ------------------------------------------------------------------------------------------------------------------

(defmethod print-object ((oself KSAR) stream)
  (format stream "~A sur ~A [~A]"
    (nom $ksource)
    (mapcar #'(lambda (element) (if (listp element) '+ element)) $elements)
        ;; $elements
    $interet))

;;; (defmethod print-object ((oself KSAR) stream) (format stream "~A sur ~A [~A]" (nom $ksource) $elements $interet))

(defmethod initialize-instance :after ((oself KSAR) &rest initargs)
  "Calcul de l'intérêt et insertion dans l'agenda"
  (declare (ignore initargs))
  (setf $interet (case (length $elements)
           (0 (interet $ksource))
           (1 (+ (interet $ksource) (interet (car $elements))))
           (t (+ (interet $ksource) (apply '+ (mapcar 'interet $elements))))))

  (labels ((inserer-dans (agenda)
      (cond ((null (cdr agenda)) (setf (cdr agenda) (list oself)))
            ((> $interet (interet (cadr agenda))) (setf (cdr agenda) (cons oself (cdr agenda))))
            (t (inserer-dans (cdr agenda))))))

    (inserer-dans (agenda (tableau $ksource)))))

(defmethod interet ((oself NULL)) 0)
(defmethod interet ((oself CONS))
  "Intérêt d'une liste d'éléments obtenue avec les UPRECOND (voir construis-ksars)"
  (apply '+ (mapcar 'interet oself)))

;; ------------------------------------------------------------------------------------------------------------------
;; Méthodes des éléments génériques ELEMENT (données et hypothèses)
;; ------------------------------------------------------------------------------------------------------------------

(defmethod print-object ((oself ELEMENT) stream) (format stream "<~A>" $nom))

(defmethod initialize-instance :after ((oself ELEMENT) &rest initargs)
  (declare (ignore initargs))
  (unless (slot-boundp oself 'nom) (setf $nom (gentsym (format nil "~S" $type-of))))
  (push oself (evenements $tableau))
  (push oself (historique $tableau))
  (tracer $tableau (format nil "Création de ~A" oself)))

(defmethod (setf statut) :after (valeur (oself ELEMENT))
  (declare (ignore valeur))
  (pushnew oself (evenements $tableau) :test #'meme)
  (tracer $tableau (format nil "Modification du statut de ~A -> ~A" oself $statut)))

(defmethod (setf interet) :after (valeur (oself ELEMENT))
  (declare (ignore valeur))
  (pushnew oself (evenements $tableau) :test #'meme)
  (tracer $tableau (format nil "Modification de l'intérêt de ~A -> ~A" oself $interet))
  (setf $modif-flag t))

(defmethod propager ((oself ELEMENT) (ksource KSOURCE))
  (and (equal $tableau (tableau ksource))
       (not (zerop (loop for precondition in (preconditions ksource)
       when (equal $niveau (niveau precondition))
       count (accept precondition oself))))))

(defmethod reactiver ((oself ELEMENT))
  "Ajouter à la main un élément dans la liste des événements"
  (setf $modif-flag t)
  (pushnew oself (evenements $tableau) :test 'meme))

(defmethod disponible ((oself ELEMENT)) (equal $statut 'disponible))

(defmethod utilise ((oself ELEMENT)) (equal $statut 'utilise))

;; ------------------------------------------------------------------------------------------------------------------
;; Méthodes des tableaux noirs génériques TABLEAU
;; ------------------------------------------------------------------------------------------------------------------

(defmethod agenda ((oself TABLEAU)) $-agenda)
```

```
(defmethod evenements ((oself TABLEAU)) $-evenements)

(defmethod (setf agenda) (valeur (oself TABLEAU)) (setf $-agenda valeur))
(defmethod (setf evenements) (valeur (oself TABLEAU)) (setf $-evenements valeur))

;; ------------------------------------------------------------------------------------------------------------------------

(defmethod initialize-instance :after ((oself TABLEAU) &rest initargs)
   (declare (ignore initargs))
   #+(and clisp (not server))
   (setf $trace-stream (xterm:open "Trace du système AMI" :xterm-args '("-iconic" "-sb" "-sl" "50000" "-fn" "fixed")))
   #+server
   (setf $trace-stream (open "trace-ami.txt" :direction :output :if-does-not-exist :create :if-exists :append))

(defmethod init-all-slots ((oself TABLEAU))
  (setf $-evenements nil
  $-agenda (list t)
  $historique nil))

(defmethod reset ((oself TABLEAU))
   "Initialisation du tableau noir et des sources de connaissance"
   (init-all-slots oself)
   (tracer oself "Initialisation du tableau noir")
   (loop for ksource in $ksources do (reinitialiser ksource))
   (initialiser oself)
   (propagation oself)
   (when $trace-avec-agenda (envoyer-agenda-vers-trace oself)))

(defmethod cherche-ksar ((oself TABLEAU) (ks KSOURCE))
  (loop for ksar in (cdr $agenda)
    when (eq ks (ksource ksar))
    return ksar
    finally return nil))

(defmethod execution ((oself TABLEAU))
  "La boucle d'éxécution du tableau noir"
  ;; (propagation oself)
  ;; (when $trace-avec-agenda (envoyer-agenda-vers-trace oself))
  (loop (if (cdr $agenda)
      (let ((ksar (cadr $agenda)))
        (setf (cdr $agenda) (cddr $agenda))
        (tracer oself (format nil "~%Éxécution de ~A" ksar))
            (apply (action ksar) (elements ksar))
            (propagation oself)
        (when $trace-avec-agenda (envoyer-agenda-vers-trace oself))
            (when $mode-pas-a-pas (return)))
    (return))))

(defmethod propagation ((oself TABLEAU))
   "Mise à jour de l'agenda en fonction des évènements"
   (let (ksources-candidates)

     (loop for element in $evenements do
       (loop for ksource in (slot-value oself (niveau element)) do
  (if (propager element ksource)
            (pushnew ksource ksources-candidates :test #'meme)))
       (setf (modif-flag element) nil))
     (setf $evenements nil)

     (loop for ksar in (cdr $agenda) do
       (if (or (member (ksource ksar) ksources-candidates)
         (not (executable (ksource ksar))))
           (delete ksar $agenda)))

;    (loop for ksar in (cdr $agenda) do
;   (if (member (ksource ksar) ksources-candidates)
;       (delete ksar $agenda)))

     (loop for ksource in ksources-candidates do
       (when (executable ksource) (construis-ksars ksource)))))

(defmethod tracer ((oself TABLEAU) texte) (format $trace-stream "~%~A" texte))

(defmethod envoyer-agenda-vers-trace ((oself TABLEAU))
  "Envoie l'agenda vers la trace"
  (format $trace-stream "~%")
  (if (cdr $agenda)
      (loop for ksar in (cdr $agenda) do
      (tracer oself (format nil " | ~A" ksar)))
    (tracer oself " | Il n'y a plus de ksar dans l'agenda")))

(defmethod afficher-historique ((oself TABLEAU) stream)
  (loop for element in (reverse $historique) do
  (format stream "~%~A" (decrire element))))

(defmethod tous-les (classe (oself TABLEAU))
  (remove-if-not #'(lambda (element) (typep element classe)) $historique))

(defmethod compile-ksources ((oself TABLEAU))
```

```
360     (loop for ksource in $ksources do (compiler ksource)))

;; ----------------------------------------------------------------------------------------------------
```

Listing: utiles/blackboard/black-rec.lsp

```
; ##############################################################################################################
; ### LE TABLEAU NOIR RECURSIF - LEHUEN 31/03/98 (lehuen@lium.univ-lemans.fr)
; ##############################################################################################################

;; Les classes du tableau 2.1 doivent être chargées au préalable

; --------------------------------------------------------------------------------------------------------------
; Déclaration des nouvelles classes pour la récursion (empilement de contextes)
; --------------------------------------------------------------------------------------------------------------

(defclass TABLEAU-R (TABLEAU) ((evenements :accessor -evenements :initform (list nil))))

(defclass PRECONDITION-R (PRECONDITION) ((rec :reader rec :initarg :rec :initform t)))

(defclass EPRECOND-R (PRECONDITION-R EPRECOND) nil)
(defclass UPRECOND-R (PRECONDITION-R UPRECOND) nil)

(defmacro cree-eprecond-r (&rest initargs) `(make-instance 'EPRECOND-R ,@initargs))
(defmacro cree-uprecond-r (&rest initargs) `(make-instance 'UPRECOND-R ,@initargs))

(defmethod print-object ((oself EPRECOND-R) stream) (format stream "~A:~A" $variable $pile))
(defmethod print-object ((oself UPRECOND-R) stream) (format stream "~A:~A" $quantificateur $pile))

; --------------------------------------------------------------------------------------------------------------
; Méthodes des tableaux noirs récursifs génériques TABLEAU-R
; --------------------------------------------------------------------------------------------------------------

(defmethod agenda ((oself TABLEAU-R)) (car $-agenda))
(defmethod evenements ((oself TABLEAU-R)) (car $-evenements))

(defmethod (setf agenda) (valeur (oself TABLEAU-R)) (setf (car $-agenda) valeur))
(defmethod (setf evenements) (valeur (oself TABLEAU-R)) (setf (car $-evenements) valeur))

; --------------------------------------------------------------------------------------------------------------

(defmethod init-all-slots ((oself TABLEAU-R))
  (setf $-evenements (list nil)
  $-agenda (list (list t))
  $historique nil))

(defmethod empiler ((oself TABLEAU-R))
  "Empile un nouveau contexte de résolution"
  (push (list t) $-agenda)
  (push nil $-evenements)
  (loop for ksource in $ksources do
  (loop for precondition in (preconditions ksource) do
       (empiler precondition))))

(defmethod depiler ((oself TABLEAU-R))
  "Retrouve le précédent contexte de résolution"
  (pop $-agenda)
  (pop $-evenements)
  (loop for ksource in $ksources do
  (loop for precondition in (preconditions ksource) do
       (depiler precondition))))

; --------------------------------------------------------------------------------------------------------------
; Méthodes des préconditions récursives génériques PRECONDITION-R
; --------------------------------------------------------------------------------------------------------------

(defmethod elements ((oself PRECONDITION-R)) (cdar $pile))

(defmethod reinitialiser ((oself PRECONDITION-R)) (setf $pile (list (list t))))

(defmethod empiler ((oself PRECONDITION-R)) (when $rec (push (list t) $pile)))

(defmethod depiler ((oself PRECONDITION-R)) (when $rec (pop $pile)))

(defmethod enleve ((oself PRECONDITION-R) (element ELEMENT)) (delete element (car $pile)))

(defmethod ajoute ((oself PRECONDITION-R) (element ELEMENT))
  (if $elements
      (setf (cdar $pile) (cons element (cdar $pile)))
      (setf (cdar $pile) (list element))))

; --------------------------------------------------------------------------------------------------------------
```

Listing: utiles/blackboard/gramm.lsp

```
; ##############################################################################################################################
; ### LE COMPILATEUR ATT ET LA GRAMMAIRE DE DESCRIPTION DES KS - LEHUEN 18/02/98 (lehuen@lium.univ-lemans.fr)
; ##############################################################################################################################

;; Ce fichier contient le compilateur d'ATT (defatt) et la grammaire des sources de connaissances (defregle)

(defmacro defatt (nom description arbre)
  (declare (ignore description))
  `(defun ,nom (listemots)
     ,(compile-att arbre)))

(defun compile-att (arbre)
  "Le compilateur ATT (Nicolle 95 modifié Lehuen 97)"
  (if (null arbre)
      `(values t nil listemots)
    (case (car arbre)
    (%ou (labels ((aux (arbre)
          (if (null arbre)
         `(values nil nil listemots)
            `(multiple-value-bind (reussi resultat restemots)
               ,(compile-att (car arbre))
               (if reussi
              (values reussi resultat restemots)
                 ,(aux (cdr arbre)))))))
           (aux (cdr arbre))))
    (%res `(values t (progn ,@(cdr arbre)) listemots))
    (%res-final `(if listemots
        (values nil nil listemots)
      (progn ,@(cdr arbre))))
    (%app `(multiple-value-bind (reussi ,(nth 2 arbre) listemots)
             (,(nth 1 arbre) listemots)
             (if reussi
           ,(compile-att (cdddr arbre))
         (values nil nil listemots))))
    (%type `(if (typep (car listemots) ',(nth 1 arbre))
          (let ((,(nth 2 arbre) (car listemots)) (listemots (cdr listemots)))
      ,(compile-att (cdddr arbre)))
        (values nil nil listemots)))
    (%parmi `(if (member (car listemots) ',(nth 1 arbre))
          (let ((,(nth 2 arbre) (car listemots)) (listemots (cdr listemots)))
      ,(compile-att (cdddr arbre)))
        (values nil nil listemots)))
    (%null `(if (null (car listemots))
         ,(compile-att (cdr arbre))))
    (%borne `(if (equal (car listemots) ',(cadr arbre))
          ,(compile-att (cddr arbre))
        (values nil nil listemots)))
    (t `(if (equal ',(car arbre) (car listemots))
      (let ((listemots (cdr listemots)))
        ,(compile-att (cdr arbre)))
      (values nil nil listemots))))))

; ----------------------------------------------------------------------------------------------------------------------------
; Définition du langage de description des sources de connaissances (defregle est la macro d'appel)
; ----------------------------------------------------------------------------------------------------------------------------

(defmacro defregle (&rest description) (att-regle description))

(defatt att-regle "La branche principale qui appelle la fonction construis-regle"
  (%type symbol identificateur
   %type symbol tableau
   %app att-interet interet
   DOCUMENTATION
   %app att-documentation documentation
   PRECONDITIONS
   %app att-preconds preconditions
   %app att-condition condition
   ACTIONS
   %app att-actions actions
   %res-final (construis-regle identificateur tableau interet documentation preconditions condition actions)))

(defatt att-interet "La branche pour le champ optionnel interet"
  (%ou (%borne PRECONDITIONS %res 0)
       (%type fixnum interet %res interet)))

(defatt att-documentation "La branche pour la documentation de la ksource"
  (%type string documentation %res documentation))

(defatt att-preconds "La branche récursive qui construit la liste des préconditions"
  (%ou (%borne CONDITION %res nil)
       (%borne ACTIONS %res nil)
       (%app att-precond precondition %app att-preconds liste %res (cons precondition liste))))

(defatt att-precond "L'embranchement vers les quatre types de syntaxes de préconditions"
  (%ou (%app att-precond1 precondition %res precondition)
       (%app att-precond2 precondition %res precondition)
       (%app att-precond3 precondition %res precondition)
       (%app att-precond4 precondition %res precondition)))
```

```
(defatt att-precond1 "La branche précondition universelle avec variable"
  (%type symbol niveau
   %parmi (un tous aucun existe collect) quantificateur
   %type cons predicat
   ->
   %type symbol variable
   %res '(cree-uprecond :variable ',variable
      :quantificateur ',quantificateur
      :niveau ',niveau
      :predicat ,predicat)))

(defatt att-precond2 "La branche précondition universelle sans variable"
  (%type symbol niveau
   %parmi (un tous aucun existe collect) quantificateur
   %type cons predicat
   %res '(cree-uprecond :quantificateur ',quantificateur
      :niveau ',niveau
      :predicat ,predicat)))

(defatt att-precond3 "La branche précondition existancielle avec variable"
  (%type symbol niveau
   %type cons predicat
   ->
   %type symbol variable
   %res '(cree-eprecond :variable ',variable
      :niveau ',niveau
      :predicat ,predicat)))

(defatt att-precond4 "La branche précondition existancielle sans variable"
  (%type symbol niveau
   %type cons predicat
   %res '(cree-uprecond :quantificateur 'existe
      :niveau ',niveau
      :predicat ,predicat)))

(defatt att-condition "La branche pour le champ optionnel condition"
  (%ou (%borne ACTIONS %res nil)
       (CONDITION %type cons condition %res condition)))

(defatt att-actions "La branche récursive qui construit la liste des actions"
  (%ou (%null %res nil)
       (%type cons action %app att-actions liste %res (cons action liste))))

; ------------------------------------------------------------------------------------------------------------------------

(defun parametres (preconditions)
  "Construction de la liste des paramètres"
  (loop for precond in preconditions
  when (eq :variable (cadr precond))
  collect (eval (caddr precond))))

(defun construis-regle (identificateur tableau interet documentation preconditions condition actions)
  "Construction de la source de connaissances"
  (if condition

      '(cree-ksource :nom ',identificateur
                     :tableau ,tableau
        :interet ,interet
                     :doc ,documentation
        :preconditions (list ,@preconditions)
        :condition #'(lambda ,(parametres preconditions) ,condition)
        :action #'(lambda ,(parametres preconditions) ,@actions))

      '(cree-ksource :nom ',identificateur
                     :tableau ,tableau
        :interet ,interet
                     :doc ,documentation
        :preconditions (list ,@preconditions)
        :action #'(lambda ,(parametres preconditions) ,@actions))))

; ------------------------------------------------------------------------------------------------------------------------
```

Listing: utiles/blackboard/gramm-rec.lsp

```
; ##########################################################################################################################
; ### LA GRAMMAIRE DE DESCRIPTION DES KS RECURSIVES - LEHUEN 31/03/98 (lehuen@lium.univ-lemans.fr)
; ##########################################################################################################################

;; Le fichier du compilateur ATT et de la grammaire de base (gramm.lsp) doit être chargé au préalable

(defmacro defregle-r (&rest description) (att-regle-rec description))

(defatt att-regle-rec "La branche principale qui appelle la fonction construis-regle"
  (%type symbol identificateur
   %type symbol tableau
   %app att-interet interet
   DOCUMENTATION
   %app att-documentation documentation
   PRECONDITIONS
   %app att-preconds-rec preconditions
   %app att-condition condition
   ACTIONS
   %app att-actions actions
   %res-final (construis-regle identificateur tableau interet documentation preconditions condition actions)))

(defatt att-preconds-rec "La branche qui construit la liste des préconditions récursives"
  (%ou (%borne CONDITION %res nil)
       (%borne ACTIONS %res nil)
       (%app att-precond-rec precondition %app att-preconds-rec liste %res (cons precondition liste))))

(defatt att-precond-rec "L'embranchement vers les quatre types de syntaxes de préconditions"
  (%ou (%app att-precond1-rec precondition %res precondition)
       (%app att-precond2-rec precondition %res precondition)
       (%app att-precond3-rec precondition %res precondition)
       (%app att-precond4-rec precondition %res precondition)))

(defatt att-precond1-rec "La branche précondition universelle avec variable"
  (%type symbol niveau
   %parmi (un tous aucun existe collect) quantificateur
   %type cons predicat
   ->
   %type symbol variable
   %app att-rec rec
   %res '(cree-uprecond-r :variable ',variable
        :quantificateur ',quantificateur
        :niveau ',niveau
        :predicat ,predicat
        :rec ,rec)))

(defatt att-precond2-rec "La branche précondition universelle sans variable"
  (%type symbol niveau
   %parmi (un tous aucun existe collect) quantificateur
   %type cons predicat
   %app att-rec rec
   %res '(cree-uprecond-r :quantificateur ',quantificateur
        :niveau ',niveau
        :predicat ,predicat
        :rec ,rec)))

(defatt att-precond3-rec "La branche précondition existancielle avec variable"
  (%type symbol niveau
   %type cons predicat
   ->
   %type symbol variable
   %app att-rec rec
   %res '(cree-eprecond-r :variable ',variable
        :niveau ',niveau
        :predicat ,predicat
        :rec ,rec)))

(defatt att-precond4-rec "La branche précondition existancielle sans variable"
  (%type symbol niveau
   %type cons predicat
   %app att-rec rec
   %res '(cree-uprecond-r :quantificateur 'existe
        :niveau ',niveau
        :predicat ,predicat
        :rec ,rec)))

(defatt att-rec "Le paramètre optionnel de récursivité"
  (%ou ([norec] %res nil)
       (%res t)))

; ------------------------------------------------------------------------------------------------------------------------
; Pour examiner la création des sources de connaissance

#|

(defmacro voir-ks (&rest description) (print (att-regle-rec description)))

(voir-ks cree-lexeme *dialogueur* 900

  PRECONDITIONS
```

```
90      niv-groupes #'initial -> groupe
        niv-termes #'vrai -> terme [norec]

      CONDITION
        (search (expression terme) (expression groupe))
95
      ACTIONS
        (cree-lexeme :pos (+ (pos groupe) (search (expression terme) (expression groupe)))
         :cat (cat terme)
         :couverture (length (expression terme))
100      :expression (expression terme)))
    |#

    ; ----------------------------------------------------------------------------------------------------------------
```

Listing: activite-dialogique/dialogueur.lsp

```
;; ###############################################################################################################################
;; ### AMI2 - DIALOGUEUR.LSP - Lemeunier 03/11/98 (Thierry.Lemeunier@lium.univ-lemans.fr) ##########################
;; ###############################################################################################################################

;; -------------------------------------------------------------------------------------------------------------------------------
;; La fenêtre de dialogue

(setf *stream-dialogue* t)

#+(and clisp (not server))
(setf *stream-dialogue* (xterm:open "Fenêtre d'interaction avec AMI" :xterm-args '("-sb" "-fn" "fixed")))

#+server
(setf *stream-dialogue* t)

;; -------------------------------------------------------------------------------------------------------------------------------
;; Les classes du tableau noir récursif TABLEAU-R doivent être chargées au préalable

;; -------------------------------------------------------------------------------------------------------------------------------
;; Classe du dialogueur
;; -------------------------------------------------------------------------------------------------------------------------------

(defclass DIALOGUEUR (TABLEAU-R)
  ((compteur :accessor compteur :initform 0)
   (pile-umi :accessor pile-umi :initform (list nil))
   (elements :accessor elements :initform (list nil))
   (structure-plate :accessor structure-plate :initform nil)

   ;; Niveaux qui concernent plus particulièrement les unités minimales d'interaction
   (niv-enoncem :accessor niv-enoncem :initform nil)
   (niv-enonceh :accessor niv-enonceh :initform nil)

   ;; Niveaux qui concernent plus particulièrement l'analyse des énoncés de l'usager
   (niv-lexemes :accessor niv-lexemes :initform nil :documentation "Les groupes catégorisés")

   ;; Niveaux qui concernent plus particulièrement la génération des énoncés
   (niv-actes :accessor niv-actes :initform nil)

   ;; Niveaux qui concernent plus particulièrement la création des intentions
   (niv-umms :accessor niv-umms :initform nil :documentation "Les umms du système")
   (niv-umms-temp :accessor niv-umms-temp :initform nil :documentation "Les umms pour travailler")

   ;; Niveaux qui concernent la gestion de la tâche applicative
   (niv-etats :accessor niv-etats :initform nil :documentation "Les états de la tâche")
   (niv-resu :accessor niv-resu :initform nil :documentation "Les résultats de la recherche dans l'annuaire")

))

;; -------------------------------------------------------------------------------------------------------------------------------
;; Méthodes du dialogueur

(defmacro pushr (-element -liste) `(setf ,-liste (append ,-liste (list ,-element))))

(defmethod initialiser ((oself DIALOGUEUR))
  "Méthode d'initialisation du dialogueur"
  (setf $structure-plate nil)
  (make-instance 'ETAT :expression 'debut-dial))

(defmethod nouv-umi ((oself DIALOGUEUR))
  (desactive-elements-courants oself)
  (incf $compteur)
  (setf (car $pile-umi) $compteur)
  (pushr $compteur $structure-plate)
  (tracer oself (separation 80))
  (tracer oself
    (format nil "Nouvelle unité minimale d'interaction UMI~A ~A" $compteur (construis $structure-plate)))
  (tracer oself (separation 80)))

(defmethod nouv-umr ((oself DIALOGUEUR))
  (tracer oself (separation 80))
  (tracer oself (format nil "Nouveau container incident"))
  (tracer oself (separation 80))
  (pushr 'deb $structure-plate)
  (push nil $elements)
  (push nil $pile-umi)
  (empiler oself))

(defmethod term-umr ((oself DIALOGUEUR))
  (tracer oself (separation 80))
  (tracer oself (format nil "Clôture du container incident"))
  (tracer oself (separation 80))
  (pushr 'fin $structure-plate)
  (desactive-elements-courants oself)
  (pop $elements)
  (pop $pile-umi)
  (depiler oself))

(defmethod desactive-elements-courants ((oself DIALOGUEUR))
  (loop for element in (car $elements) do
```

```
    (if (and (umi element) (disponible element))
       (setf (statut element) 'obsolete)
       (pushnew element $evenements :test 'meme)) ;; on remet les éléments non locaux
    (setf (courant element) nil)))

;; ------------------------------------------------------------------------------------------------------------------------
;; Instanciation du dialogueur
;; ------------------------------------------------------------------------------------------------------------------------

(defparameter *dialogueur* (make-instance 'DIALOGUEUR :trace-avec-agenda t :mode-pas-a-pas nil))

(defun nouvelle-umi () (nouv-umi *dialogueur*))
(defun nouvelle-umr () (nouv-umr *dialogueur*))
(defun terminer-umr () (term-umr *dialogueur*))

(defun lancer ()
  #+server
  (setf (trace-stream *dialogueur*)
  (open "trace-ami.txt" :direction :output :if-does-not-exist :create :if-exists :append))

  (let ((trace-stream (trace-stream *dialogueur*)))
    (format trace-stream "~2%~A" (make-string 130 :initial-element #\#))
    (format trace-stream "~%Traces du moteur AMI2 le ~A à ~A" (date) (heure))
    (format trace-stream "~%~A" (make-string 130 :initial-element #\#))
    (format trace-stream "~%~A" (separation 130))

    (reset *dialogueur*)
    (execution *dialogueur*)

    (format trace-stream "~%~A" (separation 130))
    (format trace-stream "~%Fin de la session le ~A à ~A" (date) (heure)))
)

(defun fermer ()
  #+(and clisp (not server)) (loop while (xterm:list-all-windows) do (xterm:kill (first (xterm:list-all-windows)))))

(defun e () (afficher-historique *dialogueur* t))

(defun compile-ksources-dialogueur () (compile-ksources *dialogueur*))

;; ------------------------------------------------------------------------------------------------------------------------
;; Éléments génériques du dialogueur
;; ------------------------------------------------------------------------------------------------------------------------

(defclass ELEMDIAL (ELEMENT unite)
  ((tableau :reader tableau :initarg :tableau :initform *dialogueur*)
   (courant :accessor courant :initform t)
   (umi :reader umi :initform (car (pile-umi *dialogueur*)))
   (ve :reader ve :initform (1- (length (elements *dialogueur*))))))

(defclass ELEMDIAL-NONLOC (ELEMDIAL) ((umi :accessor umi :initform nil)))

(defmethod initialize-instance :after ((oself ELEMDIAL) &rest initargs)
  (declare (ignore initargs))
  (push oself (car (elements $tableau))))

(defmethod en-test ((oself ELEMDIAL)) (equal $statut 'en-test))

(defmethod decrire ((oself ELEMDIAL))
  (format nil "~A~22T~A~A~30T~A ~A~46T[~A]~51T~A"
    oself
    (make-string $ve :initial-element #\space)
    (if $umi (format nil "UMI~A" $umi) '--)
    (if $courant "e" " ")
    $statut
    $interet
    $description))

;; ------------------------------------------------------------------------------------------------------------------------
;; Éléments du dialogueur
;; ------------------------------------------------------------------------------------------------------------------------

(defclass ETAT (ELEMDIAL-NONLOC)
  ((niveau :allocation :class :reader niveau :initform 'niv-etats)
   (expression :reader expression :initarg :expression)))

(defmacro cree-etat (&rest initargs) `(make-instance 'ETAT ,@initargs))
(defmethod description ((oself ETAT)) (format nil "[~A]" $expression))
(defmethod en-sommeil ((oself ETAT)) (equal $statut 'en-sommeil))

(defclass ENONCEH (ELEMDIAL)
  ((niveau :allocation :class :reader niveau :initform 'niv-enonceh)
   (expression :reader expression :initarg :expression)
   (expression-initiale :accessor expression-initiale)))

(defmethod initialize-instance :after ((oself ENONCEH) &rest initargs)
  (declare (ignore initargs))
  (setf $expression-initiale $expression))

(defmacro cree-enonceh (&rest initargs) `(make-instance 'ENONCEH ,@initargs))
```

```
(defmethod description ((oself ENONCEH)) (format nil "~A" $expression))

(defmacro declare-etats (&rest etats)
  (loop for etat in etats do
  (eval `(defmethod ,etat ((oself ETAT))
     (and $disponible (eq ',etat $expression))))))

(declare-etats debut-dial debut-tache)

;; ------------------------------------------------------------------------------------------------------------------------
```

Listing: activite-dialogique/umm.lsp

```
;; ##########################################################################################################################
;; ### AMI2 - UMM.LSP - Lemeunier 25/01/99 (Thierry.Lemeunier@lium.univ-lemans.fr) ##########################################
;; ##########################################################################################################################

(load "utiles/arbre.lsp")

;; --------------------------------------------------------------------------------------------------------------------------
;; Les umms (Unités Minimal de Mémoire)

(defclass UMM (ELEMDIAL-MONLOC ARBRE UNITE)
  ((niveau :allocation :class :reader niveau :initform 'niv-umms)
   (focus :accessor focus :initarg :focus :initform nil)
   (statut :accessor statut :initarg :statut :initform 'en-attente)
   (valeur :accessor valeur :initarg :valeur :initform nil)
   (demandable :accessor demandable :initarg :demandable :initform t)
   (connu :accessor connu :initarg :connu :initform nil)
   (anciens-actes :accessor anciens-actes :initarg :anciens-actes :initform nil)
   (texte-questionner :accessor texte-questionner :initarg :texte-questionner :initform '(""))
   (texte-proposer :accessor texte-proposer :initarg :texte-proposer :initform '(""))
   (texte-indiquer :accessor texte-indiquer :initarg :texte-indiquer :initform '(""))))

(defmacro cree-umm (&rest initargs) `(make-instance 'UMM ,@initargs))

(defmethod (setf pere) :after (valeur (oself UMM))
  "Ajuster la valeur du focus de l'UMM à celui de son père"
  (if valeur (setf $focus (focus valeur))))

(defmethod (setf focus) :after (valeur (oself UMM))
  (declare (ignore valeur))
  (pushnew oself (evenements $tableau) :test #'meme)
  (tracer $tableau (format nil "Modification du focus de ~A -> ~A" oself $focus)) )

(defmethod disponible ((oself UMM)) (and $focus (equal $statut 'disponible)))
(defmethod disponible-sf ((oself UMM)) (equal $statut 'disponible))

(defmethod en-test ((oself UMM)) (and $focus (equal $statut 'en-test)))
(defmethod en-test-sf ((oself UMM)) (equal $statut 'en-test))

(defmethod interprete ((oself UMM)) (and $focus (equal $statut 'interprete)))
(defmethod interprete-sf ((oself UMM)) (equal $statut 'interprete))

(defmethod en-attente ((oself UMM)) (and $focus (equal $statut 'en-attente)))
(defmethod en-attente-sf ((oself UMM)) (equal $statut 'en-attente))

(defmethod reussi ((oself UMM)) (and $focus (equal $statut 'reussi)))
(defmethod reussi-sf ((oself UMM)) (equal $statut 'reussi))

(defmethod accompli ((oself UMM)) (and $focus (equal $statut 'accompli)))
(defmethod accompli-sf ((oself UMM)) (equal $statut 'accompli))

(defmethod interessant ((oself UMM)) (or $en-attente $reussi $accompli))
(defmethod interessant-sf ((oself UMM)) (or $en-attente-sf $reussi-sf $accompli-sf $en-test-sf))

(defmethod en-sommeil ((oself UMM)) (equal $statut 'en-sommeil))

(defmethod description ((oself UMM))
  (format nil "UMM ~A ~A [Pere ~A]"
          (if $focus "[Focus]" "")
    (if $valeur (format nil "[Valeur ~A]" $valeur) "")
    $pere))

(defmethod disponible-pour-question ((oself UMM)) (and $en-attente $demandable (not $valeur)))

(defmethod disponible-pour-proposition ((oself UMM)) (and $en-attente $valeur (not $connu)))

(defmethod disponible-pour-indication ((oself UMM)) (and $accompli $valeur $demandable))

(defmethod disponible-pour-indication-negative ((oself UMM)) (and $reussi $demandable))

(defmethod devient-focus ((oself UMM))
   (setf $focus t)
   (loop for fils in $fils do (devient-focus fils))
   oself)

(defmethod enleve-focus ((oself UMM))
   (setf $focus nil)
   (loop for fils in $fils do (enleve-focus fils))
   oself)

;; --------------------------------------------------------------------------------------------------------------------------
;; Copie-profonde d'une hiérarchie d'UMM

(defmethod initialise-umm ((new-umm UMM) (old-umm UMM))
   "Initialisation des slots d'une umm à partir d'une umm"
   (setf (valeur new-umm) (valeur old-umm)
   (demandable new-umm) (demandable old-umm)
   (connu new-umm) (connu old-umm)
   (interet new-umm) (interet old-umm)
        (focus new-umm) (focus old-umm)
```

```
    (statut new-umm) (statut old-umm)
    (pere-stat new-umm) (pere-stat old-umm)
    (fils-stat new-umm) (fils-stat old-umm)

    (texte-questionner new-umm) (texte-questionner old-umm)
    (texte-proposer new-umm) (texte-proposer old-umm)
    (texte-indiquer new-umm) (texte-indiquer old-umm))
    new-umm)

(defmethod copie-profonde-1 ((umm UMM) &optional (pere nil))
  "Copier de façon profonde un arbre naires"
  (let ((racine (copie-umm umm)))
    (setf (pere racine) pere)
    (setf (fils racine)
          (loop for fils in (fils umm)
            collect (copie-profonde-1 fils racine)))
    racine))

(defmacro declare-copie-umm (&rest classes-umms)
  "Macro de création des fonctions de copie propre à chaque UMM"
  (loop for classe-umm in classes-umms do
  (let ((function-name (read-from-string (concatenate 'string "cree-umm-" (symbol-name classe-umm)))))
    (eval '(defmethod copie-umm ((oself ,classe-umm))
      (initialise-umm (eval (list ',function-name)) oself))))))

;; ------------------------------------------------------------------------------------------------------------------------
;; Ces fonctions doivent être ré-écrites de façon telle qu'elles s'appuient uniquement sur la description
;; de la hiérarchie statique des umms. Autrement dit, il faut supprimer le champ intérêt.

(defun umms-interessants (&optional (type 'UMM))
  "Lister les umms intéressantes (en attente, réussies ou accomplies)"
  (my-sort
   (my-find-if #'interessant-sf (tous-les type *dialogueur*))
   #'> :key #'interet))

;; ------------------------------------------------------------------------------------------------------------------------
;; Quelques fonctions pour l'affichage

(defun a ()
  "Lister toutes les umms du dialogueur"
  (loop for umm in (tous-les 'UMM *dialogueur*) do (format t "~%~A" (decrire umm))))

(defun ai ()
  "Afficher les umms intéressantes"
  (loop for hierarchie in (liste-en-hierarchie (umms-interessants)) do
    (loop for umm in hierarchie do
      (format t "~%~A" (decrire umm)))
    (format t "~%----------")))

;; ------------------------------------------------------------------------------------------------------------------------
;; Les umms composées
;;
;; Les umms composées sont utilisées pour le processus de génération des intentions
;; de communications pour les éventuelles opérations de regroupements.

(defclass UMM-COMPOSE (UMM ELEMDIAL)
  ((niveau :allocation :class :reader niveau :initform 'niv-umms-temp)
   (expression :accessor expression :initarg :expression :initform nil)))

(defmacro cree-umm-compose (&rest initargs) `(make-instance 'umm-compose ,@initargs))

(defmethod pere-origine ((oself UMM)) oself)

;;; (defmethod fils-origine ((oself UMM)) $fils)

(defmethod pere-origine ((oself UMM-COMPOSE))
  (if (consp $expression) (pere-origine (car $expression)) $expression))

(defmethod fils-origine ((oself UMM-COMPOSE))
  (if (consp $expression) (cdr $expression) nil))

(defmethod copie-noeud ((umm UMM))
  "Créer une umm temporaire à partir d'une umm - Redéfinition de la méthode par défaut"
  (cree-umm-compose :valeur (valeur umm)
        :demandable (demandable umm)
        :connu (connu umm)
        :focus (focus umm)
        :interet (interet umm)
        :statut (statut umm)
        :expression umm
        :pere-stat (pere-stat umm)
        :fils-stat (fils-stat umm)

        :texte-questionner (texte-questionner umm)
        :texte-proposer (texte-proposer umm)
        :texte-indiquer (texte-indiquer umm)))

;; ------------------------------------------------------------------------------------------------------------------------
```

Listing: activite-dialogique/umms-dial.lsp

```
; ###############################################################################################################################
; ### AMI2 - UMMS-DIAL.LSP - LEMEUNIER (version du 31/03/1999) Thierry.Lemeunier@lium.univ-lemans.fr ##################
; ###############################################################################################################################

;; -------------------------------------------------------------------------------------------------------------------------
;; Déclaration de la classe OUVERT-DIAL (l'ouverture du dialogue : acte de langage)

(defclass OUVERT-DIAL (UMM)
  ((texte-indiquer :reader texte-indiquer
      :initform '("système AMI, bonjour"))))

(defmacro cree-umm-ouvert-dial (&rest -args) `(make-instance 'OUVERT-DIAL ,@-args))

;; -------------------------------------------------------------------------------------------------------------------------
;; Déclaration de la classe PROPOSER (force illocutoire)

(defclass PROPOSER (UMM)
  ((interet :accessor interet :initarg :interet :initform 12)
   (fils-stat :reader fils-stat :initform '(ou lire-message laisser-message identifier-appele identifier-appelant identifier-lexeme))
   (texte-indiquer :reader texte-proposer
      :initform '("je vais vous faire une proposition"))
   (texte-proposer :reader texte-proposer
      :initform '("je vous propose une proposition"))))

(defmacro cree-umm-proposer (&rest -args) `(make-instance 'PROPOSER ,@-args))

;; -------------------------------------------------------------------------------------------------------------------------
;; Déclaration de la classe CONFIRMATION

(defclass CONFIRMATION (UMM)
  ((connu :accessor connu :initarg :connu :initform t)))

(defmacro cree-umm-confirmation (&rest -args) `(make-instance 'CONFIRMATION ,@-args))

;; -------------------------------------------------------------------------------------------------------------------------
;; Déclaration de la classe INFIRMATION

(defclass INFIRMATION (UMM)
  ((connu :accessor connu :initarg :connu :initform t)))

(defmacro cree-umm-infirmation (&rest -args) `(make-instance 'INFIRMATION ,@-args))

;; -------------------------------------------------------------------------------------------------------------------------
;; Déclaration des fonctions de copie

(declare-copie-umm OUVERT-DIAL PROPOSER CONFIRMATION INFIRMATION)
```

Listing: activite-dialogique/tache-dialogique.lsp

```
;; ###############################################################################################################################
;; ### AMI2 - TACHE-DIALOGIQUE.LSP - Lemeunier 03/11/98 (Thierry.Lemeunier@lium.univ-lemans.fr) ######################
;; ###############################################################################################################################

;; ===============================================================================================================================
;; Sources de connaissance du dialogueur récursif d'après le modèle COALA
;; ===============================================================================================================================

;; -------------------------------------------------------------------------------------------------------------------------------

(defregle-r ouverture-dialogue *dialogueur* 11000

  DOCUMENTATION
    "Ouverture de l'interaction"

  PRECONDITIONS
    niv-etats #'debut-dial

  ACTIONS
    (tracer *dialogueur* "--- Chargement des messages ------------------")
    (charge-messages)
    (tracer *dialogueur* "--- Fin du chargement des messages -----------")

    (cree-etat :expression 'debut-tache)

    ;; A revoir lors de la refonte du processus de négociation (l'état initial devra être changé en "en-attente")
    (cree-umm-ouvert-dial :statut 'accompli :valeur t :focus t)
)

;; -------------------------------------------------------------------------------------------------------------------------------

(defregle-r dialogue-en-cours *dialogueur* 8400

  DOCUMENTATION
    "Le dialogue est en cours : on fait le ménage. A revoir sans doute"

  PRECONDITIONS
    niv-umms #'(lambda (x) (and (accompli x)
        (typep x 'OUVERT-DIAL)
        (not (demandable x)))) -> umm-ouvert-dial

  ACTIONS
    (setf (statut umm-ouvert-dial) 'utilise)
)

;; -------------------------------------------------------------------------------------------------------------------------------

(defregle-r interaction *dialogueur* 500

  DOCUMENTATION
    "Affiche l'énoncé de la machine et instancie l'énoncé de l'utilisateur"

  PRECONDITIONS
    niv-enoncem existe #'disponible -> enonces

  ACTIONS
    (nouvelle-umi)

    (loop for enonce in (classement-selon-interet-decroissant enonces) do
    (setf (statut enonce) 'utilise)
    #-server
    (affiche-sans-fin (format nil "H~A> ~A~%" (compteur *dialogueur*) (expression enonce)) *stream-dialogue*)
    #+server
    (affiche-sans-fin (format nil "~A" (expression enonce) *stream-dialogue*)))

    (affiche "" *stream-dialogue*)

    #+server (lock)
    (cree-enonceh :expression (saisir (format nil "H~A> " (compteur *dialogueur*)) *stream-dialogue*))
)

;; -------------------------------------------------------------------------------------------------------------------------------

(defregle-r fermeture-dialogue *dialogueur* -300

  DOCUMENTATION
    "Fermeture de l'interaction"

  PRECONDITIONS
    niv-etats existe #'vrai -> les-etats
    niv-umms existe #'interessant -> tous-les-umms

  ACTIONS
    (loop for etat in les-etats do
      (setf (statut etat) 'fin-dialogue))
    (loop for umm in tous-les-umms
      unless (equal 'en-sommeil (statut umm))
      do (setf (statut umm) 'fin-dialogue))
```

```
90      (affiche
         #-server
         (format nil
           "M~A> Merci d'avoir utilisé notre service de messagerie. Au revoir."
           (+ 1 (compteur *dialogueur*)))
95       #+server
         (format nil "Merci d'avoir utilisé notre service de messagerie. Au revoir.")
         *stream-dialogue*)

        (tracer *dialogueur* "--- Sauvegarde des messages ------------------")
100     (sauve-messages)

        #+server (progn (signalFin) (unlock) (quit))
    )

105 ;; ------------------------------------------------------------------------------------------------------------------------
```

Listing: activite-langagiere/umms-lang.lsp

```
; ##########################################################################################################################
; ### AMI2 - UMMS-LANG.LSP - LEMEUNIER (version du 05/10/2000) Thierry.Lemeunier@lium.univ-lemans.fr ###################
; ##########################################################################################################################

;; ------------------------------------------------------------------------------------------------------------------------
;; Action d'identification d'un lexeme

(defclass IDENTIFIER-LEXEME (UMM)
  ((interet :accessor interet :initarg :interet :initform 10)
   (fils-stat :reader fils-stat :initform '(et source cible))
   (texte-questionner :reader texte-questionner
         :initform '("quel est la signification de l'expression '~A' ?" t))
   (texte-proposer :reader texte-proposer
       :initform '("~A ?" t))
   (texte-indiquer :reader texte-indiquer
       :initform '("j'ai compris la signification de l'expression '~A'" t))))

(defmacro cree-umm-identifier-lexeme (&rest -args) `(make-instance 'IDENTIFIER-LEXEME ,@-args))

;; ------------------------------------------------------------------------------------------------------------------------
;; La source à identifier

(defclass SOURCE (UMM)
  ((pere-stat :reader pere-stat :initform '(et identifier-lexeme))
   (texte-questionner :reader texte-questionner
         :initform '("qu'avez-vous voulu dire avec '~A' ?" t))
   (texte-proposer  :reader texte-proposer
         :initform '("avez-vous dit '~A' ?" t))
   (texte-indiquer  :reader texte-indiquer
         :initform '("vous avez dit '~A'" t))))

(defmacro cree-umm-source (&rest args) `(make-instance 'SOURCE ,@args))

;; ------------------------------------------------------------------------------------------------------------------------
;; Les cibles potentiels

(defclass CIBLE (UMM)
  ((pere-stat :reader pere-stat :initform '(et identifier-lexeme))
   (fils-stat :reader fils-stat :initform '(ou requete parler lire-message laisser-message identifier-appele
           identifier-appelant contacts appele appelant message nom prenom poste
           ouvert-dial proposer confirmation infirmation))
   (texte-questionner :reader texte-questionner
         :initform '("est-ce que vous avez signifé ~A ?" t))
   (texte-proposer  :reader texte-proposer
         :initform '("avez-vous désigné ~A ?" t))
   (texte-indiquer  :reader texte-indiquer
         :initform '("vous avez donné ~A" t))))

(defmacro cree-umm-cible (&rest args) `(make-instance 'CIBLE ,@args))

;; ------------------------------------------------------------------------------------------------------------------------
;; Déclaration des fonctions de copie

(declare-copie-umm IDENTIFIER-LEXEME SOURCE CIBLE)
```

Listing: activite-langagiere/analyseur.lsp

```
;; ###############################################################################################################
;; ### AMI2 - ANALYSEUR.LSP - LEMEUNIER 12/01/2000 (Thierry.Lemeunier@lium.univ-lemans.fr) ########################
;; ###############################################################################################################

;; ---------------------------------------------------------------------------------------------------------------
;; Les groupes catégorisés

(defclass LEXEME (ELEMDIAL UNITE)
  ((niveau :allocation :class :reader niveau :initform 'niv-lexemes)
   (hypothetique :accessor hypothetique :initarg :hypothetique :initform nil)
   (constituants :accessor constituants :initarg :constituants :initform nil)
   (pos :accessor pos :initarg :pos)
   (origine :accessor origine :initarg :origine :initform nil)
   (umms :accessor umms :initarg :umms :initform nil)
   (cat :accessor cat :initarg :cat)
   (pub :accessor pub :initarg :pub)
   (expression :accessor expression :initarg :expression)))

(defmacro cree-lexeme (&rest initargs) `(make-instance 'LEXEME ,@initargs))

(defmethod description ((oself LEXEME))
  (format nil "Lexeme~A Cat: ~A Expr: ~A Cons: ~A Orig: ~A Umms: ~A"
    (if $hypothetique " [hyp]" "")
    $cat
    $expression
    $constituants
    $origine
    $umms))

(defmethod supprime ((oself LEXEME)) (eq $statut 'supprime))
(defmethod apparie ((oself LEXEME)) (eq $statut 'apparie))
(defmethod en-conflit ((oself LEXEME)) (eq $statut 'en-conflit))
(defmethod en-attente ((oself LEXEME)) (eq $statut 'en-attente))
(defmethod en-test ((oself LEXEME)) (eq $statut 'en-test))

(defmethod supprime-non-terminale ((oself LEXEME))
  "Supprimer les umms nom-terminaux d'un lexème"
  (setf $umms
  (loop for type-umm in $umms
    when (member type-umm *liste-umms-terminales*)
    collect type-umm)))

(defmethod supprime-terminale ((oself LEXEME))
  "Supprimer les umms terminaux d'un lexème"
  (setf $umms
  (loop for type-umm in $umms
    unless (member type-umm *liste-umms-terminales*)
    collect type-umm)))

(defun l ()
  "Afficher tous les lexemes du dialogueur"
  (loop for lexeme in (lexemes-interessantes) do (format t "~%~A" (decrire lexeme))))

(defun lexemes-interessantes ()
  "Lister les lexemes intéressantes (disponible et apparie)"
  (my-find-if #'(lambda (x) (or (disponible x) (apparie x))) (tous-les 'LEXEME *dialogueur*)))

;; ===============================================================================================================
;; Sources de connaissance de l'analyseur séquentiel
;; ===============================================================================================================
;; L'analyse se fait en fait par l'interpréteur CLIPS.
;; Cette partie sert d'interface entre Clisp et CLIPS.

;; ---------------------------------------------------------------------------------------------------------------

(defregle-r analyse *dialogueur* 50000

  DOCUMENTATION
    "Analyse l'énoncé de l'utilisateur"

  PRECONDITIONS
    niv-enonceh #'disponible -> enonce

  ACTIONS
    ;; Création du fichier phrase à analyser
    (when (probe-file *fichier-phrase*) (delete-file *fichier-phrase*))
    (with-open-file
     (ofile *fichier-phrase* :direction :output :if-exists :overwrite :if-does-not-exist :create)
     (format ofile "~(~A~)" (cons 'analyser (str2cons (expression enonce)))))

    ;; On fait place nette avant de lancer l'analyse
    (when (probe-file *fichier-sortie*) (delete-file *fichier-sortie*))
    (when (probe-file *fichier-liste-facts*) (delete-file *fichier-liste-facts*))

    ;; On lance l'analyse
    (run-program "clips" :arguments '("-f2" ,*fichier-analyse*))

    ;; On récupère le résultat
    (let (list-facts lex)
```

```
      (with-open-file
       (ifile *fichier-liste-facts* :direction :input) ;; Qu'est-ce qui se passe s'il n'y a pas de fichier ?
       (setf liste-facts (reverse (loop while (listen ifile) do
            append (read-from-string (format nil "(~S)" (read ifile)))))))
      (with-open-file
       (ifile *fichier-sortie* :direction :input)
       (loop while (listen ifile) do
    (setq lex (read-from-string (read-line ifile)))
    (cree-lexeme
     :pos (cadr (second lex))
     :hypothetique (case (cadr (third lex)) (FALSE nil) (TRUE t))
     :cat (cadr (fifth lex))
     :expression (str2cons (constostr (cdr (sixth lex))))
     :constituants (if (> 0 (cadr (second lex))) nil (cdr (eighth lex)))
     :origine (car liste-facts)
     :umms (cdr (ninth lex)))
    (setf liste-facts (cdr liste-facts)))))
)

;; ------------------------------------------------------------------------------------------------------------------------

(defregle-r traduit-analyse *dialogueur* 50000

  DOCUMENTATION
    "On met à jour les champs constituants et origine.
     Les lexemes sont classés de la manière suivante :
       - d'abord les lexemes non-hypothétiques
       - puis ensuite les lexemes hypothetiques"

  PRECONDITIONS
    niv-lexemes collect #'disponible -> liste-lexemes
    niv-enonceh #'disponible -> enonce

  ACTIONS
    (setf (statut enonce) 'utilise)
    (loop for lexeme in liste-lexemes
      when (= 0 (pos lexeme))
      do (loop while (numberp (car (constituants lexeme))) do
     (loop for lexeme2 in liste-lexemes
       when (and (< 0 (origine lexeme2)) (= (car (constituants lexeme)) (origine lexeme2)))
       do (progn
       (setf (constituants lexeme) (reverse (cons lexeme2 (reverse (cdr (constituants lexeme))))))
       (return nil)))))

    (loop for lexeme in liste-lexemes do
      (reactiver lexeme)                    ;;; IL FAUT REACTIVER LES LEXEMES POUR PROVOQUER L'APPARIEMENT !!!!!
      (setf (origine lexeme) nil))
)

;; ========================================================================================================================
```

Listing: activite-langagiere/analyseur.clp

```
;; ============================================
;; Fichier : analyseur.clp
;; Langage : CLIPS 6.10
;; Auteurs : Jérôme Lehuen & Thierry Lemeunier
;; Version : mardi 03 février 2000
;; ============================================

;; Analyse à complexité linéaire par trigrammes
;; L'ancrage et la propagation de l'ancrage défini des ilôts de confiance
;; On part d'un ancrage bilatéral et on propage la vague à gauche et à droite de ce point de départ
;; Deux vagues d'ancrage peuvent se rencontrer ce qui provoque un ancrage bilatéral

;; Analyse en quatre phases séquentielles :
;; 1) Création des lexèmes à partir des catégories
;;    et création des hypothèses à partir des trigrammes et de lexèmes créées.
;; 2) Ancrage, propagation des ancrages et suppression des lexèmes conflictuels
;; 3) Création des actes
;; 4) Suppression des actes synonymes

(defglobal
  ?*fichier-categories* = "/lium/buster1/lemeunier/recherche/commonlisp/ami/activite-langagiere/categories.clp"
  ?*fichier-phrase* = "/lium/buster1/lemeunier/recherche/commonlisp/ami/activite-langagiere/phrase.clp"
  ?*fichier-sortie* = "/lium/buster1/lemeunier/recherche/commonlisp/ami/activite-langagiere/sortie-analyse.clp"
  ?*fichier-liste-facts* = "/lium/buster1/lemeunier/recherche/commonlisp/ami/activite-langagiere/liste-facts.clp")

;(defglobal
; ?*fichier-categories* = "/lium/buster1/lemeunier/public_html/serveur-ami/categories.clp"
; ?*fichier-phrase* = "/lium/buster1/lemeunier/public_html/serveur-ami/phrase.clp"
; ?*fichier-sortie* = "/lium/buster1/lemeunier/public_html/serveur-ami/sortie-analyse.clp"
; ?*fichier-liste-facts* = "/lium/buster1/lemeunier/public_html/serveur-ami/liste-facts.clp")

;(defglobal
; ?*fichier-categories* = "/home/thierry/recherche/commonlisp/ami/activite-langagiere/categories.clp"
; ?*fichier-phrase* = "/home/thierry/recherche/commonlisp/ami/activite-langagiere/phrase.clp"
; ?*fichier-sortie* = "/home/thierry/recherche/commonlisp/ami/activite-langagiere/sortie-analyse.clp"
; ?*fichier-liste-facts* = "/home/thierry/recherche/commonlisp/ami/activite-langagiere/liste-facts.clp")

(deffunction non-vide$ ($?liste) (> (length$ ?liste) 0))

(deftemplate acte
  (multislot suite)
  (multislot umms))

(deftemplate categorie
  (slot nom (type SYMBOL))
  (multislot umms (type SYMBOL))
  (multislot expression))

(deftemplate trigramme
  (slot avant (type SYMBOL))
  (slot categorie (type SYMBOL))
  (slot apres (type SYMBOL)))

(deftemplate lexeme
  (slot position (type INTEGER))
  (slot hypothese (allowed-symbols FALSE TRUE) (default FALSE))
  (slot ancrage (allowed-symbols nil gauche droit bilateral))
  (slot categorie (type SYMBOL))
  (multislot expression)
  (multislot liste-position)
  (multislot constituants)
  (multislot umms (type SYMBOL)))

;; ------------------------------------------------------------------------------------------------------------------

(defrule initialisation
  =>
  (load-facts ?*fichier-categories*)
  (load-facts ?*fichier-phrase*))

;; ------------------------------------------------------------------------------------------------------------------

(defrule debut-phase1
  (declare (salience 999))
  (analyser $?enonce)
  =>
  (assert (iteration 1 ?enonce))
  (assert (phase1)))

(defrule creer-lexeme
  "Pour générer un lexeme non hypothétique (06/01/00)"
  (declare (salience 90))
  (phase1)
  (iteration ?position ?mot $?reste)
  ?categorie <- (categorie (expression ?mot $?mots) (umms $?umms))
  (test (eq ?mots (subseq$ ?reste 1 (length ?mots))))
  =>
  (assert (lexeme
    (position ?position)
```

```
    (categorie (fact-slot-value ?categorie nom))
    (expression (fact-slot-value ?categorie expression))
    (umms ?umms))))

(defrule mots-suivants
  "Pour progresser dans l'analyse des lexemes non hypothétiques (06/01/00)"
  (declare (salience 80))
  (phase1)
  ?p <- (iteration ?position ?lettre $?reste&:(non-vide$ ?reste))
  =>
  (retract ?p)
  (assert (iteration (+ ?position 1) ?reste)))

(defrule creer-lexeme-hypothetique-1
  "Pour générer un lexeme hypothétique au milieu de l'énoncé (06/01/00)"
  (declare (salience 70))
  (phase1)
  (analyser $?enonce)
  ?g1 <- (lexeme (hypothese FALSE) (position ?pos1) (categorie ?cat1) (expression $?expr1))
  ?g2 <- (lexeme (hypothese FALSE) (position ?pos2&:(> ?pos2 (+ ?pos1 (length ?expr1)))) (categorie ?cat2))
  ?trigramme <- (trigramme (avant ?cat1) (categorie ?cat) (apres ?cat2))
  (categorie (nom ?cat) (umms $?umms))
  =>
  (assert (lexeme
    (hypothese TRUE)
    (position (+ ?pos1 (length ?expr1)))
    (categorie (fact-slot-value ?trigramme categorie))
    (expression (subseq$ ?enonce (+ ?pos1 (length ?expr1)) (- ?pos2 1)))
    (umms ?umms))))

(defrule creer-lexeme-hypothetique-2
  "Pour générer un lexeme hypothétique au début d'un énoncé (06/01/00)"
  (declare (salience 70))
  (phase1)
  (analyser $?enonce)
  ?g <- (lexeme (hypothese FALSE) (position ?pos&:(> ?pos 1)) (categorie ?cat1))
  ?trigramme <- (trigramme (avant nil) (categorie ?cat) (apres ?cat1))
  (categorie (nom ?cat) (umms $?umms))
  =>
  (assert (lexeme
    (hypothese TRUE)
    (position 1)
    (categorie (fact-slot-value ?trigramme categorie))
    (expression (subseq$ ?enonce 0 (- ?pos 1)))
    (umms ?umms))))

(defrule creer-lexeme-hypothetique-3
  "Pour générer un lexeme hypothétique à la fin d'un énoncé (06/01/00)"
  (declare (salience 70))
  (phase1)
  (analyser $?enonce)
  ?g <- (lexeme (hypothese FALSE) (position ?pos) (categorie ?cat1) (expression $?expr))
  (test (<= (+ ?pos (length ?expr)) (length ?enonce)))
  ?trigramme <- (trigramme (avant ?cat1) (categorie ?cat) (apres nil))
  (categorie (nom ?cat) (umms $?umms))
  =>
  (assert (lexeme
    (hypothese TRUE)
    (position (+ ?pos (length ?expr)))
    (categorie (fact-slot-value ?trigramme categorie))
    (expression (subseq$ ?enonce (+ ?pos (length ?expr)) (length ?enonce)))
    (umms ?umms))))

;; ------------------------------------------------------------------------------------------------------------------

(defrule debut-phase2
  (declare (salience 65))
  ?p <- (phase1)
  =>
  (retract ?p)
  (assert (phase2)))

(defrule ancrer-lexeme
  "Pour ancrer un lexeme non hypothétique (19/12/99)"
  (declare (salience 60))
  (phase2)

  ?g1 <- (lexeme (position ?pos) (categorie ?cat) (expression $?expr) (hypothese TRUE))
  ?g2 <- (lexeme (position ?pos) (categorie ?cat) (expression $?expr) (hypothese FALSE) (ancrage nil))
  =>
  (retract ?g1)
  (modify ?g2 (ancrage bilateral)))

(defrule supprimer-lexeme-hypothetique
  "Pour supprimer un lexeme hypothétique en conflit avec un lexeme non hypothétique (19/12/99)"
  (declare (salience 50))
  (phase2)

  ?g1 <- (lexeme (categorie ?cat) (position ?pos1) (expression $?expr1) (hypothese TRUE))
  ?g2 <- (lexeme (categorie ?cat) (position ?pos2) (expression $?expr2) (hypothese FALSE))
```

```
180
      ; ?g1 <- (lexeme (position ?pos1) (expression $?expr1) (hypothese TRUE))
      ; ?g2 <- (lexeme (position ?pos2) (expression $?expr2) (hypothese FALSE))

        (test (or (and (>= ?pos1 ?pos2) (< ?pos1 (+ ?pos2 (length ?expr2))))
185             (and (>= ?pos2 ?pos1) (< ?pos2 (+ ?pos1 (length ?expr1))))))
        =>
        (retract ?g1))

      (defrule propager-ancrage-vers-gauche
190     "Pour propager un ancrage vers la gauche - une propagation ne peut pas changer de direction (22/12/99)"
        (declare (salience 40))
        (phase2)

        ?g1 <- (lexeme (position ?pos1) (categorie ?cat1) (expression $?expr1) (ancrage ?ancrage&nil|gauche))
195     ?g2 <- (lexeme (position ?pos2) (categorie ?cat2) (expression $?expr2) (ancrage bilateral|droit))
        (test (= ?pos2 (+ ?pos1 (length ?expr1))))
        (or (trigramme (avant ?cat1) (categorie ?cat2))
            (trigramme (apres ?cat2) (categorie ?cat1)))
        =>
200     (switch ?ancrage
          (case nil then (modify ?g1 (ancrage droit)))
          (case gauche then (modify ?g1 (ancrage bilateral)))))

      (defrule propager-ancrage-vers-droite
205     "Pour propager un ancrage vers la droite - une propagation ne peut pas changer de direction (22/12/99)"
        (declare (salience 40))
        (phase2)

        ?g1 <- (lexeme (position ?pos1) (categorie ?cat1) (expression $?expr1) (ancrage bilateral|gauche))
210     ?g2 <- (lexeme (position ?pos2) (categorie ?cat2) (expression $?expr2) (ancrage ?ancrage&nil|droit))
        (test (= ?pos2 (+ ?pos1 (length ?expr1))))
        (or (trigramme (avant ?cat1) (categorie ?cat2))
            (trigramme (apres ?cat2) (categorie ?cat1)))
        =>
215     (switch ?ancrage
          (case nil then (modify ?g2 (ancrage gauche)))
          (case droit then (modify ?g2 (ancrage bilateral)))))

      ;; Ces règles posent des problèmes dans certains cas
220
      (defrule supprimer-lexeme-libre
        "Pour supprimer un lexeme libre en conflit avec un lexeme ancré (11/02/2000)"
        (declare (salience 30))
        (phase2)
225     ?g1 <- (lexeme (hypothese ?hypo) (position ?pos1) (expression $?expr1) (ancrage nil))
        ?g2 <- (lexeme (hypothese ?hypo) (position ?pos2) (expression $?expr2) (ancrage ~nil))
        (test (or (and (>= ?pos1 ?pos2) (< ?pos1 (+ ?pos2 (length ?expr2))))
                  (and (>= ?pos2 ?pos1) (< ?pos2 (+ ?pos1 (length ?expr1))))))
        =>
230     (retract ?g1))

      (defrule supprimer-lexeme-ancre
        "Pour supprimer un lexeme ancré unilatéralement en conflit avec un lexeme ancré bilatéralement (11/02/2000)";
        (declare (salience 30))
235     (phase2)

        ?g1 <- (lexeme (hypothese ?hypo) (position ?pos1) (expression $?expr1) (ancrage gauche|droit))
        ?g2 <- (lexeme (hypothese ?hypo) (position ?pos2) (expression $?expr2) (ancrage bilateral))
        (test (or (and (>= ?pos1 ?pos2) (< ?pos1 (+ ?pos2 (length ?expr2))))
240             (and (>= ?pos2 ?pos1) (< ?pos2 (+ ?pos1 (length ?expr1))))))
        =>
        (retract ?g1))

      ;; ------------------------------------------------------------------------------------------------------------------------
245
      (defrule debut-phase3
        (declare (salience 25))
        ?p <- (phase2)
        =>
250     (retract ?p)
        (assert (phase3)))

      (defrule creer-acte-longueur-7
        "Pour créer un acte de dialogue à partir de sept categories (16/02/00)"
255     (declare (salience 25))
        (phase3)
        (acte (suite ?cat1 ?cat2 ?cat3 ?cat4 ?cat5 ?cat6 ?cat7) (umms $?umms))
        ?lex1 <- (lexeme (categorie ?cat1) (position ?pos1) (hypothese ?hypo1) (expression $?expr1))
        ?lex2 <- (lexeme (categorie ?cat2) (position ?pos2) (hypothese ?hypo2) (expression $?expr2))
260     ?lex3 <- (lexeme (categorie ?cat3) (position ?pos3) (hypothese ?hypo3) (expression $?expr3))
        ?lex4 <- (lexeme (categorie ?cat4) (position ?pos4) (hypothese ?hypo4) (expression $?expr4))
        ?lex5 <- (lexeme (categorie ?cat5) (position ?pos5) (hypothese ?hypo5) (expression $?expr5))
        ?lex6 <- (lexeme (categorie ?cat6) (position ?pos6) (hypothese ?hypo6) (expression $?expr6))
        ?lex7 <- (lexeme (categorie ?cat7) (position ?pos7) (hypothese ?hypo7) (expression $?expr7))
265     (test (and (= (+ ?pos1 (length ?expr1)) ?pos2)
                   (= (+ ?pos2 (length ?expr2)) ?pos3)
                   (= (+ ?pos3 (length ?expr3)) ?pos4)
                   (= (+ ?pos4 (length ?expr4)) ?pos5)
                   (= (+ ?pos5 (length ?expr5)) ?pos6)
```

```
            (= (+ ?pos6 (length ?expr6)) ?pos7)))
  =>
  (assert (lexeme
        (expression (create$ ?expr1 ?expr2 ?expr3 ?expr4 ?expr5 ?expr6 ?expr7))
    (hypothese (or ?hypo1 ?hypo2 ?hypo3 ?hypo4 ?hypo5 ?hypo6 ?hypo7))
    (umms ?umms)
    (liste-position (create$ ?pos1 ?pos2 ?pos3 ?pos4 ?pos5 ?pos6 ?pos7))
    (constituants (create$ (fact-index ?lex1) (fact-index ?lex2) (fact-index ?lex3) (fact-index ?lex4)
              (fact-index ?lex5) (fact-index ?lex6) (fact-index ?lex7))))))

(defrule creer-acte-longueur-6
  "Pour créer un acte de dialogue à partir de six categories (16/02/00)"
  (declare (salience 24))
  (phase3)
  (acte (suite ?cat1 ?cat2 ?cat3 ?cat4 ?cat5 ?cat6) (umms $?umms))
  ?lex1 <- (lexeme (categorie ?cat1) (position ?pos1) (hypothese ?hypo1) (expression $?expr1))
  ?lex2 <- (lexeme (categorie ?cat2) (position ?pos2) (hypothese ?hypo2) (expression $?expr2))
  ?lex3 <- (lexeme (categorie ?cat3) (position ?pos3) (hypothese ?hypo3) (expression $?expr3))
  ?lex4 <- (lexeme (categorie ?cat4) (position ?pos4) (hypothese ?hypo4) (expression $?expr4))
  ?lex5 <- (lexeme (categorie ?cat5) (position ?pos5) (hypothese ?hypo5) (expression $?expr5))
  ?lex6 <- (lexeme (categorie ?cat6) (position ?pos6) (hypothese ?hypo6) (expression $?expr6))
  (test (and (= (+ ?pos1 (length ?expr1)) ?pos2)
             (= (+ ?pos2 (length ?expr2)) ?pos3)
             (= (+ ?pos3 (length ?expr3)) ?pos4)
             (= (+ ?pos4 (length ?expr4)) ?pos5)
             (= (+ ?pos5 (length ?expr5)) ?pos6)))
  =>
  (assert (lexeme
    (expression (create$ ?expr1 ?expr2 ?expr3 ?expr4 ?expr5 ?expr6))
    (hypothese (or ?hypo1 ?hypo2 ?hypo3 ?hypo4 ?hypo5 ?hypo6))
    (umms ?umms)
    (liste-position (create$ ?pos1 ?pos2 ?pos3 ?pos4 ?pos5 ?pos6))
    (constituants (create$ (fact-index ?lex1) (fact-index ?lex2) (fact-index ?lex3) (fact-index ?lex4)
              (fact-index ?lex5) (fact-index ?lex6))))))

(defrule creer-acte-longueur-5
  "Pour créer un acte de dialogue à partir de cinq categories (16/02/00)"
  (declare (salience 23))
  (phase3)
  (acte (suite ?cat1 ?cat2 ?cat3 ?cat4 ?cat5) (umms $?umms))
  ?lex1 <- (lexeme (categorie ?cat1) (position ?pos1) (hypothese ?hypo1) (expression $?expr1))
  ?lex2 <- (lexeme (categorie ?cat2) (position ?pos2) (hypothese ?hypo2) (expression $?expr2))
  ?lex3 <- (lexeme (categorie ?cat3) (position ?pos3) (hypothese ?hypo3) (expression $?expr3))
  ?lex4 <- (lexeme (categorie ?cat4) (position ?pos4) (hypothese ?hypo4) (expression $?expr4))
  ?lex5 <- (lexeme (categorie ?cat5) (position ?pos5) (hypothese ?hypo5) (expression $?expr5))
  (test (and (= (+ ?pos1 (length ?expr1)) ?pos2)
             (= (+ ?pos2 (length ?expr2)) ?pos3)
             (= (+ ?pos3 (length ?expr3)) ?pos4)
             (= (+ ?pos4 (length ?expr4)) ?pos5)))
  =>
  (assert (lexeme
    (expression (create$ ?expr1 ?expr2 ?expr3 ?expr4 ?expr5))
    (hypothese (or ?hypo1 ?hypo2 ?hypo3 ?hypo4 ?hypo5))
    (umms ?umms)
    (liste-position (create$ ?pos1 ?pos2 ?pos3 ?pos4 ?pos5))
    (constituants (create$ (fact-index ?lex1) (fact-index ?lex2) (fact-index ?lex3) (fact-index ?lex4)
              (fact-index ?lex5))))))

(defrule creer-acte-longueur-4
  "Pour créer un acte de dialogue à partir de quatre categories (16/02/00)"
  (declare (salience 22))
  (phase3)
  (acte (suite ?cat1 ?cat2 ?cat3 ?cat4) (umms $?umms))
  ?lex1 <- (lexeme (categorie ?cat1) (position ?pos1) (hypothese ?hypo1) (expression $?expr1))
  ?lex2 <- (lexeme (categorie ?cat2) (position ?pos2) (hypothese ?hypo2) (expression $?expr2))
  ?lex3 <- (lexeme (categorie ?cat3) (position ?pos3) (hypothese ?hypo3) (expression $?expr3))
  ?lex4 <- (lexeme (categorie ?cat4) (position ?pos4) (hypothese ?hypo4) (expression $?expr4))
  (test (and (= (+ ?pos1 (length ?expr1)) ?pos2)
             (= (+ ?pos2 (length ?expr2)) ?pos3)
             (= (+ ?pos3 (length ?expr3)) ?pos4)))
  =>
  (assert (lexeme
    (expression (create$ ?expr1 ?expr2 ?expr3 ?expr4))
    (hypothese (or ?hypo1 ?hypo2 ?hypo3 ?hypo4))
    (umms ?umms)
    (liste-position (create$ ?pos1 ?pos2 ?pos3 ?pos4))
    (constituants (create$ (fact-index ?lex1) (fact-index ?lex2) (fact-index ?lex3) (fact-index ?lex4)
              (fact-index ?lex5))))))

(defrule creer-acte-longueur-3
  "Pour créer un acte de dialogue à partir de trois categories (16/02/00)"
  (declare (salience 21))
  (phase3)
  (acte (suite ?cat1 ?cat2 ?cat3) (umms $?umms))
  ?lex1 <- (lexeme (categorie ?cat1) (position ?pos1) (hypothese ?hypo1) (expression $?expr1))
  ?lex2 <- (lexeme (categorie ?cat2) (position ?pos2) (hypothese ?hypo2) (expression $?expr2))
  ?lex3 <- (lexeme (categorie ?cat3) (position ?pos3) (hypothese ?hypo3) (expression $?expr3))
  (test (and (= (+ ?pos1 (length ?expr1)) ?pos2)
             (= (+ ?pos2 (length ?expr2)) ?pos3)))
  =>
  (assert (lexeme
```

```
      (expression (create$ ?expr1 ?expr2 ?expr3))
      (hypothese (or ?hypo1 ?hypo2 ?hypo3))
      (umms ?umms)
      (liste-position (create$ ?pos1 ?pos2 ?pos3))
      (constituants (create$ (fact-index ?lex1) (fact-index ?lex2) (fact-index ?lex3))))))

(defrule creer-acte-longueur-2
  "Pour créer un acte de dialogue à partir de deux categories (16/02/00)"
  (declare (salience 20))
  (phase3)
  (acte (suite ?cat1 ?cat2) (umms $?umms))
  ?lex1 <- (lexeme (categorie ?cat1) (position ?pos1) (hypothese ?hypo1) (expression $?expr1))
  ?lex2 <- (lexeme (categorie ?cat2) (position ?pos2) (hypothese ?hypo2) (expression $?expr2))
  (test (= (+ ?pos1 (length ?expr1)) ?pos2))
  =>
  (assert (lexeme
    (expression (create$ ?expr1 ?expr2))
    (hypothese (or ?hypo1 ?hypo2))
    (umms ?umms)
    (liste-position (create$ ?pos1 ?pos2))
    (constituants (create$ (fact-index ?lex1) (fact-index ?lex2))))))

;; ------------------------------------------------------------------------------------------------------------------------

(deffunction commun$ (?liste1 ?liste2)
  (if (non-vide$ ?liste1)
     then (if (member$ (first$ ?liste1) ?liste2)
       then (return TRUE)
       else (return (commun$ (create$ (rest$ ?liste1)) ?liste2)))
     else (return FALSE)))

(defrule debut-phase4
  (declare (salience 10))
  ?p <- (phase3)
  =>
  (retract ?p)
  (assert (phase4)))

;; Solution demandant beaucoup de temps mais il y a très peu d'actes

(defrule supprimer-acte-synonynme-1
  "Pour supprimer des actes synonymes - Heuristique 1 (09/02/2000)"
  (declare (salience 5))
  (phase4)
  ?lex1 <- (lexeme (position 0) (hypothese ?hypo) (umms $?reste1 $?match&:(non-vide$ ?match) $?reste2)
       (liste-position $?lpos1))
  ?lex2 <- (lexeme (position 0) (hypothese ?hypo) (umms $?reste3 $?match $?reste4) (liste-position $?lpos2))
  (test (and (neq ?lex1 ?lex2)
       (>= (length $?lpos1) (length $?lpos2))
            (commun$ ?lpos1 ?lpos2)))
  =>
  (retract ?lex2))

(defrule supprimer-acte-synonyme-2
  "Pour supprimer des actes synonymes - Heuristique 2 (09/02/2000)"
  (declare (salience 5))
  (phase4)
  ?lex1 <- (lexeme (position 0) (hypothese FALSE) (umms $?reste1 $?match&:(non-vide$ ?match) $?reste2)
               (liste-position $?lpos1))
  ?lex2 <- (lexeme (position 0) (hypothese TRUE) (umms $?reste3 $?match $?reste4) (liste-position $?lpos2))
  (test (and (neq ?lex1 ?lex2)
       (>= (length $?lpos1) (length $?lpos2))
            (commun$ ?lpos1 ?lpos2)))
  =>
  (retract ?lex2))

;; ------------------------------------------------------------------------------------------------------------------------

(defrule pres-fin-analyse
  "Pour sauvegarder les numeros des faits"
  (declare (salience 1))
  ?lex <- (lexeme)
  =>
  (open ?*fichier-liste-facts* fichier-sortie "a")
  (format fichier-sortie "%d " (fact-index ?lex))
  (close fichier-sortie))

(defrule fin-analyse
  "Pour sauvegarder les faits"
  =>
  (save-facts ?*fichier-sortie* local lexeme))

;; ------------------------------------------------------------------------------------------------------------------------
;; On initialise, on lance puis on quitte.

(reset)
(run)
(exit)
```

Listing: activite-langagiere/categories.clp (extrait)

```
(categorie (nom cat01) (expression madame))
(categorie (nom cat01) (expression monsieur))
(categorie (nom cat01) (expression m))
(categorie (nom cat01) (expression mr))
(categorie (nom cat01) (expression mde))
(categorie (nom cat01) (expression mademoiselle))

(categorie (nom ouverture-dial) (expression bonjour))
(categorie (nom fermeture-dial) (expression au revoir))
(categorie (nom fermeture-dial) (expression salut))
(categorie (nom fermeture-dial) (expression ciao))

(categorie (nom connecteur-et) (expression et))

(categorie (nom connecteur-ou) (expression ou))

(categorie (nom forme-politesse) (expression svp))
(categorie (nom forme-politesse) (expression merci))
(categorie (nom forme-politesse) (expression s il vous plait))

(categorie (nom confirmation) (expression oui) (umms confirmation))
(categorie (nom confirmation) (expression voui) (umms confirmation))
(categorie (nom confirmation) (expression exact) (umms confirmation))
(categorie (nom confirmation) (expression d accord) (umms confirmation))
(categorie (nom confirmation) (expression ok) (umms confirmation))
(categorie (nom confirmation) (expression yes) (umms confirmation))
(categorie (nom confirmation) (expression ouais) (umms confirmation))
(categorie (nom confirmation) (expression mouais) (umms confirmation))
(categorie (nom confirmation) (expression volontiers) (umms confirmation))
(categorie (nom confirmation) (expression okay) (umms confirmation))
(categorie (nom confirmation) (expression certes) (umms confirmation))
(categorie (nom confirmation) (expression pourquoi pas) (umms confirmation))

(categorie (nom infirmation) (expression non) (umms infirmation))
(categorie (nom infirmation) (expression pas d accord) (umms infirmation))

(categorie (nom nom) (expression allissali) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression allys) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression baloul) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression barre) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression baudry) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression beacco) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression benard) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression bonet) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression bourdet) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression bru) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression contamines) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression daubias) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression delannoy) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression deleglise) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression delozanne) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression despres) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression dubourg) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression floccari) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression george) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression gonon) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression gueye) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression jacoboni) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression jean) (umms contacts appele appelant nom) )
(categorie (nom nom) (expression lecordier) (umms contacts appele appelant nom) )
```

Listing: activite-langagiere/conflit-lexeme.lsp

```
;; ##########################################################################################################################
;; ### AMI2 - CONFLIT-LEXEME.LSP - LEMEUNIER 17/01/2000 (Thierry.Lemeunier@lium.univ-lemans.fr) ######################
;; ##########################################################################################################################

;; --------------------------------------------------------------------------------------------------------------------------
;; Les sources de connaissances de gestion des conflits entre lexeme non hypothétique

;; --------------------------------------------------------------------------------------------------------------------------

(defregle-r detection-conflit *dialogueur* 40000

  DOCUMENTATION
    "Détection d'un conflit entre deux lexèmes non hypothétiques (06/10/2000)"

  PRECONDITIONS
    niv-lexemes collect #'disponible -> liste-lexemes
    niv-lexemes #'(lambda (x)
        (and (disponible x)
       (not (hypothetique x))
       (< 0 (pos x))
       (member (cat x) *liste-umms-terminales*))) -> lexeme1
    niv-lexemes #'(lambda (x)
        (and (disponible x)
       (not (hypothetique x))
       (< 0 (pos x))
       (member (cat x) *liste-umms-terminales*))) -> lexeme2

  CONDITION
    (and (not (eq lexeme1 lexeme2))
         (eq (pos lexeme1) (pos lexeme2))
         (not (eq (cat lexeme1) (cat lexeme2)))
   (equalp (expression lexeme1) (expression lexeme2)))

  ACTIONS
    (let ((umm-identifier (ajoute-fils (cree-umm-identifier-lexeme :focus t)
              (cree-umm-source :valeur (expression lexeme1) :statut 'accompli)
              (ajoute-fils (cree-umm-cible)
               ;; Créer les umms terminales selon les catégories des deux lexemes
               ;; Partie ad hoc
               (loop for lexeme in (list lexeme1 lexeme2)
                 collect
                 (case (cat lexeme)
                (nom (cree-umm-nom))
                (prenom (cree-umm-prenom))
                (poste (cree-umm-poste))
                (confirmation (cree-umm-confirmation))
                (infirmation (cree-umm-infirmation))))))))

      (setf (valeur umm-identifier)
      (str2cons (format nil "est-ce que '~A' est ~A"
           (cons2str (expression lexeme1))
           ;; On prend par défaut la première catégorie - Partie ad hoc
           (case (type-of (first (fils (second-fils umm-identifier))))
        (NOM "un nom")
        (PRENOM "un prénom")
        (POSTE "un numéro de poste")
        (CONFIRMATION "une confirmation")
        (INFIRMATION "une infirmation"))))))

    ;; On oublie provisoirement les autres lexemes
    (loop for lexeme in liste-lexemes do
      (setf (statut lexeme) 'en-attente))
    (setf (statut lexeme1) 'en-conflit)
    (setf (statut lexeme2) 'en-conflit)

    (reactiver lexeme1)
    (reactiver lexeme2)
)

;; --------------------------------------------------------------------------------------------------------------------------

(defregle-r gestion-conflit-1 *dialogueur* 39900

  DOCUMENTATION
    "Confirmation de l'hypothèse (06/10/2000)"

  PRECONDITIONS
    niv-umms #'(lambda (x) (and (typep x 'IDENTIFIER-LEXEME) (en-attente x))) -> umm-identifier
    niv-umms #'(lambda (x) (and (typep x 'SOURCE) (accompli x))) -> umm-source
    niv-umms #'(lambda (x) (and (typep x 'CIBLE) (en-attente x))) -> umm-cible
    niv-umms #'(lambda (x) (and (typep x 'CONFIRMATION) (interprete x))) -> umm-conf
    niv-lexemes #'en-conflit -> lexeme1
    niv-lexemes #'en-conflit -> lexeme2
    niv-lexemes collect #'en-attente -> liste-lexemes

  CONDITION
    (and (not (eq lexeme1 lexeme2))
   (eq (pere umm-source) umm-identifier)
   (eq (pere umm-cible) umm-identifier))
```

```
90
      ACTIONS
        (setf (statut umm-conf) 'utilise)

        (applique-statut umm-identifier 'oublie)
95
        (loop for lexeme in liste-lexemes do
          (setf (statut lexeme) 'disponible))

        (if (equal (cat lexeme1) (type-of (premier-fils umm-cible)))
100     (progn
          (setf (statut lexeme2) 'infirme)
          (setf (statut lexeme1) 'disponible))
            (progn
          (setf (statut lexeme1) 'infirme)
105       (setf (statut lexeme2) 'disponible)))
      )

      ;; ------------------------------------------------------------------------------------------------------------------

110   (defregle-r gestion-conflit-2 *dialogueur* 39900

        DOCUMENTATION
          "Infirmation de l'hypothèse (10/10/2000)"

115     PRECONDITIONS
          niv-umms #'(lambda (x) (and (typep x 'IDENTIFIER-LEXEME) (en-attente x))) -> umm-identifier
          niv-umms #'(lambda (x) (and (typep x 'SOURCE) (accompli x))) -> umm-source
          niv-umms #'(lambda (x) (and (typep x 'CIBLE) (en-attente x))) -> umm-cible
          niv-umms #'(lambda (x) (and (typep x 'INFIRMATION) (interprete x))) -> umm-inf
120       niv-lexemes #'en-conflit -> lexeme1
          niv-lexemes #'an-conflit -> lexeme2
          niv-lexemes collect #'en-attente -> liste-lexemes

        CONDITION
125       (and (not (eq lexeme1 lexeme2))
         (eq (pere umm-source) umm-identifier)
         (eq (pere umm-cible) umm-identifier))

        ACTIONS
130       (setf (statut umm-inf) 'utilise)

          (when (= 2 (length (fils umm-cible)))
            (applique-statut umm-identifier 'oublie)
            (loop for lexeme in liste-lexemes do
135       (setf (statut lexeme) 'disponible))
              (if (equal (cat lexeme2) (type-of (premier-fils umm-cible)))
            (progn
              (setf (statut lexeme2) 'disponible)
              (setf (statut lexeme1) 'infirme)
140           (setf (umms lexeme2)
              (cons (type-of (second-fils umm-cible)) (remove (type-of (premier-fils umm-cible)) (umms lexeme2))))
              (setf (cat lexeme2)
              (type-of (second-fils umm-cible))))
          (progn
145         (setf (statut lexeme1) 'disponible)
            (setf (statut lexeme2) 'infirme)
            (setf (umms lexeme1)
            (cons (type-of (second-fils umm-cible)) (remove (type-of (premier-fils umm-cible)) (umms lexeme2))))
            (setf (cat lexeme1)
150         (type-of (second-fils umm-cible))))))

          (enleve-fils umm-cible (premier-fils umm-cible))

          (when (> 1 (length (fils umm-cible)))
155         (setf (valeur umm-identifier)
            (str2cons (format nil "est-ce que '~A' est ~A"
                 (cons2str (expression lexeme1))
                 ;; On prend par défaut la première catégorie - Partie ad hoc
                 (case (type-of (first (fils (second-fils umm-identifier))))
160               (NOM "un nom")
                  (PRENOM "un prénom")
                  (POSTE "un numéro de poste")
                  (CONFIRMATION "une confirmation")
                  (INFIRMATION "une infirmation"))))))
165   )

      ;; ==================================================================================================================
```

Listing: activite-langagiere/gestion-incomprehension.lsp

```
;; ###############################################################################################################################
;; ### AMI2 - GESTION-INCOMPREHENSION.LSP - LEMEUNIER 09/10/2000 (Thierry.Lemeunier@lium.univ-lemans.fr) ###############
;; ###############################################################################################################################

;; -------------------------------------------------------------------------------------------------------------------------------
;; Les sources de connaissances de gestion des incompréhensions sémantiques
;; Elles sont déclenchables après l'interprétation.

;; -------------------------------------------------------------------------------------------------------------------------------
;; Premier cas : il y a un seul acte hypothétique.

;; -------------------------------------------------------------------------------------------------------------------------------

(defregle-r detection-incomprehension-cas1 *dialogueur* 20000

  DOCUMENTATION
    "Détection d'un acte hypothétique (11/10/2000)."

  PRECONDITIONS
    niv-lexemes collect #'disponible -> liste-lexemes
    niv-lexemes #'(lambda (x)
        (and (disponible x) (null (cat x)) (hypothetique x)
          (member (car (umms x)) *liste-umms-actes*))) -> acte-hypo
    niv-lexemes aucun #'(lambda (x) (and (disponible x) (null (cat x)) (not (hypothetique x))))

  ACTIONS
    (let ((umm-identifier
     (ajoute-fils (cree-umm-identifier-lexeme :focus t)
      (cree-umm-source :valeur (expression acte-hypo) :statut 'reussi)
      (ajoute-fils (cree-umm-cible)
            ;; On prend en compte les deux premières umms potentielles
            (ajoute-fils (case (car (umms acte-hypo))
              (REQUETE (cree-umm-requete))
              (PROPOSER (cree-umm-proposer)))
             (case (cadr (umms acte-hypo))
              (PARLER (cree-umm-parler))
              (LIRE-MESSAGE (cree-umm-lire-message))
              (LAISSER-MESSAGE (cree-umm-laisser-message))))))))
      (setf (valeur umm-identifier)
      (str2cons (format nil "vous avez dit \"~A\", est-ce que c'est ~A pour ~A"
           (cons2str (expression acte-hypo))
           (case (type-of (first (fils (second-fils umm-identifier))))
        (REQUETE "une requête")
        (PROPOSER "une proposition"))
           (case (type-of (first (fils (first (fils (second-fils umm-identifier))))))
        (PARLER "contacter un membre du LIUM")
        (LIRE-MESSAGE "lire vos messages")
        (LAISSER-MESSAGE "laisser un message"))))))

    (loop for lexeme in liste-lexemes do (setf (statut lexeme) 'en-attente))
    (setf (statut acte-hypo) 'en-test)
    (reactiver acte-hypo)
)

;; -------------------------------------------------------------------------------------------------------------------------------

(defregle-r gestion-incomprehension-cas1-1 *dialogueur* 20100

  DOCUMENTATION
    "Confirmation de l'acte hypothétique (12/10/2000)."

  PRECONDITIONS
    niv-lexemes collect #'en-attente -> liste-lexemes
    niv-lexemes #'(lambda (x) (and (en-test x) (null (cat x)) (hypothetique x))) -> acte-hypo
    niv-umms #'(lambda (x) (and (typep x 'IDENTIFIER-LEXEME) (en-attente x))) -> umm-identifier
    niv-umms #'(lambda (x) (and (typep x 'CONFIRMATION) (interprete x))) -> umm-conf

  ACTIONS
    (setf (statut umm-conf) 'utilise)
    (applique-statut umm-identifier 'oublie)
    (loop for lexeme in liste-lexemes do (setf (statut lexeme) 'disponible))
    (setf (statut acte-hypo) 'disponible)
    (setf (hypothetique acte-hypo) nil)
)

;; -------------------------------------------------------------------------------------------------------------------------------

(defregle-r gestion-incomprehension-cas1-2 *dialogueur* 20100

  DOCUMENTATION
    "Infirmation de l'acte hypothétique (12/10/2000)."

  PRECONDITIONS
    niv-lexemes collect #'en-attente -> liste-lexemes
    niv-lexemes #'(lambda (x) (and (en-test x) (null (cat x)) (hypothetique x))) -> acte-hypo
    niv-umms #'(lambda (x) (and (typep x 'IDENTIFIER-LEXEME) (en-attente x))) -> umm-identifier
    niv-umms #'(lambda (x) (and (typep x 'INFIRMATION) (interprete x))) -> umm-inf

  ACTIONS
```

```
90      (setf (statut umm-inf) 'utilise)
        (loop for lexeme in liste-lexemes do (setf (statut lexeme) 'disponible))
        (setf (statut acte-hypo) 'infirme)
        (applique-statut umm-identifier 'reussi)
        (reactiver umm-identifier)
95      (setf (valeur umm-identifier) (expression acte-hypo))
    )

    ;; ================================================================================================================================
```

Listing: activite-langagiere/interpreteur.lsp

```
;; ###############################################################################################################
;; ### AMI2 - INTERPRETEUR.LSP - Lemeunier 25/01/99 (Thierry.Lemeunier@lium.univ-lemans.fr) ########################
;; ###############################################################################################################

;; ---------------------------------------------------------------------------------------------------------------
;; Les sources de connaissances de l'interpreteur

(defregle-r appariement-1 *dialogueur* 30000

  DOCUMENTATION
    "Appariement entre umms en attente et umms avec constituants issus de l'analyse"

  PRECONDITIONS
    niv-umms #'en-attente -> umm
    niv-lexemes #'(lambda (x)
        (and (not (hypothetique x))
       (umms x) (constituants x) (not (origine x))
       (or (disponible x) (apparie x)))) -> lexeme

  CONDITION
    (and (member (type-of umm) (umms lexeme))
   (equal (umi umm) (compteur *dialogueur*)))

  ACTIONS
    (when (terminale umm) (setf (valeur umm) (expression lexeme)))
    (setf (statut umm) 'interprete)

    (setf (umms lexeme) (remove (type-of umm) (umms lexeme) :count 1))
    (setf (statut lexeme) 'apparie)
    (setf (origine lexeme) umm)

    (loop for lex in (constituants lexeme) do
      (unless (origine lex)
         (progn
           (setf (statut lex) 'apparie)
           (supprime-non-terminale lex)
           (when (constituants lex) (supprime-terminale lex))))
      (setf (origine lex) umm))
    (when (constituants lexeme) (supprime-terminale lexeme))
)

;; ---------------------------------------------------------------------------------------------------------------

(defregle-r appariement-2 *dialogueur* 29000

  DOCUMENTATION
    "Appariement entre umms en attente et umms avec constituants issus de l'analyse"

  PRECONDITIONS
    niv-umms #'en-attente -> umm
    niv-lexemes #'(lambda (x)
        (and (not (hypothetique x))
       (umms x) (constituants x) (not (origine x))
       (or (disponible x) (apparie x)))) -> lexeme

  CONDITION
    (member (type-of umm) (umms lexeme))

  ACTIONS
    (when (terminale umm) (setf (valeur umm) (expression lexeme)))
    (setf (statut umm) 'interprete)

    (setf (umms lexeme) (remove (type-of umm) (umms lexeme) :count 1))
    (setf (statut lexeme) 'apparie)
    (setf (origine lexeme) umm)

    (loop for lex in (constituants lexeme) do
      (unless (origine lex)
         (progn
           (setf (statut lex) 'apparie)
           (supprime-non-terminale lex)
           (when (constituants lex) (supprime-terminale lex))))
      (setf (origine lex) umm))
    (when (constituants lexeme) (supprime-terminale lexeme))
)


;; ---------------------------------------------------------------------------------------------------------------

(defregle-r appariement-3 *dialogueur* 28000

  DOCUMENTATION
    "Appariement entre umms en attente et umms issus de l'analyse"

  PRECONDITIONS
    niv-umms #'en-attente -> umm
    niv-lexemes #'(lambda (x) (and (not (hypothetique x)) (umms x) (or (disponible x) (apparie x)))) -> lexeme

  CONDITION
```

```
    (and (member (type-of umm) (umms lexeme))
   (if (origine lexeme) (equal (pere umm) (origine lexeme)) t)
   (equal (umi umm) (compteur *dialogueur*)))

  ACTIONS
    (when (terminale umm) (setf (valeur umm) (expression lexeme)))
    (setf (statut umm) 'interprete)

    (setf (umms lexeme) (remove (type-of umm) (umms lexeme) :count 1))
    (setf (statut lexeme) 'apparie)
    (setf (origine lexeme) umm)

    (loop for lex in (constituants lexeme) do
      (unless (origine lex)
         (progn
           (setf (statut lex) 'apparie)
           (supprime-non-terminale lex)
           (when (constituants lex) (supprime-terminale lex))))
      (setf (origine lex) umm))
    (when (constituants lexeme) (supprime-terminale lexeme))
)

;; ------------------------------------------------------------------------------------------------------------------------

(defregle-r appariement-4 *dialogueur* 27000

  DOCUMENTATION
    "Appariement entre umms en attente et umms issus de l'analyse"

  PRECONDITIONS
    niv-umms #'en-attente -> umm
    niv-lexemes #'(lambda (x) (and (not (hypothetique x)) (umms x) (or (disponible x) (apparie x)))) -> lexeme

  CONDITION
    (and (member (type-of umm) (umms lexeme))
   (if (origine lexeme) (equal (pere umm) (origine lexeme)) t))

  ACTIONS
    (when (terminale umm) (setf (valeur umm) (expression lexeme)))
    (setf (statut umm) 'interprete)

    (setf (umms lexeme) (remove (type-of umm) (umms lexeme) :count 1))
    (setf (statut lexeme) 'apparie)
    (setf (origine lexeme) umm)

    (loop for lex in (constituants lexeme) do
      (unless (origine lex)
         (progn
           (setf (statut lex) 'apparie)
           (supprime-non-terminale lex)
           (when (constituants lex) (supprime-terminale lex))))
      (setf (origine lex) umm))
    (when (constituants lexeme) (supprime-terminale lexeme))
)

;; ------------------------------------------------------------------------------------------------------------------------

(defregle-r non-attendus-1 *dialogueur* 26000

  DOCUMENTATION
    "Activité d'interprétation - Prise en compte de umms non-attendus avec constituants"

  PRECONDITIONS
    niv-lexemes #'(lambda (x) (and (not (hypothetique x)) (disponible x) (umms x) (constituants x))) -> lexeme

  ACTIONS
    (make-instance (car (umms lexeme)) :focus t)
)

;; ------------------------------------------------------------------------------------------------------------------------

(defregle-r non-attendus-2 *dialogueur* 25000

  DOCUMENTATION
    "Activité d'interprétation - Prise en compte de umms non-attendus sans constituants"

  PRECONDITIONS
    niv-lexemes #'(lambda (x) (and (not (hypothetique x)) (disponible x) (umms x) (not (constituants x)))) -> lexeme

  ACTIONS
    (make-instance (car (umms lexeme)) :focus t)
)

;; ------------------------------------------------------------------------------------------------------------------------

(defregle-r propagation-validite-umm *dialogueur* 30100

  DOCUMENTATION
    "Propagation de la satisfaction d'une umm (les sous-umms disponibles deviennent inutiles)"
```

```
  PRECONDITIONS
    niv-umms #'accompli -> umm-accompli
    niv-umms #'en-attente -> umm-en-attente

  CONDITION
    (eq (pere umm-en-attente) umm-accompli)

  ACTIONS
    (enleve-fils (pere umm-en-attente) umm-en-attente)
    (setf (statut umm-en-attente) 'inutile)
)

;; ------------------------------------------------------------------------------------------------------------------------

(defregle-r propagation-validite-lexeme *dialogueur* 30100

  DOCUMENTATION
    "Propagation de la satisfaction d'une umm (les lexemes répondant au umm satisfait ne sont plus utiles)"

  PRECONDITIONS
    niv-umms #'accompli -> umm-accompli
    niv-lexemes #'apparie -> lexeme

  CONDITION
    (eq (origine lexeme) umm-accompli)

  ACTIONS
    (setf (statut lexeme) 'utilise)
)

;; ------------------------------------------------------------------------------------------------------------------------
```

Listing: activite-langagiere/generateur.lsp

```
;; ##############################################################################################################################
;; ### AMI2 - GENERATEUR.LSP - Lemeunier 10/12/98 (Thierry.Lemeunier@lium.univ-lemans.fr) ########################################
;; ##############################################################################################################################

;; ------------------------------------------------------------------------------------------------------------------------------
;; Les actes de communications sont de trois types :
;; - questionner (intérêt de 3)
;; - proposer (intérêt de 2)
;; - indiquer (intérêt de 1)

(defclass ACTE (ELEMDIAL)
  ((niveau :allocation :class :reader niveau :initform 'niv-actes)
   (expression :accessor expression :initarg :expression)
   (forme :reader forme :initarg :forme :initform 'positive)
   (type :reader type :initarg :type)
   (lien :accessor lien :initarg :lien :initform nil)))

(defmacro cree-acte (&rest initargs) `(make-instance 'ACTE ,@initargs))

(defmethod description ((oself ACTE)) (format nil "~A sur : ~A" $type $expression))

(defmethod type-actep ((oself ACTE) type-acte) (equal type-acte $type))

(defmethod indiquer ((oself ACTE)) (and $disponible (type-actep oself 'indiquer)))

(defmethod proposer ((oself ACTE)) (and $disponible (type-actep oself 'proposer)))

(defmethod questionner ((oself ACTE)) (and $disponible (type-actep oself 'questionner)))

(defmethod indiquer-negation ((oself ACTE)) (and $indiquer (equal 'negative $forme)))

(defmethod lies-p ((acte1 ACTE) (acte2 ACTE))
  "Retourner vrai si l'acte1 et l'acte2 sont liés sinon retourner faux"
  (cond ((and (consp (lien acte1)) (consp (lien acte2)))
   (intersection (lien acte1) (lien acte2)))
  ((and (consp (lien acte1)) (atom (lien acte2)))
   (member (lien acte2) (lien acte1)))
  ((and (atom (lien acte1)) (consp (lien acte2)))
   (member (lien acte1) (lien acte2)))
  ((and (atom (lien acte1)) (atom (lien acte2)))
   (or (equal (lien acte1) acte2)
       (equal acte1 (lien acte2)))))
)

;; ------------------------------------------------------------------------------------------------------------------------------
;; la génération individuelle bête est méchante sans chercher à prendre en compte les autres actes

(defmethod genere-enonce ((oself ACTE))
  (format nil (if (equal 'negative $forme)
      (negation (car $choisit-texte))
    (car $choisit-texte))
    $construit-sous-enonce))

(defmethod construit-sous-enonce ((oself ACTE))
  (if (cdr $choisit-texte) (cons2str (valeur $expression)) ""))

(defmethod choisit-texte ((oself ACTE))
  (case $type
    (questionner (texte-questionner $expression))
    (proposer (texte-proposer $expression))
    (indiquer (texte-indiquer $expression))))

;; La négation se fait pour l'instant de manière complètement ad hoc (mon dieu quelle horreur !!)

(defun negation (chaine)
  (let ((liste-mots (str2cons chaine)) (continue t))
    (loop while continue do
      (setf continue nil)
       (when (member "peux" liste-mots :test #'equalp)
  (nsubstitute "ne peux pas" "peux" liste-mots :test #'equalp)
  (setf continue t))
      (when (member "pouvez" liste-mots :test #'equalp)
  (nsubstitute "ne pouvez pas" "pouvez" liste-mots :test #'equalp)
  (setf continue t))
      (when (member "voulez" liste-mots :test #'equalp)
  (nsubstitute "ne pouvez pas" "voulez" liste-mots :test #'equalp)
  (setf continue t))
      (when (member "disponible" liste-mots :test #'equalp)
  (nsubstitute "indisponible" "disponible" liste-mots :test #'equalp)
  (setf continue t))
      (when (member "j'ai" liste-mots :test #'equalp)
  (nsubstitute "je n'ai pas" "j'ai" liste-mots :test #'equalp)
  (setf continue t))
      (when (and (member "des" liste-mots :test #'equalp)
     (member "messages" liste-mots :test #'equalp))
  (nsubstitute "de" "des" liste-mots :test #'equalp)
  (nsubstitute "message" "messages" liste-mots :test #'equalp)
  (setf continue t))
      (when (and (member "y" liste-mots :test #'equalp)
```

```
      (member "a" liste-mots :test #'equalp))
    (nsubstitute "n'y" "y" liste-mots :test #'equalp)
    (nsubstitute "a pas" "a" liste-mots :test #'equalp)
    (setf continue t)))
      (cons2str liste-mots)))

;; ------------------------------------------------------------------------------------------------------------------------
;; Les enoncés générés par le système

(defclass ENONCEM (ELEMDIAL)
  ((niveau :allocation :class :reader niveau :initform 'niv-enoncem)
   (expression :accessor expression :initarg :expression)
   (acte :reader acte :initarg :acte)))

(defmacro cree-enoncem (&rest initargs) `(make-instance 'ENONCEM ,@initargs))
(defmethod description ((oself ENONCEM)) (format nil "~A" $expression))

(defmethod type-actep ((oself ENONCEM) type-acte) (equal type-acte $acte))

(defmethod pret ((oself ENONCEM)) (equal $statut 'pret))

(defmethod supprime-blanc ((oself ENONCEM))
  (setf $expression (supprime-blanc-intern $expression)))

(defmethod applique-majuscule ((oself ENONCEM))
  (setf (char $expression 0) (char-upcase (char $expression 0))))

(defmethod applique-virgule ((oself ENONCEM))
  (setf $expression (concatenate 'string $expression ",")))

(defmethod applique-point ((oself ENONCEM))
  (if (not (eq (char $expression (1- (length $expression))) #\?))
    (setf $expression (concatenate 'string $expression "."))))

(defun classement-selon-interet-decroissant (liste-enonces)
  (my-sort liste-enonces #'<= :key #'interet))

;; ------------------------------------------------------------------------------------------------------------------------
;; Les sources de connaissances pour générer les énoncés du système
;; ------------------------------------------------------------------------------------------------------------------------

;; ------------------------------------------------------------------------------------------------------------------------

(defregle-r generation-enoncem *dialogueur* 800

  DOCUMENTATION
    "Génération des énoncés du système"

  PRECONDITIONS
    niv-actes existe #'disponible -> liste-actes

  ACTIONS
    (loop for acte in liste-actes do
      ;; Mise à jour des champs de l'UMM concernée
      (setf (umi (pere-origine (expression acte)))
      (+ 1 (compteur *dialogueur*)))
      (if (not (member (list (type acte) (forme acte)) (anciens-actes (pere-origine (expression acte))) :test #'equal))
    (setf (anciens-actes (pere-origine (expression acte)))
    (cons (list (type acte) (forme acte)) (anciens-actes (pere-origine (expression acte))))))
      ;; Dubli de l'acte
      (setf (statut acte) 'utilise)
      ;; Création de l'énoncé associé à l'acte/intention
      (cree-enoncem :statut 'pret
        :acte (type acte)
        :expression (genere-enonce acte)))
)

;; ------------------------------------------------------------------------------------------------------------------------

(defregle-r mise-en-forme-finale *dialogueur* 700

  DOCUMENTATION
    "Mise en forem finale avant affichage (10/10/2000)."

  PRECONDITIONS
    niv-enoncem existe #'pret -> liste-enoncem

  ACTIONS
    (let ((enonces (classement-selon-interet-decroissant liste-enoncem)))

      (supprime-blanc (first enonces))
      (applique-majuscule (first enonces))
      (setf (statut (first enonces)) 'disponible)

      (loop for enonce in (butlast enonces) do
    (supprime-blanc enonce)
    (applique-virgule enonce)
    (setf (statut enonce) 'disponible))

      (supprime-blanc (first (last enonces)))
```

```
180         (applique-point (first (last enonces)))
            (setf (statut (first (last enonces))) 'disponible))
      )

      ;; ============================================================================================================
```

Listing: activite-dialogique/intention.lsp

```
;; ###############################################################################################################################
;; ### AMI2 - INTENTION.LSP - Lemeunier 07/12/1999 (Thierry.Lemeunier@lium.univ-lemans.fr) ########################################
;; ###############################################################################################################################

;; -------------------------------------------------------------------------------------------------------------------------------
;; les sources de connaissances pour développer les intentions de communication
;; -------------------------------------------------------------------------------------------------------------------------------

(defregle-r copie-umms *dialogueur* 2300

 DOCUMENTATION
  "Copie des umms dans le niveau 'umms-temp' pour travailler"

 PRECONDITIONS
  niv-umms existe #'interessant -> umms-interessants

 ACTIONS
  (loop for umm in (my-sort umms-interessants #'> :key #'interet) do
   (unless (pere umm) (copie-profonde umm)))
)

;; -------------------------------------------------------------------------------------------------------------------------------

(defregle-r regroupement-horizontal *dialogueur* 2200

 DOCUMENTATION
  "Opération de regroupement horizontal - A revoir"

 PRECONDITIONS
  niv-umms-temp #'interessant -> umm-principal

 CONDITION
  (and (> (length (fils umm-principal)) 1)
   (or (> (length (fils-meme-predicat umm-principal :predicat #'en-attente)) 1)
       (> (length (fils-meme-predicat umm-principal :predicat #'reussi)) 1)
       (> (length (fils-meme-predicat umm-principal :predicat #'accompli)) 1)))

 ACTIONS
  (let*
  ((liste-fils
   (cond ((> (length (fils-meme-predicat umm-principal :predicat #'en-attente)) 1)
    (fils-meme-predicat umm-principal :predicat #'en-attente))
   ((> (length (fils-meme-predicat umm-principal :predicat #'reussi)) 1)
    (fils-meme-predicat umm-principal :predicat #'reussi))
   ((> (length (fils-meme-predicat umm-principal :predicat #'accompli)) 1)
    (fils-meme-predicat umm-principal :predicat #'accompli))))

  (umm-composee
   (cree-umm-compose
    :focus (focus (car liste-fils))
    :valeur (loop for un-fils in liste-fils append (valeur un-fils))
    :demandable (demandable (car liste-fils))
    :connu (connu (car liste-fils))
    :expression (loop for un-fils in liste-fils collect (expression un-fils))
    :interet (interet (car liste-fils))
    :statut (statut (car liste-fils))
    :pere-stat (pere-stat (car liste-fils))
    :fils-stat (loop for un-fils in liste-fils append (fils-stat un-fils))
    :pere umm-principal
    :fils (loop for un-fils in liste-fils append (fils un-fils)))))

   (ajoute-fils umm-principal umm-composee)
   (loop for un-fils in liste-fils do
  (setf (statut un-fils) 'obsolete)
  (enleve-fils umm-principal un-fils))

     (loop for un-fils in liste-fils do
       (loop for fils in (fils un-fils)
        do (setf (pere fils) umm-composee))))
)

;; -------------------------------------------------------------------------------------------------------------------------------

(defregle-r regroupement-vertical *dialogueur* 2100

 DOCUMENTATION
  "Opération de regroupement vertical"

 PRECONDITIONS
  niv-umms-temp #'interessant -> umm-principal

 CONDITION
  (and (fils umm-principal)
  (every #'(lambda (x) (equal (statut umm-principal) (statut x))) (fils umm-principal))
  (if (> (length (fils umm-principal)) 1)
      (notany #'fils (fils umm-principal))
   t))

 ACTIONS
```

```
    (let ((umm-compose
           (cree-umm-compose
             :focus (focus umm-principal)
             :valeur (valeur umm-principal)
             :demandable (demandable umm-principal)
             :connu (connu umm-principal)
             :expression (cons umm-principal (fils umm-principal))
             :interet (interet umm-principal)
             :statut (statut umm-principal)
             :pere-stat (pere-stat umm-principal)
             :fils-stat (fils-stat (car (fils umm-principal)))
             :pere (pere umm-principal)
             :fils (loop for un-fils in (fils umm-principal) append (fils un-fils)))))

      (setf (statut umm-principal) 'obsolete)
      (loop for fils in (fils umm-principal) do (setf (statut fils) 'obsolete))

      (when (pere umm-principal)
        (ajoute-fils (pere umm-principal) umm-compose)
        (enleve-fils (pere umm-principal) umm-principal))

      (loop for un-fils in (fils umm-principal) do
        (loop for fils in (fils un-fils)
          do (setf (pere fils) umm-compose))))
)

;; ------------------------------------------------------------------------------------------------------------------------

(defregle-r C1a *dialogueur* 2000

  DOCUMENTATION
    "C1a : une umm est en attente sans pere et sans fils (26/09/2000)"

  PRECONDITIONS
    niv-umms-temp #'en-attente -> umm-principal

  CONDITION
    (and (not (fils umm-principal))
   (not (pere umm-principal))
   (or (disponible-pour-question umm-principal)
       (disponible-pour-proposition umm-principal)))

  ACTIONS
    (let ((type-acte nil) (interet nil))
      (cond ((disponible-pour-question umm-principal)
       (setf type-acte 'questionner interet 3))
      ((disponible-pour-proposition umm-principal)
       (setf type-acte 'proposer interet 2)))
      (when type-acte
        (setf (demandable (pere-origine umm-principal)) nil)
        (setf (demandable umm-principal) nil)
        (cree-acte :interet interet
                   :type type-acte
                   :expression umm-principal)))
)

;; ------------------------------------------------------------------------------------------------------------------------

(defregle-r C1b *dialogueur* 2000

  DOCUMENTATION
    "C1b : une umm est en attente (elle possède des fils d'états quelconques). (29/03/2000)"

  PRECONDITIONS
    niv-umms-temp #'en-attente -> umm-principal

  CONDITION
    (and (> (length (fils umm-principal)) 1)
   (or (disponible-pour-question umm-principal)
       (disponible-pour-proposition umm-principal)))

  ACTIONS
    (let ((type-acte nil) (interet nil))
      (cond ((disponible-pour-question umm-principal)
       (setf type-acte 'questionner interet 3))
      ((disponible-pour-proposition umm-principal)
       (setf type-acte 'proposer interet 2)))
      (when type-acte
        (setf (demandable (pere-origine umm-principal)) nil)
        (setf (demandable umm-principal) nil)
        (cree-acte :interet interet
                   :type type-acte
                   :expression umm-principal)))
)

;; ------------------------------------------------------------------------------------------------------------------------

(defregle-r C2a *dialogueur* 2000

  DOCUMENTATION
```

```
    "C2a : X est réussi avec un seul fils Y en attente. (25/09/2000)"

  PRECONDITIONS
    niv-umms-temp #'reussi -> umm-principal

  CONDITION
    (and (= (length (fils umm-principal)) 1)
   (en-attente (premier-fils umm-principal))
   (or (disponible-pour-question (premier-fils umm-principal))
       (disponible-pour-proposition (premier-fils umm-principal))))

  ACTIONS
    (let ((type-acte nil) (interet nil))
      (cond ((disponible-pour-question (premier-fils umm-principal))
       (setf type-acte 'questionner interet 3))
      ((disponible-pour-proposition (premier-fils umm-principal))
       (setf type-acte 'proposer interet 2)))

      (setf (demandable (premier-fils umm-principal)) nil)
      (setf (demandable (pere-origine (premier-fils umm-principal))) nil)
      (cree-acte :interet interet
     :type type-acte
     :expression (premier-fils umm-principal)))
)

;; ------------------------------------------------------------------------------------------------------------------

(defregle-r C2b *dialogueur* 2000

  DOCUMENTATION
    "C2b : dans une disjonction, la racine réussie possède \
          deux fils : un fils en attente et une fils en réussi.
          (26/09/2000)"

  PRECONDITIONS
    niv-umms-temp #'reussi -> umm-principal

  CONDITION
    (and (= (length (fils umm-principal)) 2)
   (hierarchie-ou umm-principal)
   (or (and (en-attente (premier-fils umm-principal))
      (reussi (second-fils umm-principal)))
       (and (reussi (premier-fils umm-principal))
      (en-attente (second-fils umm-principal)))))

  ACTIONS
    (let ((type-acte nil) (interet nil))
      (cond ((disponible-pour-question (premier-fils umm-principal :predicat #'en-attente))
       (setf type-acte 'questionner interet 3))
      ((disponible-pour-proposition (premier-fils umm-principal :predicat #'en-attente))
       (setf type-acte 'proposer interet 2)))

      (setf (demandable (premier-fils umm-principal :predicat #'en-attente)) nil)
      (setf (demandable (pere-origine (premier-fils umm-principal :predicat #'en-attente))) nil)
      (cree-acte :interet interet
     :type type-acte
     :expression (premier-fils umm-principal :predicat #'en-attente)))
)

;; ------------------------------------------------------------------------------------------------------------------

(defregle-r C2c *dialogueur* 2000

  DOCUMENTATION
    "C2c : dans une conjonction, la racine réussie possède deux fils :\
          un fils en attente et un fils accompli. (26/09/2000)"

  PRECONDITIONS
    niv-umms-temp #'reussi -> umm-principal

  CONDITION
    (and (= (length (fils umm-principal)) 2)
   (hierarchie-et umm-principal)
   (or (and (en-attente (premier-fils umm-principal))
      (accompli (second-fils umm-principal)))
       (and (accompli (premier-fils umm-principal))
      (en-attente (second-fils umm-principal)))))

  ACTIONS
    (let ((type-acte nil) (interet nil))
      (cond ((disponible-pour-question (premier-fils umm-principal :predicat #'en-attente))
       (setf type-acte 'questionner interet 3))
      ((disponible-pour-proposition (premier-fils umm-principal :predicat #'en-attente))
       (setf type-acte 'proposer interet 2)))

      (setf (demandable (premier-fils umm-principal :predicat #'en-attente)) nil)
      (setf (demandable (pere-origine (premier-fils umm-principal :predicat #'en-attente))) nil)
      (cree-acte :interet interet
     :type type-acte
     :expression (premier-fils umm-principal :predicat #'en-attente)))
```

```
)

;; ------------------------------------------------------------------------------------------------------------------------

(defregle-r C3a *dialogueur* 2000

  DOCUMENTATION
    "C3a : X est réussi sans pere ni sans fils. (26/09/2000)"

  PRECONDITIONS
    niv-umms-temp #'reussi -> umm-principal

  CONDITION
    (and (not (pere umm-principal))
   (not (fils umm-principal))
   (disponible-pour-indication-negative umm-principal))

  ACTIONS
    (setf (demandable umm-principal) nil)
    (setf (demandable (pere-origine umm-principal)) nil)
    (cree-acte :interet 1
         :type 'indiquer
         :forme 'negative
         :expression umm-principal)
)

;; ------------------------------------------------------------------------------------------------------------------------

(defregle-r C3b *dialogueur* 2000

  DOCUMENTATION
    "C3b : X est réussi avec un fils Y accompli. (26/09/2000)"

  PRECONDITIONS
    niv-umms-temp #'reussi -> umm-principal

  CONDITION
    (and (= (length (fils umm-principal)) 1)
   (accompli (premier-fils umm-principal))
   (disponible-pour-indication-negative umm-principal))

  ACTIONS
    (setf (demandable umm-principal) nil)
    (setf (demandable (pere-origine umm-principal)) nil)
    (cree-acte :interet 1
         :type 'indiquer
         :forme 'negative
         :expression umm-principal)
)

;; ------------------------------------------------------------------------------------------------------------------------

(defregle-r C3c *dialogueur* 2000

  DOCUMENTATION
    "C3c : dans une disjonction, X est réussi et un fils est accompli.\
          Le nombre des autres fils non-accomplis est > 1. (26/09/2000)"

  PRECONDITIONS
    niv-umms-temp #'reussi -> umm-principal

  CONDITION
    (and (> (length (fils umm-principal)) 1)
   (hierarchie-ou umm-principal)
   (premier-fils umm-principal :predicat #'accompli)
   (disponible-pour-indication-negative umm-principal))

  ACTIONS
    (setf (demandable umm-principal) nil)
    (setf (demandable (pere-origine umm-principal)) nil)
    (cree-acte :interet 1
         :type 'indiquer
         :forme 'negative
         :expression umm-principal)
)

;; ------------------------------------------------------------------------------------------------------------------------

(defregle-r C3d *dialogueur* 2000

  DOCUMENTATION
    "C3d : dans une disjonction, X est réussi et un fils est réussi.\
          Le nombre des autres fils non-réussis est > 1. (26/09/2000)"

  PRECONDITIONS
    niv-umms-temp #'reussi -> umm-principal

  CONDITION
    (and (> (length (fils umm-principal)) 1)
   (hierarchie-et umm-principal)
```

```
    (premier-fils umm-principal :predicat #'reussi)
    (disponible-pour-indication-negative umm-principal)
    (disponible-pour-indication-negative (premier-fils umm-principal :predicat #'reussi)))

  ACTIONS
    (setf (demandable umm-principal) nil)
    (setf (demandable (pere-origine umm-principal)) nil)

    (setf (demandable (premier-fils umm-principal :predicat #'reussi)) nil)
    (setf (demandable (pere-origine (premier-fils umm-principal :predicat #'reussi))) nil)

    (let ((acte1 (cree-acte :interet 1
         :type 'indiquer
         :forme 'negative
         :expression umm-principal))
    (acte2 (cree-acte :interet 1
         :type 'indiquer
         :forme 'negative
         :expression (premier-fils umm-principal :predicat #'reussi))))
      (setf (lien acte1) acte2)
      (setf (lien acte2) acte1))
)

;; ------------------------------------------------------------------------------------------------------------------

(defregle-r C4a *dialogueur* 2000

  DOCUMENTATION
    "C4a : X est accompli sans pere et sans fils (26/09/2000)"

  PRECONDITIONS
    niv-umms-temp #'accompli -> umm-principal

  CONDITION
    (and (not (pere umm-principal))
   (not (fils umm-principal))
   (disponible-pour-indication umm-principal))

  ACTIONS
    (setf (demandable (pere-origine umm-principal)) nil)
    (setf (demandable umm-principal) nil)
    (cree-acte :interet 1
         :type 'indiquer
         :expression umm-principal)
)

;; ------------------------------------------------------------------------------------------------------------------

(defregle-r C4b *dialogueur* 2000

  DOCUMENTATION
    "C4b : dans une disjonction, X est accompli et un fils est accompli.\
          Le nombre des autres fils non-accomplis est > 1. (26/09/2000)"

  PRECONDITIONS
    niv-umms-temp #'accompli -> umm-principal

  CONDITION
    (and (> (length (fils umm-principal)) 1)
   (hierarchie-ou umm-principal)
   (premier-fils umm-principal :predicat #'accompli)
   (disponible-pour-indication umm-principal))

  ACTIONS
    (setf (demandable (pere-origine umm-principal)) nil)
    (setf (demandable umm-principal) nil)
    (cree-acte :interet 1
         :type 'indiquer
         :expression umm-principal)
)

;; ------------------------------------------------------------------------------------------------------------------
;; Opérations de fissions
;; ------------------------------------------------------------------------------------------------------------------

(defregle-r fission-acte-proposer-ou-questionner *dialogueur* 1900

  DOCUMENTATION
    "Opération de fission : il faut faire porter l'intention sur les UMM des UMM composées"

  PRECONDITIONS
    niv-actes #'(lambda (x) (or (proposer x) (questionner x))) -> acte

  CONDITION
    (and (typep (expression acte) 'UMM-COMPOSE)
   (consp (expression (expression acte))))

  ACTIONS
    (loop for un-frere in (expression (expression acte))
```

```
        when (demandable (pere-origine un-frere)) do
        collect (cree-acte :interet (interet acte)
              :type (type acte)
              :expression (pere-origine un-frere)))
    (setf (statut acte) 'fusionne)
    (reactiver acte)
)

;; ------------------------------------------------------------------------------------------------------------------------

(defregle-r fission-acte-indiquer-negation *dialogueur* 1900

 DOCUMENTATION
   "Opération de fission : on indique une négation sur l'umm principale et les sous-umms"

 PRECONDITIONS
   niv-actes #'indiquer-negation -> acte-indiquer

 CONDITION
   (and (fils (expression acte-indiquer))
        (disponible-pour-indication-negative (premier-fils (expression acte-indiquer))))

 ACTIONS
   (setf (lien acte-indiquer)
   (loop for un-fils in (fils (pere-origine (expression acte-indiquer)))
     when (disponible-pour-indication-negative un-fils) do
     (setf (demandable un-fils) nil)
     collect (cree-acte :interet 1
           :type 'indiquer
           :forme 'negative
           :lien acte-indiquer
           :expression un-fils)))
   (setf (demandable (expression acte-indiquer)) nil)
   (setf (expression acte-indiquer) (pere-origine (expression acte-indiquer)))
   (reactiver acte-indiquer)
)

;; ------------------------------------------------------------------------------------------------------------------------

(defregle-r fission-acte-indiquer *dialogueur* 1900

 DOCUMENTATION
   "Opération de fission : on informe que sur l'umm la plus importante"

 PRECONDITIONS
   niv-actes #'indiquer -> acte-indiquer

 CONDITION
   (typep (expression acte-indiquer) 'UMM-COMPOSE)

 ACTIONS
   (setf (demandable (expression acte-indiquer)) nil)
   (setf (expression acte-indiquer) (pere-origine (expression acte-indiquer)))
   (reactiver acte-indiquer)
)

;; ------------------------------------------------------------------------------------------------------------------------
;; Règles de filtrage
;; ------------------------------------------------------------------------------------------------------------------------

(defregle-r regle-anti-repetition *dialogueur* 1850

  DOCUMENTATION
    "La répétition n'a lieu que si c'est la seule possibilité. (06/10/2000)"

  PRECONDITIONS
    niv-actes #'disponible -> acte1
    niv-actes #'disponible -> acte2

  CONDITION
    (and (not (equal acte1 acte2))
   (not (appartient-meme-arbre-p (expression acte1) (expression acte2)))
   (equal (type acte1) (type acte2))
   (equal (forme acte1) (forme acte2))
   (member (list (type acte1) (forme acte1)) (anciens-actes (pere-origine (expression acte1))) :test #'equal)
   (not (member (list (type acte2) (forme acte2)) (anciens-actes (pere-origine (expression acte2))) :test #'equal)))

  ACTIONS
    (decf (interet acte1))
)

;; ------------------------------------------------------------------------------------------------------------------------

(defregle-r regle-filtrage-1 *dialogueur* 1800

  DOCUMENTATION
    "Nous limitons à 1 le nombre d'intentions portant sur les UMM d'une arborescence (28/09/2000)."

  PRECONDITIONS
```

```
    niv-actes #'disponible -> acte1
    niv-actes #'disponible -> acte2

  CONDITION
    (and (not (equal acte1 acte2))
   (appartient-meme-arbre-p (expression acte1) (expression acte2))
   (not (lies-p acte1 acte2))
   (>= (interet acte1) (interet acte2)))

  ACTIONS
    (setf (demandable (pere-origine (expression acte2))) t)
    (setf (statut acte1) 'filtre2)
)

;; ------------------------------------------------------------------------------------------------------------------------

(defregle-r regle-filtrage-2 *dialogueur* 1800

  DOCUMENTATION
    "Nous limitons à 1 le nombre de type d'intentions portant sur l'ensemble\
     des UMM des arborescences actives (28/09/2000)."

  PRECONDITIONS
    niv-actes #'disponible -> acte1
    niv-actes #'disponible -> acte2

  CONDITION
    (and (not (equal acte1 acte2))
   (not (appartient-meme-arbre-p (expression acte1) (expression acte2)))
   (equal (type acte1) (type acte2)))

  ACTIONS
    (setf (demandable (pere-origine (expression acte2))) t)
    (setf (statut acte2) 'filtre2)
)

;; ------------------------------------------------------------------------------------------------------------------------

(defregle-r regle-filtrage-3 *dialogueur* 1800

  DOCUMENTATION
    "Le contenu intentionnel des questions et des propositions ne peut pas \
     être composé de plus de deux UMM (28/09/2000)."

  PRECONDITIONS
    niv-actes #'(lambda (x) (or (proposer x) (questionner x))) -> acte

  CONDITION
    (and (typep (expression acte) 'UMM-COMPOSE)
   (consp (expression (expression acte)))
   (> (length (expression (expression acte))) 2))

  ACTIONS
    (setf (expression (expression acte))
    (list (car (expression (expression acte)))
    (cadr (expression (expression acte)))))
)

;; ------------------------------------------------------------------------------------------------------------------------
;; Fin de la génération des intentions
;; Nettoyage des UMM temporaires

(defregle-r supprime-umm-compose *dialogueur* 1000

 DOCUMENTATION
   "Destruction de toutes les umms composées"

 PRECONDITIONS
   niv-umms-temp collect #'courant -> umms-temp

 ACTIONS
   (loop for umm in umms-temp do (setf (statut umm) 'obsolete))
)

;; ------------------------------------------------------------------------------------------------------------------------
```

Listing: activite-applicative/umms-appli.lsp

```
; ##############################################################################################################################
; ### AMI2 - UMMS-APPLI.LSP - LEMEUNIER (version du 24/02/1999) Thierry.Lemeunier@lium.univ-lemans.fr ##################
; ##############################################################################################################################

;; ----------------------------------------------------------------------------------------------------------------------------
;; Définition des objets du domaine de l'application AMI

;; Liste des umms (pour faciliter la programmation)

(defparameter *liste-umms-terminales* '(message poste nom prenom infirmation confirmation))
(defparameter *liste-umms-actes* '(requete proposer))

;; Déclaration de la classe REQUETE (force illocutoire)

(defclass REQUETE (UMM)
 ((interet :accessor interet :initarg :interet :initform 12)
   (connu :accessor connu :initarg :connu :initform t)
   (texte-questionner :reader texte-questionner
         :initform '("quelle est votre requête ?"))
   (texte-proposer :reader texte-proposer
       :initform '("vous pouvez soit lire vos messages soit laisser des messages"))
   (texte-indiquer :reader texte-indiquer
       :initform '("j'ai bien pris en compte votre requête"))
   (fils-stat :reader fils-stat :initform '(ou parler lire-message laisser-message))))

(defmacro cree-umm-requete (&rest -args) `(make-instance 'REQUETE ,@-args))

;; Déclaration de la classe PARLER (l'appelant veut parler à un appelé)

(defclass PARLER (UMM)
  ((interet :accessor interet :initarg :interet :initform 11)
   (connu :accessor connu :initarg :connu :initform t)
   (texte-questionner :reader texte-questionner
         :initform '("voulez-vous parler à un membre du laboratoire ?"))
   (texte-proposer :reader texte-proposer
       :initform '("je vous propose de contacter un membre du laboratoire"))
   (texte-indiquer :reader texte-indiquer
       :initform '("vous voulez parler à un membre du laboratoire"))
   (pere-stat :reader pere-stat :initform '(et requete))
   (fils-stat :reader fils-stat :initform '(et contacts appelant))))

(defmacro cree-umm-parler (&rest -args) `(make-instance 'PARLER ,@-args))

;; Déclaration de la classe LIRE-MESSAGE (l'appelant veut lire ses messages)

(defclass LIRE-MESSAGE (UMM)
  ((interet :accessor interet :initarg :interet :initform 10)
   (connu :accessor connu :initarg :connu :initform t)
   (texte-questionner :reader texte-questionner
         :initform '("voulez-vous lire vos messages ?"))
   (texte-proposer :reader texte-proposer :initform '("je vous propose de lire vos messages"))
   (texte-indiquer :reader texte-indiquer :initform '("il y a des messages en attente pour vous"))
   (pere-stat :reader pere-stat :initform '(ou requete proposer))
   (fils-stat :reader fils-stat :initform '(et appelant))))

(defmacro cree-umm-lire-message (&rest -args) `(make-instance 'LIRE-MESSAGE ,@-args))

;; Déclaration de la classe LAISSER-MESSAGE (l'appelant veut laisser un message pour l'appelé)

(defclass LAISSER-MESSAGE (UMM)
  ((interet :accessor interet :initarg :interet :initform 9)
   (connu :accessor connu :initarg :connu :initform t)
   (texte-questionner :reader texte-questionner :initform '("voulez-vous laisser un message ?"))
   (texte-proposer :reader texte-proposer :initform '("je vous propose de lui laisser un message"))
   (texte-indiquer :reader texte-indiquer :initform '("~A" t))
   (pere-stat :reader pere-stat :initform '(ou requete proposer))
   (fils-stat :reader fils-stat :initform '(et contacts appelant message))))

(defmacro cree-umm-laisser-message (&rest -args) `(make-instance 'LAISSER-MESSAGE ,@-args))

(defmethod donne-message ((oself LAISSER-MESSAGE))
  (let ((appelant (cons2str (valeur (second $fils))))
        (date (quand (car (last $fils))))
        (valeur (valeur (car (last $fils)))))
    (str2cons (format nil "~A vous a laissé le message suivant le ~A :~T~T~A" appelant date valeur))))

;; Déclaration de la classe IDENTIFIER-APPELE (le système veut identifier l'appelé)

(defclass IDENTIFIER-APPELE (UMM)
  ((interet :accessor interet :initarg :interet :initform 8)
   (texte-proposer :reader texte-proposer
       :initform '("je vous propose d'identifier la personne que vous voulez contacter"))
   (texte-indiquer :reader texte-indiquer
       :initform '("nous allons identifier la personne que vous voulez contacter"))
   (pere-stat :reader pere-stat :initform '(et proposer))
   (fils-stat :reader fils-stat :initform '(ou appele appele appele appele appele))))

(defmacro cree-umm-identifier-appele (&rest -args) `(make-instance 'IDENTIFIER-APPELE ,@-args))
```

```
;; Déclaration de la classe IDENTIFIER-APPELANT (le système veut identifier l'appelant)

(defclass IDENTIFIER-APPELANT (UMM)
  ((interet :accessor interet :initarg :interet :initform 7)
   (texte-proposer :reader texte-proposer
       :initform '("aidez-moi à vous identifier"))
   (texte-indiquer :reader texte-indiquer
       :initform '("j'ai réussi à vous identifier"))
   (pere-stat :reader pere-stat :initform '(et proposer))
   (fils-stat :reader fils-stat :initform '(ou appelant appelant appelant appelant appelant))))

(defmacro cree-umm-identifier-appelant (&rest -args) `(make-instance 'IDENTIFIER-APPELANT ,@-args))

;; Déclaration de la classe CONTACTS (les personnes à contacter)

(defclass CONTACTS (UMM)
  ((interet :accessor interet :initarg :interet :initform 6)
   (connu :accessor connu :initarg :connu :initform t)
   (texte-questionner :reader texte-questionner
        :initform '("pouvez-vous m'indiquer l'identité de la personne que vous voulez contacter ?"))
   (texte-proposer :reader texte-proposer
       :initform '("j'ai plusieurs propositions"))
   (texte-indiquer :reader texte-indiquer
       :initform '("vous pouvez contacter cette personne car elle est disponible"))
   (pere-stat :reader pere-stat :initform '(ou parler laisser-message))
   (fils-stat :reader fils-stat :initform '(ou appele appele appele appele appele))))

(defmacro cree-umm-contacts (&rest -args) `(make-instance 'CONTACTS ,@-args))

;; Déclaration de la classe APPELE (la personne à contacter)

(defclass APPELE (UMM)
  ((interet :accessor interet :initarg :interet :initform 3)
   (connu :accessor connu :initarg :connu :initform t)
   (texte-questionner :reader texte-questionner
        :initform '("pouvez-vous m'indiquer l'identité de la personne que vous voulez contacter"))
   (texte-proposer :reader texte-proposer
       :initform '("je vous propose de contacter ~A" t))
   (texte-indiquer :reader texte-indiquer
       :initform '("je connais la personne que vous voulez contacter"))
   (pere-stat :reader pere-stat :initform '(et contacts identifier-appele))
   (fils-stat :reader fils-stat :initform '(ou poste nom prenom))))

(defmacro cree-umm-appele (&rest -args) `(make-instance 'APPELE ,@-args))

;; Déclaration de la classe APPELANT (la personne qui appelle)

(defclass APPELANT (UMM)
  ((interet :accessor interet :initarg :interet :initform 5)
   (connu :accessor connu :initarg :connu :initform t)
   (texte-questionner :reader texte-questionner :initform '("quelle est votre identité ?"))
   (texte-proposer :reader texte-proposer :initform '("êtes-vous ~A ?" t))
   (texte-indiquer :reader texte-indiquer :initform '("je sais qui vous êtes"))
   (pere-stat :reader pere-stat :initform '(ou parler lire-message laisser-message identifier-appelant))
   (fils-stat :reader fils-stat :initform '(ou nom prenom))))

(defmacro cree-umm-appelant (&rest initargs) `(make-instance 'APPELANT ,@initargs))

;; Déclaration de la classe MESSAGE (un message à transmettre)

(defclass MESSAGE (UMM)
  ((interet :accessor interet :initarg :interet :initform 4)
   (texte-questionner :reader texte-questionner
        :initform '("allez-si, donnez-moi votre message"))
   (texte-proposer :reader texte-proposer
       :initform '("je propose que vous me donniez votre message"))
   (texte-indiquer :reader texte-indiquer
       :initform '("le message est : ~A" t))
   (quand :accessor quand :initarg :quand :initform (concatenate 'string (aujourdhui) " à " (heure)))
   (pere-stat :reader pere-stat :initform '(et laisser-message))))

(defmacro cree-umm-message (&rest -args) `(make-instance 'MESSAGE ,@-args))

(defmethod initialise-umm ((new-umm MESSAGE) (old-umm MESSAGE))
  "Initialisation spécifique pour les messages"
  (setf (quand new-umm) (quand old-umm))
  (call-next-method))

(defclass POSTE (UMM)
  ((interet :accessor interet :initarg :interet :initform 2)
   (connu :accessor connu :initarg :connu :initform t)
   (texte-questionner :reader texte-questionner
        :initform '("pouvez-vous me donner le numéro de poste"))
   (texte-proposer :reader texte-proposer
       :initform '("je vous propose de me donner le numéro de poste"))
   (texte-indiquer :reader texte-indiquer
       :initform '("le numéro de poste est : ~A" t))
   (pere-stat :reader pere-stat :initform '(et appele))))

(defmacro cree-umm-poste (&rest -args) `(make-instance 'POSTE ,@-args))
```

```
180
      (defclass NOM (UMM)
        ((interet :accessor interet :initarg :interet :initform 1)
         (connu :accessor connu :initarg :connu :initform t)
         (texte-questionner :reader texte-questionner
185            :initform '("pouvez-vous m'indiquer un nom ?"))
         (texte-proposer :reader texte-proposer
             :initform '("je propose que vous me donniez un nom"))
         (texte-indiquer :reader texte-indiquer
             :initform '("le nom est : ~A" t))
190      (pere-stat :reader pere-stat :initform '(ou appele appelant))))

      (defmacro cree-umm-nom (&rest -args) `(make-instance 'NOM ,@-args))

      (defclass PRENOM (UMM)
195     ((interet :accessor interet :initarg :interet :initform 0)
         (connu :accessor connu :initarg :connu :initform t)
         (texte-questionner :reader texte-questionner
               :initform '("pouvez-vous m'indiquer un prénom ?"))
         (texte-proposer :reader texte-proposer
200          :initform '("je propose que vous me donniez un prénom"))
         (texte-indiquer :reader texte-indiquer
             :initform '("le prénom est : ~A" t))
         (pere-stat :reader pere-stat :initform '(ou appele appelant))))

205   (defmacro cree-umm-prenom (&rest -args) `(make-instance 'PRENOM ,@-args))

      ;; ------------------------------------------------------------------------------------------------------------------------
      ;; Déclaration des fonctions de copie

210   (declare-copie-umm
        REQUETE PARLER LIRE-MESSAGE LAISSER-MESSAGE IDENTIFIER-APPELE
        IDENTIFIER-APPELANT CONTACTS APPELE APPELANT MESSAGE NOM PRENOM POSTE)
```

Listing: activite-applicative/tache0.lsp

```
; ##########################################################################################################################
; ### AMI2 - TACHE0.LSP (version du 18/01/2000) ############################################################################
; ##########################################################################################################################

(load "activite-applicative/personne.lsp") ;; Chargement de la classe ITEM-PERSONNE
(load "activite-applicative/annuaire.lsp") ;; et l'annuaire des personnes du labo

; ##########################################################################################################################
; ## LES SOURCES DE CONNAISSANCES DE LA TACHE APPLICATIVE ###################################################################
; ##########################################################################################################################

; --------------------------------------------------------------------------------------------------------------------------

(defregle-r tache0-debut-tache *dialogueur* 10000

  DOCUMENTATION
    "Activité applicative - Instanciation de l'umm racine de la hiérarchie applicative"

  PRECONDITIONS
    niv-etats #'debut-tache

  ACTIONS
    (setf (interet (unite 'tache1-laisser-message-appele)) 7000) ;; Modification de l'intérêt
    (setf (interet (unite 'tache2-aucun-message)) 6800)

    (cree-umm-requete :focus t)
)

; --------------------------------------------------------------------------------------------------------------------------

(defregle-r activation-automatique *dialogueur* 4000

  DOCUMENTATION
    "Règle de réactivation automatique des umms pour forcer la création d'événements"

  PRECONDITIONS
    niv-umms collect #'vrai -> tous-les-umms

  ACTIONS
    (loop for umm in tous-les-umms do (reactiver umm))
)

; --------------------------------------------------------------------------------------------------------------------------

(defregle-r tache0-explicite-requete *dialogueur* 10000

  DOCUMENTATION
    "Activité applicative - Identification d'une requête"

  PRECONDITIONS
    niv-umms #'(lambda (x) (and (typep x 'REQUETE) (interprete x))) -> umm-requete

  ACTIONS
    (setf (statut umm-requete) 'reussi)
    (ajoute-fils umm-requete
      (cree-umm-parler)
      (cree-umm-lire-message)
      (cree-umm-laisser-message)
      (cree-umm-appelant))
)

; --------------------------------------------------------------------------------------------------------------------------

(defregle-r tache0-tache1 *dialogueur* 10000

  DOCUMENTATION
    "Activité applicative - Identification de la première tâche"

  PRECONDITIONS
    niv-umms #'(lambda (x) (and (typep x 'REQUETE) (reussi x))) -> umm-requete
    niv-umms #'(lambda (x) (and (typep x 'PARLER) (interprete x))) -> umm-parler
    niv-umms existe #'en-attente -> les-umms-en-attente

  CONDITION
    (equal (pere umm-parler) umm-requete)

  ACTIONS
    (cree-etat :expression 'tache1)
    (mapcar #'(lambda (umm) (setf (statut umm) 'inutile)) les-umms-en-attente)

    (setf (fils umm-requete) (list umm-parler))
    (setf (valeur umm-requete) umm-parler)

    (setf (statut umm-parler) 'reussi)
    (ajoute-fils umm-parler
      (cree-umm-contacts)
      (cree-umm-appelant))
)
```

```
; ------------------------------------------------------------------------------------------------------------------------

(defregle-r tache0-tache2 *dialogueur* 10000

  DOCUMENTATION
    "Activité applicative - Identification de la seconde tâche"

  PRECONDITIONS
    niv-umms #'(lambda (x) (and (typep x 'REQUETE) (reussi x))) -> umm-requete
    niv-umms #'(lambda (x) (and (typep x 'LIRE-MESSAGE) (interprete x))) -> umm-lire-message
    niv-umms existe #'en-attente -> les-umms-en-attente

  CONDITION

    (equal (pere umm-lire-message) umm-requete)

  ACTIONS
    (cree-etat :expression 'tache2)
    (mapcar #'(lambda (umm) (setf (statut umm) 'inutile)) les-umms-en-attente)

    (setf (fils umm-requete) (list umm-lire-message))
    (setf (valeur umm-requete) umm-lire-message)

    (setf (statut umm-lire-message) 'reussi)
    (ajoute-fils umm-lire-message (cree-umm-appelant))
)

; ------------------------------------------------------------------------------------------------------------------------

(defregle-r tache0-tache3 *dialogueur* 10000

  DOCUMENTATION
    "Activité applicative - Identification de la troisième tâche"

  PRECONDITIONS
    niv-umms #'(lambda (x) (and (typep x 'REQUETE) (reussi x))) -> umm-requete
    niv-umms #'(lambda (x) (and (typep x 'LAISSER-MESSAGE) (interprete x))) -> umm-laisser-message
    niv-umms existe #'en-attente -> les-umms-en-attente

  CONDITION
    (equal (pere umm-laisser-message) umm-requete)

  ACTIONS
    (cree-etat :expression 'tache3)
    (mapcar #'(lambda (umm) (setf (statut umm) 'inutile)) les-umms-en-attente)

    (setf (fils umm-requete) (list umm-laisser-message))
    (setf (valeur umm-requete) umm-laisser-message)

    (setf (statut umm-laisser-message) 'reussi)
    (ajoute-fils umm-laisser-message
      (cree-umm-contacts)
      (cree-umm-appelant)
      (cree-umm-message :valeur t))
)

; ------------------------------------------------------------------------------------------------------------------------

(defregle-r tache0-cree-contacts *dialogueur* 9990

  DOCUMENTATION
    "Activité applicative - Création du groupe de contacts"

  PRECONDITIONS
    niv-umms #'(lambda (x) (and (typep x 'CONTACTS) (interprete x))) -> umm-contacts

  ACTIONS
    (setf (statut umm-contacts) 'reussi)
    (ajoute-fils umm-contacts
      (cree-umm-appele)
      (cree-umm-appele))
)

; ------------------------------------------------------------------------------------------------------------------------

(defregle-r tache0-nettoyage-resultat *dialogueur* 9950

  DOCUMENTATION
    "Activité applicative - Suppression des résultats pouvant gêner (18/01/2000)"

  PRECONDITIONS
    niv-resu #'(lambda (x) (accompli (origine x))) -> un-resultat
    niv-resu collect #'en-test -> les-autres-resultats

  ACTIONS
    (setf (statut un-resultat) 'utilise)
    (loop for un-resu in les-autres-resultats
      when (eq (origine un-resultat) (origine un-resu))
      do (setf (statut un-resu) 'utilise))
)
```

```
180
    ; ------------------------------------------------------------------------------------------------------------------------

    (defregle-r tache0-recuperation-appele-1 *dialogueur* 9900

185   DOCUMENTATION
        "Activité applicative - Récupération de l'identité de l'appelé"

      PRECONDITIONS
        niv-umms #'(lambda (x) (and (typep x 'APPELE) (or (en-attente-sf x) (reussi-sf x)))) -> umm-appele
190     niv-umms #'(lambda (x) (and (typep x 'APPELE) (accompli x))) -> umm-appele-accompli

      CONDITION
        (and (not (eq (pere umm-appele) (pere umm-appele-accompli))) ;; Il ne faut pas qu'ils aient le même père
       (not (valeur umm-appele))                      ;; Il ne faut pas que le umm soit valué
195    (or (typep (racine umm-appele) 'PROPOSER)      ;; La récupération se fait pour les propositions
           (typep (racine umm-appele-accompli) 'PROPOSER)))

      ACTIONS
        (applique-statut umm-appele 'inutile)
200     (setf (fils umm-appele) nil)

        (setf (valeur umm-appele) (valeur umm-appele-accompli))
        (setf (statut umm-appele) 'accompli)

205     (unless (pere umm-appele-accompli) (applique-statut umm-appele-accompli 'inutile))
    )

    ; ------------------------------------------------------------------------------------------------------------------------

210 (defregle-r tache0-recuperation-appele-2 *dialogueur* 9900

      DOCUMENTATION
        "Activité applicative - Récupération de l'identité de l'appelé"

215   PRECONDITIONS
        niv-umms #'(lambda (x) (and (typep x 'APPELE) (or (en-attente x) (reussi x)))) -> umm-appele
        niv-umms #'(lambda (x) (and (typep x 'APPELE) (accompli-sf x))) -> umm-appele-accompli

      CONDITION
220     (and (not (eq (pere umm-appele) (pere umm-appele-accompli))) ;; Il ne faut pas qu'ils aient le même père
       (not (valeur umm-appele))                      ;; Il ne faut pas que le umm soit valué
       (or (typep (racine umm-appele) 'PROPOSER)      ;; La récupération se fait pour les propositions
           (typep (racine umm-appele-accompli) 'PROPOSER)))

225   ACTIONS
        (applique-statut umm-appele 'inutile)
        (setf (fils umm-appele) nil)

        (setf (valeur umm-appele) (valeur umm-appele-accompli))
230     (setf (statut umm-appele) 'accompli)

        (unless (pere umm-appele-accompli) (applique-statut umm-appele-accompli 'inutile))
    )

235 ; ------------------------------------------------------------------------------------------------------------------------

    (defregle-r tache0-explicite-ident-appele *dialogueur* 9900

      DOCUMENTATION
240     "Activité applicative - Identification explicite de la personne appelé"

      PRECONDITIONS
        niv-umms #'(lambda (x) (and (typep x 'APPELE) (interprete x))) -> umm-appele

245   ACTIONS
        (setf (statut umm-appele) 'reussi)
        (ajoute-fils umm-appele
          (cree-umm-poste)
          (cree-umm-nom)
250       (cree-umm-prenom))
    )

    ; ------------------------------------------------------------------------------------------------------------------------

255 (defregle-r tache0-supprime-un-contact *dialogueur* 9850

      DOCUMENTATION
        "Activité applicative - Suppression d'un appelé de la liste des contacts"

260   PRECONDITIONS
        niv-umms #'(lambda (x) (and (typep x 'CONTACTS) (reussi x))) -> umm-contacts
        niv-umms aucun #'(lambda (x) (and (typep x 'PROPOSER) (reussi x)))

      ACTIONS
265     (loop for appele in (fils umm-contacts)
          when (en-attente appele)
          do (enleve-fils umm-contacts appele)
             (setf (statut appele) 'obsolete))
    )
```

```
270
    ; ------------------------------------------------------------------------------------------------------------------------

    (defregle-r tache0-recuperation-appelant *dialogueur* 9900

275   DOCUMENTATION
        "Activité applicative - Récupération de l'identité de l'appelant"

      PRECONDITIONS
        niv-umms #'(lambda (x) (and (typep x 'APPELANT) (or (reussi x) (en-attente x)))) -> umm-appelant
280     niv-umms #'(lambda (x) (and (typep x 'APPELANT) (accompli-sf x))) -> umm-appelant-accompli

      ACTIONS
        (applique-statut umm-appelant 'inutile)
        (setf (fils umm-appelant) nil)
285
        (setf (valeur umm-appelant) (valeur umm-appelant-accompli))
        (setf (statut umm-appelant) 'accompli)

        (unless (pere umm-appelant-accompli) (applique-statut umm-appelant-accompli 'inutile))
290 )

    ; ------------------------------------------------------------------------------------------------------------------------

    (defregle-r tache0-explicite-ident-appelant *dialogueur* 9900
295
      DOCUMENTATION
        "Activité applicative - Identification explicite de la personne appelante"

      PRECONDITIONS
300     niv-umms #'(lambda (x) (and (typep x 'APPELANT) (interprete x))) -> umm-appelant

      ACTIONS
        (setf (statut umm-appelant) 'reussi)
        (ajoute-fils umm-appelant
305       (cree-umm-nom)
          (cree-umm-prenom))
    )

    ; ------------------------------------------------------------------------------------------------------------------------
310
    (defregle-r tache0-recherche-dans-annuaire *dialogueur* 7000

      DOCUMENTATION
        "Activité applicative - Recherche dans l'annuaire interne du laboratoire (17/01/2000)"
315
      PRECONDITIONS
        niv-umms existe #'(lambda (x) (and (interprete x) (member (type-of x) *liste-umms-terminales*))) -> umms-interpretes

      ACTIONS
320     (loop for partition in (partitionne-liste umms-interpretes :key #'pere) do
          (loop for personne in (rechercher-personnes
               (loop for umm in partition
                 when (setf (statut umm) 'en-test)
                 collect (case (type-of umm)
325              (NOM (cons 'NOM (valeur umm)))
                 (PRENOM (cons 'PRENOM (valeur umm)))
                 (POSTE (cons 'POSTE (valeur umm))))))) do
      (cree-resultat :valeur (list (lenom personne) (leprenom personne))
              :origine (pere (car partition)))))
330 )

    ; ------------------------------------------------------------------------------------------------------------------------

    (defregle-r tache0-un-seul-resultat *dialogueur* 9800
335
      DOCUMENTATION
        "Activité applicative - Il y un seul résultat aprés la recherche dans l'annuaire"

      PRECONDITIONS
340     niv-resu #'disponible -> un-resultat
        niv-resu collect #'disponible -> les-resultats
        niv-umms #'(lambda (x) (and (typep x 'REQUETE) (or (en-attente-sf x) (reussi-sf x))))
        niv-umms existe #'en-test -> umms-en-test

345   CONDITION
        (not (my-find-if
        #'(lambda (x) (eq (origine x) (origine un-resultat)))
        (remove un-resultat les-resultats)))

350   ACTIONS
        (loop for umm in umms-en-test
          when (equal (pere umm) (origine un-resultat))
          do (setf (statut umm) 'accompli))
        (setf (statut un-resultat) 'accepte)
355     (setf (valeur (origine un-resultat)) (valeur un-resultat))
        (setf (statut (origine un-resultat)) 'accompli)
    )

    ; ------------------------------------------------------------------------------------------------------------------------
```

```
360
(defregle-r tache0-plusieurs-resultats *dialogueur* 9700

  DOCUMENTATION
    "Activité applicative - Il y a plusieurs résultats après la recherche dans l'annuaire"
365
  PRECONDITIONS
    niv-resu existe #'disponible -> les-resultats
    niv-umms #'(lambda (x) (and (typep x 'REQUETE) (or (en-attente x) (reussi x)))) -> umm-requete

370 CONDITION
    (> (length les-resultats) 1)

  ACTIONS
    (enleve-focus umm-requete)
375
    (loop for partition in (partitionne-liste les-resultats :key #'origine) do
      (setf umm-proposer (cree-umm-proposer :statut 'reussi :focus t))

      when (typep (origine (car partition)) 'APPELE) do
380   (let ((umm-identifier-appele (cree-umm-identifier-appele :statut 'reussi)))
    (ajoute-fils umm-proposer umm-identifier-appele)
    (loop for resultat in partition do
      (setf (statut resultat) 'en-test)
      (ajoute-fils umm-identifier-appele (cree-umm-appele :connu nil :valeur (valeur resultat)))))
385
      when (typep (origine (car partition)) 'APPELANT) do
      (let ((umm-identifier-appelant (cree-umm-identifier-appelant :statut 'reussi)))
    (ajoute-fils umm-proposer umm-identifier-appelant)
    (loop for resultat in partition do
390   (setf (statut resultat) 'en-test)
      (ajoute-fils umm-identifier-appelant (cree-umm-appelant :connu nil :valeur (valeur resultat)))))

      (setf (valeur umm-proposer) (car (fils umm-proposer))))
  )
395
  ; ------------------------------------------------------------------------------------------------------------------------

  (defregle-r tache0-fin-proposer *dialogueur* 9600

400 DOCUMENTATION
    "Activité applicative - Une proposition a été acceptée"

  PRECONDITIONS
    niv-umms existe #'(lambda (x) (and (typep x 'PROPOSER)
405               (reussi x)
                  (or (typep (car (fils x)) 'IDENTIFIER-APPELE)
                (typep (car (fils x)) 'IDENTIFIER-APPELANT)))) -> umms-proposer

    niv-umms #'(lambda (x) (and (typep x 'REQUETE) (or (en-attente-sf x) (reussi-sf x)))) -> umm-requete
410
  CONDITION
    (every #'(lambda (umm-proposer) (some #'accompli (fils umm-proposer))) umms-proposer)

  ACTIONS
415 (applique-statut umm-proposer 'oublie)
    (enleve-focus umm-proposer)
    (devient-focus umm-requete)
  )

420 ; ------------------------------------------------------------------------------------------------------------------------

  (defregle-r tache0-echec-ident-appele *dialogueur* 9200

    DOCUMENTATION
425   "Activité applicative - Identification de la personne appele non faite"

    PRECONDITIONS
      niv-umms #'(lambda (x) (and (typep x 'APPELE) (reussi x))) -> umm-appele
      niv-resu aucun #'(lambda (x) (or (disponible x) (en-test x)))
430   niv-umms existe #'en-test -> umms-en-test
      niv-umms collect #'en-attente -> umms-en-attentes

    ACTIONS
      (loop for umm in umms-en-test do (setf (statut umm) 'reussi))
435   (loop for umm in umms-en-attentes do (setf (statut umm) 'inutile))
  )

  ; ------------------------------------------------------------------------------------------------------------------------

440 (defregle-r tache0-echec-ident-appelant *dialogueur* 9100

    DOCUMENTATION
      "Activité applicative - Identification de la personne appelante non faite"

445 PRECONDITIONS
      niv-umms #'(lambda (x) (and (typep x 'APPELANT) (reussi x))) -> umm-appelant
      niv-resu aucun #'(lambda (x) (or (disponible x) (en-test x)))
      niv-umms existe #'en-test -> umms-en-test
```

```
  ACTIONS
    (loop for umm in umms-en-test do (setf (statut umm) 'accompli))
    (setf (valeur umm-appelant) nil)
    (setf (statut umm-appelant) 'accompli)
)

; ------------------------------------------------------------------------------------------------------------------------

(defregle-r tache0-oublie-ancienne-tache *dialogueur* 8500

  DOCUMENTATION
    "Activité applicative - Il y a eu un changement de tache : il faut oublier l'ancienne"

  PRECONDITIONS
    niv-umms #'(lambda (x) (and (typep x 'REQUETE) (or (reussi x) (accompli x)))) -> umm-ancienne-requete
    niv-umms #'(lambda (x) (and (typep x 'REQUETE) (reussi x))) -> umm-nouvelle-requete

  CONDITION
    (not (eq umm-nouvelle-requete umm-ancienne-requete))

  ACTIONS
    (applique-statut umm-ancienne-requete 'oublie)
)

; ------------------------------------------------------------------------------------------------------------------------

(load "activite-applicative/tache1.lsp") ;; Chargement de la tâche 1
(load "activite-applicative/tache2.lsp") ;; Chargement de la tâche 2
(load "activite-applicative/tache3.lsp") ;; Chargement de la tâche 3
```

Listing: activite-applicative/tache1.lsp

```
; ##########################################################################################################################
; ### AMI2 - TACHE1.LSP (version du 01/12/1999) Thierry.Lemeunier@lium.univ-lemans.fr ######################################
; ##########################################################################################################################

; ==========================================================================================================================
; Traitement de la tâche principale de l'application AMI
; Mise en contact d'un appelant avec un appelé
; ==========================================================================================================================

; --------------------------------------------------------------------------------------------------------------------------

(defregle-r tache1-appele-present *dialogueur* 8000

  DOCUMENTATION
    "Activité 11 - Au moins un appelé est disponible"

  PRECONDITIONS
    niv-umms #'(lambda (x) (and (typep x 'CONTACTS) (reussi x))) -> umm-contacts
    niv-umms #'(lambda (x) (and (typep x 'APPELE) (accompli x))) -> umm-appele

  CONDITION
    (and (equal (pere umm-appele) umm-contacts)

         (est-present (valeur umm-appele))
     (est-dispo (valeur umm-appele)))

  ACTIONS
    (setf (statut umm-contacts) 'accompli)
)

; --------------------------------------------------------------------------------------------------------------------------
; Pour l'instant on part du principe qu'il n'y a qu'un seul appelé

(defregle-r tache1-appelant-a-des-messages *dialogueur* 8000

  DOCUMENTATION
    "Activité 11 - L'appelant a des messages"

  PRECONDITIONS
    niv-umms #'(lambda (x) (and (typep x 'PARLER) (reussi x))) -> umm-parler
    niv-umms #'(lambda (x) (and (typep x 'APPELANT) (accompli x))) -> umm-appelant
    niv-umms #'(lambda (x) (and (typep x 'LAISSER-MESSAGE) (en-sommeil x))) -> umm-laisser-message

  CONDITION
    (and (equal (pere umm-appelant) umm-parler)

         (loop for umm-appele-message in (fils (first (fils umm-laisser-message)))
           when (equalp (valeur umm-appelant) (valeur umm-appele-message))
           do (return t) finally (return nil)))

  ACTIONS
    (enleve-focus (pere umm-parler))
    (let ((umm-proposer (cree-umm-proposer :statut 'reussi))
          (umm-lire-message (cree-umm-lire-message :connu nil :valeur t)))
      (ajoute-fils umm-proposer umm-lire-message)
      (setf (valeur umm-proposer) umm-lire-message)
      (devient-focus umm-proposer))
)

; --------------------------------------------------------------------------------------------------------------------------

(defregle-r tache1-appelant-veux-lire-ses-messages *dialogueur* 7900

  DOCUMENTATION
    "Activité 11 - L'appelant veux bien lire ses messages"

  PRECONDITIONS
    niv-umms #'(lambda (x) (and (typep x 'PROPOSER) (reussi x))) -> umm-proposer
    niv-umms #'(lambda (x) (and (typep x 'LIRE-MESSAGE) (en-attente x))) -> umm-lire-message
    niv-umms #'(lambda (x) (and (typep x 'CONFIRMATION) (interprete x))) -> umm-confirmation

  CONDITION
    (equal (pere umm-lire-message) umm-proposer)

  ACTIONS
    (setf (statut umm-confirmation) 'utilise)
    (ajoute-fils umm-lire-message (cree-umm-appelant))
    (setf (statut umm-lire-message) 'reussi)
)

; --------------------------------------------------------------------------------------------------------------------------
; L'interet doit être inférieur à la sous-tache lire-messsage
; Le umm-proposer ne doit plus être demandable (pour être sûr que la proposition a été faite)

(defregle-r tache1-fin-lecture-des-messages *dialogueur* 6000

  DOCUMENTATION
    "Activité 11 - Fin de la proposition de lecteur des messages"
```

```
 90     PRECONDITIONS
          niv-umms #'(lambda (x) (and (typep x 'REQUETE) (reussi-sf x))) -> umm-requete
          niv-umms #'(lambda (x) (and (typep x 'PROPOSER) (or (reussi x) (accompli x)))) -> umm-proposer
          niv-umms #'(lambda (x) (and (typep x 'LIRE-MESSAGE))) -> umm-lire-message

 95     CONDITION
          (and (eq (pere umm-lire-message) umm-proposer)
         (not (demandable umm-proposer)))

        ACTIONS
100       (applique-statut umm-proposer 'oublie)
          (enleve-focus umm-proposer)
          (devient-focus umm-requete)
      )

105   ; ----------------------------------------------------------------------------------------------------------------------

      (defregle-r tache1-parler-appele *dialogueur* 7800

        DOCUMENTATION
110       "Activité i1 - L'appelé est présent et disponible"

        PRECONDITIONS
          niv-umms #'(lambda (x) (and (typep x 'REQUETE) (reussi x))) -> umm-requete
          niv-umms #'(lambda (x) (and (typep x 'PARLER) (reussi x))) -> umm-parler
115       niv-umms #'(lambda (x) (and (typep x 'CONTACTS) (accompli x))) -> umm-contacts
          niv-umms #'(lambda (x) (and (typep x 'APPELANT) (accompli x))) -> umm-appelant
          niv-umms #'(lambda (x) (and (typep x 'APPELE) (accompli x))) -> umm-appele

        CONDITION
120       (and (equal (pere umm-parler) umm-requete)
               (equal (pere umm-contacts) umm-parler)
               (equal (pere umm-appelant) umm-parler)
               (equal (pere umm-appele) umm-contacts)

125            (est-present (valeur umm-appele))
               (est-dispo (valeur umm-appele) (valeur umm-appelant)))

        ACTIONS
          (setf (valeur umm-parler) t)
130       (setf (statut umm-parler) 'accompli)
          (setf (demandable umm-requete) t)
          (setf (statut umm-requete) 'accompli)
      )

135   ; ----------------------------------------------------------------------------------------------------------------------

      (defregle-r tache1-appele-non-dispo *dialogueur* 7800

        DOCUMENTATION
140       "Activité applicative - La requête de l'utilisateur ne peut pas être satisfaite car \
           l'appelé n'est pas disponible"

        PRECONDITIONS
          niv-umms #'(lambda (x) (and (typep x 'REQUETE) (reussi x))) -> umm-requete
145       niv-umms #'(lambda (x) (and (typep x 'PARLER) (reussi x))) -> umm-parler
          niv-umms #'(lambda (x) (and (typep x 'CONTACTS) (reussi x))) -> umm-contacts
          niv-umms #'(lambda (x) (and (typep x 'APPELE) (accompli x))) -> umm-appele
          niv-umms aucun #'(lambda (x) (and (typep x 'PROPOSER) (reussi x)))

150     CONDITION
          (and (equal (pere umm-parler) umm-requete)
               (equal (pere umm-contacts) umm-parler)
         (equal (pere umm-appele) umm-contacts)
         (or (not (est-present (valeur umm-appele)))
155          (not (est-dispo (valeur umm-appele)))))

        ACTIONS
          (setf (valeur umm-contacts) umm-appele)
          (setf (demandable umm-contacts) t)
160   )

      ; ----------------------------------------------------------------------------------------------------------------------
      ; Pour l'instant on part du principe qu'il n'y a qu'un seul appelé

165   (defregle-r tache1-laisser-message-appele *dialogueur* 7000

        DOCUMENTATION
          "Activité i1 - L'appele est absent ou pas disponible : l'appelant veut-il laisser un message ?"

170     PRECONDITIONS
          niv-umms #'(lambda (x) (and (typep x 'REQUETE) (reussi x))) -> umm-requete
          niv-umms #'(lambda (x) (and (typep x 'PARLER) (reussi x))) -> umm-parler
          niv-umms #'(lambda (x) (and (typep x 'CONTACTS) (reussi x))) -> umm-contacts
          niv-umms #'(lambda (x) (and (typep x 'APPELANT) (or (accompli x) (en-attente x)))) -> umm-appelant
175       niv-umms #'(lambda (x) (and (typep x 'APPELE) (accompli x))) -> umm-appele
          niv-umms aucun #'(lambda (x) (and (typep x 'PROPOSER) (or (accompli x) (reussi x))))

        CONDITION
          (and (equal (pere umm-parler) umm-requete)
```

```
180     (equal (pere umm-contacts) umm-parler)
            (equal (pere umm-appele) umm-contacts)

            (or (est-absent (valeur umm-appele))
                (if (en-attente umm-appelant)
185         (and (est-present (valeur umm-appele))
                 (not (est-dispo (valeur umm-appele))))
                (and (est-present (valeur umm-appele))
               (not (est-dispo (valeur umm-appele) (valeur umm-appelant)))))))

190     ACTIONS
          (setf (demandable umm-requete) t)
          (let ((umm-proposer (cree-umm-proposer :statut 'reussi))
                (umm-laisser-message (cree-umm-laisser-message :connu nil :valeur t)))
            (ajoute-fils umm-proposer umm-laisser-message)
195         (setf (valeur umm-proposer) umm-laisser-message)
            (devient-focus umm-proposer))
          (setf (interet (unite 'tache1-laisser-message-appele)) 0) ;; Auto-modification de l'intérêt
        )

200     ; ------------------------------------------------------------------------------------------------------------------------
        ; Pour l'instant on part du principe qu'il n'y a qu'un seul appelé

        (defregle-r tache1-veux-laisser-message *dialogueur* 6900

205       DOCUMENTATION
            "Activité applicative 11 - L'appelant veux laisser un message à l'appelé (01/02/2000)"

          PRECONDITIONS
            niv-umms #'(lambda (x) (and (typep x 'REQUETE) (reussi x))) -> umm-requete
210
            niv-umms #'(lambda (x) (and (typep x 'PROPOSER) (reussi x))) -> umm-proposer
            niv-umms #'(lambda (x) (and (typep x 'LAISSER-MESSAGE) (en-attente x))) -> umm-laisser-message
            niv-umms #'(lambda (x) (and (typep x 'CONFIRMATION) (interprete x))) -> umm-confirmation

215       CONDITION
            (equal (pere umm-laisser-message) umm-proposer)

          ACTIONS
            (enleve-focus umm-requete)
220
            (setf (statut umm-confirmation) 'utilise)
            (ajoute-fils umm-laisser-message
              (ajoute-fils (cree-umm-contacts :statut 'reussi) (cree-umm-appele))
              (cree-umm-appelant)
225           (cree-umm-message :connu nil :valeur t))
            (setf (statut umm-laisser-message) 'reussi)

            (devient-focus umm-proposer)
        )
230
        ; ------------------------------------------------------------------------------------------------------------------------

        (defregle-r tache1-fin-laisser-message-appele *dialogueur* 6800

235       DOCUMENTATION
            "Activité applicative 11 - Fin de la prise de message"

          PRECONDITIONS
            niv-umms #'(lambda (x) (and (typep x 'REQUETE) (reussi x))) -> umm-requete
240         niv-umms #'(lambda (x) (and (typep x 'PARLER) (reussi x))) -> umm-parler

            niv-umms #'(lambda (x) (and (typep x 'PROPOSER) (or (reussi x) (accompli x)))) -> umm-proposer
            niv-umms #'(lambda (x) (and (typep x 'LAISSER-MESSAGE) (not (demandable x)))) -> umm-laisser-message

245       CONDITION
            (and (equal (pere umm-parler) umm-requete)
                 (equal (car (fils umm-proposer)) umm-laisser-message))

          ACTIONS
250         (if (accompli umm-proposer) (applique-statut umm-requete 'oublie))
            (enleve-focus umm-proposer)
            (applique-statut umm-proposer 'oublie)
        )

255     ; ------------------------------------------------------------------------------------------------------------------------
        ; Pour l'instant on part du principe qu'il n'y a qu'un seul appelé

        (defregle-r tache1-recherche-transfert-appele *dialogueur* 6700

260       DOCUMENTATION
            "Activité applicative 11 - Recherche des transferts"

          PRECONDITIONS
            niv-umms #'(lambda (x) (and (typep x 'PARLER) (reussi x))) -> umm-parler
265         niv-umms #'(lambda (x) (and (typep x 'CONTACTS) (reussi x))) -> umm-contacts
            niv-umms #'(lambda (x) (and (typep x 'APPELANT) (or (en-attente x) (accompli x)))) -> umm-appelant
            niv-umms #'(lambda (x) (and (typep x 'APPELE) (accompli x))) -> umm-appele

            niv-umms aucun #'(lambda (x) (and (typep x 'PROPOSER) (or (accompli x) (reussi x))))
```

```
270
      CONDITION
        (and (equal (pere umm-contacts) umm-parler)
             (equal (pere umm-appelant) umm-parler)
             (equal (pere umm-appele) umm-contacts))
275
      ACTIONS
        (let* ((groupe-appele (groupe (car (rechercher-personnes (list (list 'NOM (car (valeur umm-appele)))
                                                                       (list 'PRENOM (cadr (valeur umm-appele))))))))
               (autres-personnes
280       (when groupe-appele
            (loop for personne in (rechercher-personnes (list (list 'GROUPE groupe-appele)))
        when (if (accompli umm-appelant)
                 (est-dispo (list (lenom personne) (leprenom personne)) (valeur umm-appelant))
                 (est-dispo (list (lenom personne) (leprenom personne))))
285     collect personne))))

          (when (and groupe-appele autres-personnes)
        (setf autres-personnes (my-sort autres-personnes #'est-superieur))
        (devient-focus
290      (ajoute-fils
         (cree-umm-proposer :statut 'reussi)
         (ajoute-fils
          (cree-umm-parler :statut 'reussi)
          (ajoute-fils (cree-umm-contacts :statut 'reussi)
295        (loop for personne in autres-personnes
             for interet from 1 do
             collect (cree-umm-appele
                :interet (cdr (assoc (niveau personne) *niveau-assoc*))
                :connu nil
300             :valeur (list (lenom personne) (leprenom personne)))))
          (cree-umm-appelant))))))
      )

      ; ------------------------------------------------------------------------------------------------------------------------
305
      (defregle-r tache1-refuse-transfert-appele *dialogueur* 6600

        DOCUMENTATION
          "Activité applicative 11 - L'appelant refuse le transfert"
310
        PRECONDITIONS
          niv-umms #'(lambda (x) (and (typep x 'PROPOSER) (reussi x))) -> umm-proposer
          niv-umms #'(lambda (x) (and (typep x 'PARLER) (reussi x))) -> umm-parler
          niv-umms #'(lambda (x) (and (typep x 'CONTACTS) (reussi x))) -> umm-contacts
315
          niv-umms #'(lambda (x) (and (typep x 'INFIRMATION) (interprete x))) -> umm-infirmation

        CONDITION
          (and (equal (pere umm-parler) umm-proposer)
320      (equal (pere umm-contacts) umm-parler)
         (fils umm-contacts)
         (not (demandable (premier-fils umm-contacts))))

        ACTIONS
325       (setf (statut umm-infirmation) 'utilise)
          (applique-statut (premier-fils umm-contacts) 'inutile)
          (enleve-fils umm-contacts (premier-fils umm-contacts))
      )

330   ; ------------------------------------------------------------------------------------------------------------------------

      (defregle-r tache1-accepte-transfert-appele *dialogueur* 6500

        DOCUMENTATION
335       "Activité applicative 11 - L'appelant est d'accord pour un transfert"

        PRECONDITIONS
          niv-umms #'(lambda (x) (and (typep x 'REQUETE) (reussi x))) -> umm-requete
          niv-umms #'(lambda (x) (and (typep x 'PARLER) (reussi x))) -> umm-parler-1
340       niv-umms #'(lambda (x) (and (typep x 'CONTACTS) (reussi x))) -> umm-contacts-1

          niv-umms #'(lambda (x) (and (typep x 'PROPOSER) (reussi x))) -> umm-proposer
          niv-umms #'(lambda (x) (and (typep x 'PARLER) (reussi x))) -> umm-parler-2
          niv-umms #'(lambda (x) (and (typep x 'CONTACTS) (reussi x))) -> umm-contacts-2
345
          niv-umms #'(lambda (x) (and (typep x 'CONFIRMATION) (interprete x))) -> umm-confirmation

        CONDITION
          (and (equal (pere umm-parler-1) umm-requete)
350      (equal (pere umm-contacts-1) umm-parler-1)
         (equal (pere umm-parler-2) umm-proposer)
         (equal (pere umm-contacts-2) umm-parler-2)
         (not (demandable (premier-fils umm-contacts-2))))

355     ACTIONS
          (setf (statut umm-confirmation) 'utilise)

          (applique-statut-fils umm-contacts-1 'oublie)
          (enleve-fils umm-contacts-1 (fils umm-contacts-1))
```

```
360
    (let ((fils (premier-fils umm-contacts-2)))
      (setf (statut fils) 'accompli)
      (enleve-fils umm-contacts-2 fils)
      (ajoute-fils umm-contacts-1 fils))
365
    (applique-statut umm-proposer 'oublie)
)

; ------------------------------------------------------------------------------------------------------------------------
370
(defregle-r tachei-appele-non-ref *dialogueur* 6400

  DOCUMENTATION
    "Activité applicative - La requête de l'utilisateur ne peut pas être satisfaite car \
375   l'appelé n'est pas référencé."

  PRECONDITIONS
    niv-umms #'(lambda (x) (and (typep x 'REQUETE) (reussi x))) -> umm-requete
    niv-umms #'(lambda (x) (and (typep x 'PARLER) (reussi x))) -> umm-parler
380 niv-umms #'(lambda (x) (and (typep x 'CONTACTS) (reussi x))) -> umm-contacts
    niv-umms #'(lambda (x) (and (typep x 'APPELE) (reussi x))) -> umm-appele
    niv-umms aucun #'(lambda (x) (and (typep x 'PROPOSER) (or (accompli x) (reussi x))))

  CONDITION
385   (and (equal (pere umm-parler) umm-requete)
         (equal (pere umm-contacts) umm-parler)
   (equal (pere umm-appele) umm-contacts))

  ACTIONS
390   (setf (demandable umm-contacts) t)
)

; ------------------------------------------------------------------------------------------------------------------------
```

Listing: activite-applicative/tache2.lsp

```
; ##########################################################################################################################
; ### AMI2 - TACHE2.LSP (version du 28/04/1999) ############################################################################
; ##########################################################################################################################

; ==========================================================================================================================
; Traitement d'une tâche secondaire de l'application AMI
; L'appelant veux lire ses messages
; ==========================================================================================================================

; --------------------------------------------------------------------------------------------------------------------------

(defregle-r tache2-lecture-demande-message *dialogueur* 7000

  DOCUMENTATION
    "Activité applicative - Lecture des messages à l'attention de l'appelant"

  PRECONDITIONS
    niv-umms #'(lambda (x) (and (typep x 'LIRE-MESSAGE) (reussi x))) -> umm-lire-message
    niv-umms #'(lambda (x) (and (typep x 'APPELANT) (accompli x))) -> umm-appelant

    niv-umms #'(lambda (x) (and (typep x 'LAISSER-MESSAGE) (en-sommeil x))) -> umm-laisser-message

  CONDITION
    (and (equal (pere umm-appelant) umm-lire-message)

         (loop for umm-appele-message in (fils (first (fils umm-laisser-message)))
           when (equalp (valeur umm-appelant) (valeur umm-appele-message))
           do (return t) finally (return nil)))

  ACTIONS
    (setf (interet (unite 'tache2-aucun-message)) 0) ;; Modification de l'intérêt de la Ksource
    (setf (interet (unite 'tache2-fin-lecteur-message)) 6900) ;; Modification de l'intérêt de la Ksource

    (let* ((new-umm-laisser-message (copie-profonde-1 umm-laisser-message)))

       (loop for appele in (fils (first (fils umm-laisser-message)))
         when (equalp (valeur umm-appelant) (valeur appele)) do
         (enleve-fils (first (fils umm-laisser-message)) appele))

       (unless (fils (first (fils umm-laisser-message)))
          (applique-statut umm-laisser-message 'inutile))

       (loop for appele in (fils (first (fils new-umm-laisser-message)))
         unless (equalp (valeur umm-appelant) (valeur appele)) do
         (enleve-fils (first (fils new-umm-laisser-message)) appele))

       (setf (valeur new-umm-laisser-message) (donne-message new-umm-laisser-message))
       (devient-focus new-umm-laisser-message)
       (applique-statut new-umm-laisser-message 'accompli))
)

; --------------------------------------------------------------------------------------------------------------------------
;; La valeur de l'intêrêt est modifiée par la KSOURCE tache2-lecture-demande-message

(defregle-r tache2-fin-lecteur-message *dialogueur* 0

  DOCUMENTATION
    "Activité applicative - Il n'y a plus de message pour l'appelant"

  PRECONDITIONS
    niv-umms #'(lambda (x) (and (typep x 'LIRE-MESSAGE) (reussi x))) -> umm-lire-message
    niv-umms #'(lambda (x) (and (typep x 'APPELANT) (accompli x))) -> umm-appelant

  CONDITION
    (equal (pere umm-appelant) umm-lire-message)

  ACTIONS
    (when (pere umm-lire-message)
      (setf (statut (pere umm-lire-message)) 'oublie)
      (setf (fils (pere umm-lire-message)) nil)
      (setf (pere umm-lire-message) nil))

    (applique-statut umm-lire-message 'accompli)
    (setf (demandable umm-lire-message) t)
    (setf (valeur umm-lire-message) t)
)

; --------------------------------------------------------------------------------------------------------------------------

(defregle-r tache2-suppression-des-messages *dialogueur* 6900

  DOCUMENTATION
    "Activité applicative - Il faut supprimer les messages apres les avoir lus à l'appelant"

  PRECONDITIONS
    niv-umms #'(lambda (x) (and (typep x 'LAISSER-MESSAGE) (accompli x))) -> umm-laisser-message

  CONDITION
    (and (not (pere umm-laisser-message))
```

```
        (not (demandable umm-laisser-message)))

  ACTIONS
    (applique-statut umm-laisser-message 'oublie)
)

; ----------------------------------------------------------------------------------------------------------------------

(defregle-r tache2-aucun-message *dialogueur* 6800

  DOCUMENTATION
    "Activité applicative - Il n'y aucun message pour l'appelant"

  PRECONDITIONS
    niv-umms #'(lambda (x) (and (typep x 'REQUETE) (reussi x))) -> umm-requete
    niv-umms #'(lambda (x) (and (typep x 'LIRE-MESSAGE) (reussi x))) -> umm-lire-message
    niv-umms #'(lambda (x) (and (typep x 'APPELANT) (accompli x))) -> umm-appelant

  CONDITION
    (and (equal (pere umm-lire-message) umm-requete)
         (equal (pere umm-appelant) umm-lire-message))

  ACTIONS
    (setf (fils umm-requete) nil)
    (setf (statut umm-requete) 'oublie)
    (setf (pere umm-lire-message) nil)
    (setf (demandable umm-lire-message) t))

; ----------------------------------------------------------------------------------------------------------------------
```

Listing: activite-applicative/tache3.lsp

```
; ##########################################################################################################################
; ### AMI2 - TACHE3.LSP (version du 21/10/1998) ############################################################################
; ##########################################################################################################################

; ==========================================================================================================================
; Traitement d'une tâche secondaire de l'application AMI
; L'appelant veux laisser un message pour un appelé.
; ==========================================================================================================================

; --------------------------------------------------------------------------------------------------------------------------

(defregle-r tache3-ajoute-contacts *dialogueur* 9150

  DOCUMENTATION
    "Activité applicative - Fin de la liste des contacts"

  PRECONDITIONS
    niv-umms #'(lambda (x) (and (typep x 'LAISSER-MESSAGE) (reussi x))) -> umm-laisser-message
    niv-umms #'(lambda (x) (and (typep x 'CONTACTS) (reussi x))) -> umm-contacts
    niv-umms #'(lambda (x) (and (typep x 'APPELE) (accompli x))) -> umm-appele

  CONDITION
    (and (equal (pere umm-contacts) umm-laisser-message)
         (equal (pere umm-appele) umm-contacts))

  ACTIONS
    (setf (valeur umm-contacts) (fils umm-contacts))
    (setf (statut umm-contacts) 'accompli)
)

; --------------------------------------------------------------------------------------------------------------------------
; L'intérêt est grand pour ne pas faire d'analyse

(defregle-r tache3-fin-ident-message-appele *dialogueur* 60000

  DOCUMENTATION
    "Activité applicative - Fin de la prise de message (01/02/2000)"

  PRECONDITIONS
    niv-enonceh #'disponible -> enonceh

    niv-umms #'(lambda (x) (and (typep x 'LAISSER-MESSAGE) (reussi x))) -> umm-laisser-message
    niv-umms #'(lambda (x) (and (typep x 'CONTACTS) (accompli x))) -> umm-contacts
    niv-umms #'(lambda (x) (and (typep x 'APPELANT) (accompli x))) -> umm-appelant
    niv-umms #'(lambda (x) (and (typep x 'MESSAGE) (en-attente x))) -> umm-message

  CONDITION
    (and (equal (pere umm-contacts) umm-laisser-message)
         (equal (pere umm-appelant) umm-laisser-message)
         (equal (pere umm-message) umm-laisser-message))

  ACTIONS
    (setf (statut enonceh) 'utilise)
    (setf (valeur umm-message) (expression-initiale enonceh))

    (setf (demandable (pere umm-laisser-message)) t)
    (setf (statut (pere umm-laisser-message)) 'accompli)
    (setf (fils (pere umm-laisser-message)) nil)

    (setf (pere umm-laisser-message) nil)
    (setf (valeur umm-laisser-message) t)
    (applique-statut umm-laisser-message 'en-sommeil)
)

; --------------------------------------------------------------------------------------------------------------------------
; [tache3-appele-non-reference] (l'appelé n'est pas référencé)

(defregle-r tache3-appele-non-reference *dialogueur* 6800

  DOCUMENTATION
    "Activité applicative - L'appelé n'est pas référencé"

  PRECONDITIONS
    niv-umms #'(lambda (x) (and (typep x 'REQUETE) (reussi x))) -> umm-requete
    niv-umms #'(lambda (x) (and (typep x 'LAISSER-MESSAGE) (reussi x))) -> umm-laisser-message
    niv-umms #'(lambda (x) (and (typep x 'CONTACTS) (reussi x))) -> umm-contacts

  CONDITION
    (and (equal (pere umm-laisser-message) umm-requete)
         (equal (pere umm-contacts) umm-laisser-message))

  ACTIONS
    (setf (demandable umm-requete) t)
)

; --------------------------------------------------------------------------------------------------------------------------
```

Listing: activite-applicative/personne.lsp

```
; ##########################################################################################################################
; ### AMI1 - PERSONNE.LSP - LEMEUNIER (version du 26/11/1999) Thierry.Lemeunier@lium.univ-lemans.fr ######################
; ##########################################################################################################################

(defun construit-critere-nom-prenom (liste-nom-prenom)
  (list (list 'NOM (car liste-nom-prenom))
  (list 'PRENOM (cadr liste-nom-prenom))))

(defun est-absent (liste-nom-prenom)
  (est-absent-interne (car (rechercher-personnes (construit-critere-nom-prenom liste-nom-prenom)))))

(defun est-present (liste-nom-prenom)
  (est-present-interne (car (rechercher-personnes (construit-critere-nom-prenom liste-nom-prenom)))))

(defun est-dispo (liste1-nom-prenom &optional (liste2-nom-prenom nil))
   (if liste2-nom-prenom
      (est-dispo-interne (car (rechercher-personnes (construit-critere-nom-prenom liste1-nom-prenom)))
                         (car (rechercher-personnes (construit-critere-nom-prenom liste2-nom-prenom))))
      (est-dispo-interne (car (rechercher-personnes (construit-critere-nom-prenom liste1-nom-prenom))))))

; ==========================================================================================================================
; Description de la classe ITEM-PERSONNE (les instances des personnes extraits de la base de données)
; ==========================================================================================================================

(defclass ITEM-PERSONNE (UNITE)
  ((chaine :allocation :class :reader chaine :initform 'item-perso)
   (indesirable :accessor indesirable :initarg :indesirable :initform nil)
   (emploi :reader emploi :initarg :emploi :type array :initform (make-array '(11 5)))
   (lenom :reader lenom :initarg :lenom :initform "")
   (leprenom :reader leprenom :initarg :leprenom :initform "")
   (poste :reader poste :initarg :poste :initform "")
   (responsable :reader responsable :initarg :responsable :initform nil)
   (groupe :reader groupe :initarg :groupe :initform nil)
   (niveau :reader niveau :initarg :niveau :initform nil)))

(defmacro cree-item-personne (&rest -args) `(make-instance 'ITEM-PERSONNE ,@-args))

(defun ressemble (-source -cible)
  (equalp (mise-en-forme -source) (mise-en-forme -cible)))

(defmethod verifie ((oself ITEM-PERSONNE) critere)
  (case (car critere)
    (NOM         (ressemble (cadr critere) $lenom))
    (PRENOM      (ressemble (cadr critere) $leprenom))
    (POSTE       (ressemble (cadr critere) $poste))
    (GROUPE      (equal (cadr critere) $groupe))
    (RESPONSABLE (member (cadr critere) $responsable))))

(defmethod correspond ((oself ITEM-PERSONNE) criteres)
  (or (null criteres)
      (and (verifie oself (car criteres))
     (correspond oself (cdr criteres)))))

(defmethod meme ((oself ITEM-PERSONNE) (-personne ITEM-PERSONNE))
  (and (equalp $lenom (lenom -personne))
       (equalp $leprenom (leprenom -personne))))

(defmethod actuel ((oself ITEM-PERSONNE))
;;;   (aref $emploi (- (lheure) 8) (jour)))
   (aref $emploi 1 1))

(defmethod est-libre ((oself ITEM-PERSONNE)) (equal "R" $actuel))

(defmethod en-cours ((oself ITEM-PERSONNE)) (equal "C" $actuel))

(defmethod au-repas ((oself ITEM-PERSONNE)) (equal "RE" $actuel))

(defmethod en-tp ((oself ITEM-PERSONNE)) (equal "TP" $actuel))

(defmethod en-td ((oself ITEM-PERSONNE)) (equal "TD" $actuel))

(defmethod est-absent-interne ((oself ITEM-PERSONNE)) (equal "ABS" $actuel))

(defmethod est-present-interne ((oself ITEM-PERSONNE)) (not $est-absent-interne))

(defmethod est-dispo-interne ((oself ITEM-PERSONNE) &optional (-personne nil))
  (and (if -personne (not (member (lenom -personne) $indesirable :test #'equalp)) t) $est-libre))

(defvar *niveau-assoc* (pairlis '(Prof PRAG MdC PAST Ingenieur Secretaire Technicien ATER Vacataire Thesard)
        '(6   6   5   4   3       2         2         2   2       1)))

(defmethod est-superieur ((pers1 ITEM-PERSONNE) (pers2 ITEM-PERSONNE))
  "Prof = PRAG > MdC > PAST > Ingenieur > Secretaire = Technicien = ATER = Vacataire > Thesard"
  (>= (cdr (assoc (niveau pers1) *niveau-assoc*)) (cdr (assoc (niveau pers2) *niveau-assoc*))))

; ==========================================================================================================================
; Une fonction de recherche multicritères (rechercher-personnes '((prenom "daniel") (poste "3865")))
; ==========================================================================================================================
```

```
(defun rechercher-personnes (criteres)
  (loop for membre-labo in (lister-unites 'ITEM-PERSONNE)
  when (correspond membre-labo criteres)
  collect membre-labo))

; ================================================================================================================================
; Description de la classe RESULTAT (un résultat de la recherche dans l'annuaire)
; ================================================================================================================================

(defclass RESULTAT (ELEMDIAL UNITE)
  ((niveau :allocation :class :reader niveau :initform 'niv-resu)
   (valeur :reader valeur :initarg :valeur)
   (origine :reader origine :initarg :origine)))

(defmacro cree-resultat (&rest -args) `(make-instance 'RESULTAT ,@-args))

(defmethod description ((oself RESULTAT))
  (format nil "Resultat [Valeur ~A]" $valeur))

(defmethod accepte ((oself RESULTAT)) (equal $statut 'accepte))
```

Listing: activite-applicative/annuaire.lsp (extrait)

```
;; ##########################################################################################################################
;; ### AMI2 - ANNUAIRE.LSP - LEMEUNIER 26/11/99 (Thierry.Lemeunier@lium.univ-lemans.fr)
;; ##########################################################################################################################

;; Ici sont créés toutes les personnes du laboratoire

; Equipes :
;

; Les niveaux des personnes permet de les classer
; Prof > MdC = PRAG = Ingenieur > PAST > Secretaire = Technicien = ATER = Vacataire > Thesard

; ----------------------------------------------------------------------------
; Prof

(cree-item-personne
 :LENOM "Luzzati"
 :LEPRENOM "Daniel"
 :POSTE "3857"
 :RESPONSABLE '(GE1G1 GE1G2)
 :GROUPE 'GE1G2
 :NIVEAU 'Prof
 :EMPLOI (make-array '(11 5)
        :initial-contents '(( "C" "C"  "R"   "ABS" "ABS" )    ; 8 - 9 h
          ( "R"   "C"   "R"   "C"   "ABS" )    ; 9 - 10 h
          ( "C"   "TD"  "R"   "C"   "ABS" )    ; 10 - 11 h
          ( "C"   "TD"  "R"   "R"   "ABS" )    ; 11 - 12 h
          ( "RE"  "RE"  "RE"  "R"   "ABS" )    ; 12 - 13 h
          ( "R"   "R"   "C"   "R"   "TP" )     ; 13 - 14 h
          ( "R"   "R"   "C"   "R"   "TP" )     ; 14 - 15 h
          ( "ABS" "R"   "C"   "R"   "TP" )     ; 15 - 16 h
          ( "ABS" "R"   "TD"  "R"   "ABS" )    ; 16 - 17 h
          ( "ABS" "ABS" "R"   "ABS" "ABS" )    ; 17 - 18 h
          ( "ABS" "ABS" "ABS" "R"   "ABS" )))) ; 18 - 19 h

; ----------------------------------------------------------------------------

(cree-item-personne
 :LENOM "Vivet"
 :LEPRENOM "Martial"
 :POSTE nil
 :RESPONSABLE nil
 :GROUPE nil
 :NIVEAU 'Prof
 :EMPLOI (make-array '(11 5)
        :initial-contents '(( "ABS" "ABS" "ABS" "ABS" "ABS" ) ; 8 - 9 h
          ( "ABS" "ABS" "ABS" "ABS" "ABS" )  ; 9 - 10 h
          ( "ABS" "ABS" "ABS" "ABS" "ABS" )  ; 10 - 11 h
          ( "ABS" "ABS" "ABS" "ABS" "ABS" )  ; 11 - 12 h
          ( "ABS" "ABS" "ABS" "ABS" "ABS" )  ; 12 - 13 h
          ( "ABS" "ABS" "ABS" "ABS" "ABS" )  ; 13 - 14 h
          ( "ABS" "ABS" "ABS" "ABS" "ABS" )  ; 14 - 15 h
          ( "ABS" "ABS" "ABS" "ABS" "ABS" )  ; 15 - 16 h
          ( "ABS" "ABS" "ABS" "ABS" "ABS" )  ; 16 - 17 h
          ( "ABS" "ABS" "ABS" "ABS" "ABS" )  ; 17 - 18 h
          ( "ABS" "ABS" "ABS" "ABS" "ABS" )))) ; 18 - 19 h

; ----------------------------------------------------------------------------
```

Listing: activite-applicative/gestion-message.lsp

```
;; ##########################################################################################################################
;; ### AMI2 - GESTION-MESSAGE.LSP - Lemeunier 20/01/2000 (Thierry.Lemeunier@lium.univ-lemans.fr) ######################
;; ##########################################################################################################################

;; --------------------------------------------------------------------------------------------------------------------------

(defun charge-messages ()
  "Chargement des messages"
  (when (probe-file *fichier-message*)
    (with-open-file
     (ifile *fichier-message* :direction :input)
     (loop while (setf ligne (read ifile nil nil)) do
       (applique-statut
  (ajoute-fils
   (make-instance 'LAISSER-MESSAGE :valeur t)
   (ajoute-fils (make-instance 'CONTACTS)
         (loop for appele in (first ligne) do
       collect (make-instance 'APPELE :valeur appele)))
   (make-instance 'APPELANT :valeur (second ligne))
   (make-instance 'MESSAGE :valeur (third ligne) :quand (fourth ligne)))
  'en-sommeil))))

;; --------------------------------------------------------------------------------------------------------------------------

(defun sauve-messages ()
  "Sauvegarde des messages dans le fichier des messages"
  (when (probe-file *fichier-message*) (delete-file *fichier-message*))
  (let ((umms-laisser-message (my-find-if #'(lambda (x) (en-sommeil x)) (lister-unites 'LAISSER-MESSAGE))))
    (if umms-laisser-message
  (with-open-file
   (ofile *fichier-message* :direction :output :if-does-not-exist :create)
   (loop for umm-laisser-message in umms-laisser-message do
     (write (cons
       (loop for appele in
       (fils (premier-fils umm-laisser-message :predicat #'(lambda (x) (typep x 'CONTACTS))))
       do collect (valeur appele))
       (cons
        (valeur (premier-fils umm-laisser-message :predicat #'(lambda (x) (typep x 'APPELANT))))
        (cons
         (valeur (premier-fils umm-laisser-message :predicat #'(lambda (x) (typep x 'MESSAGE))))
         (list (quand (premier-fils umm-laisser-message :predicat #'(lambda (x) (typep x 'MESSAGE))))))))
      :stream ofile)
     (fresh-line ofile))))))

;; --------------------------------------------------------------------------------------------------------------------------
```

L'intentionnalité communicative dans le dialogue homme-machine en langue naturelle

Résumé :

Notre travail de thèse porte sur la modélisation des intentions de communications des systèmes de dialogue homme-machine en langue naturelle et de manière générale de tout agent logiciel. Il s'agit de proposer un modèle de fonctionnement permettant à la machine d'avoir ses propres intentions de communiquer avec l'utilisateur humain, et ceci de telle sorte que la pertinence optimale de ses énonciations soit garantie.

Notre modèle s'appuie sur l'idée que le sens échangé entre les interactants d'une conversation n'est pas un sens préexistant à celle-ci, mais au contraire, un sens négocié et co-construit par les interactants durant la conversation. Cette co-construction s'appuie sur l'hypothèse de l'existence d'un terrain commun, c'est-à-dire d'un ensemble de connaissances, hypothèses et croyances que le locuteur pense être partagées.

Notre travail a consisté à définir une mémoire interactionnelle pour la machine permettant le travail de négociation du sens. Cette mémoire contient des éléments de différents états organisés en arborescences. Ces éléments proviennent de l'interprétation des actes illocutoires de l'utilisateur et des résultats des raisonnements faits par les différentes activités du système de dialogue. Nous distinguons l'activité applicative dont le but est de fournir un service quelconque à l'utilisateur, l'activité langagière qui consiste à analyser les énoncés de l'utilisateur et générer les énoncés du système, et enfin l'activité dialogique qui consiste à dialoguer avec l'utilisateur. Les intentions de communications de la machine sont générées par la reconnaissance de configurations remarquables que nous avons définies en étudiant les arborescences qu'il est normalement possible d'obtenir. Ce principe de génération, à l'origine des actes langagiers de la machine, est général et indépendant de l'application. Il s'appuie uniquement sur la forme structurelle des éléments mnésiques (appelé UMM pour Unité Minimale de Mémoire) et sur l'état de ces derniers.

Mots clés :

intentions communicatives, dialogue homme-machine, planification *versus* oppurtunisme, négociation du sens interactionnel, mémoire interactionnelle, théorie des actes de langage

Communicative Intentionality in Human-Computer Dialog in Natural Language

Summary:

My thesis is about the modelization of the communicative intentions of human-computer dialog systems in natural language and more generally, about any artificial agent. The goal is to propose a model of processing that gives its own communicative intentions to the computer to interact with human users, so that optimal relevance of the computer utterances is certified.

My model is based upon the idea that the sense exchanged between dialog participants is not a pre-existing sense but, by opposition, a sense negotiated and co-constructed by participants during the dialog. This co-construction depends on the hypothesis of the existence of a common ground, that is a knowledge set assumptions and believes that the speaker thinks to be shared.

My work has consisted in defining a computer interactional memory allowing the sense negotiation task. This memory contains elements of three different states organized in tree diagrams. These elements become from interpretations of user's illocutionary acts and from inferences of the activities of the dialog system. We distinguish three activities: the applicative activity performs any service to the human user, the linguistic activity analyzes user utterances and generates system utterances, and the dialogical activity converses with the human user in the sense previously defined. Computer communicative intentions are generated from the recognition of a set of configurations that I have defined by analyzing empirically the possible tree structures. This principle that gives computer speech acts, is general and independent of the field application. It is just based on the structural shape of the memory elements (called MMU for Minimal Memory Unit) and the state of these elements.

Keywords:

communicatives intentions, human-computer dialogue, planning against opportunism, interactional sense negotiation, interactional memory, speech acts theory