Co-construction de sens par négociation pour la réutilisation en situation de l'expérience tracée - Vers le partage et l'échange d'expérience collective

Arnaud Stuber

EDIIS

Université Claude Bernard
Lyon I

École doctorale Informatique
et Information pour la Société

Co-construction de sens par négociation pour la réutilisation en situation de l'expérience tracée

Vers le partage et l'échange d'expérience collective

THÈSE

présentée et soutenue publiquement le 4 décembre 2007

pour l'obtention du

Doctorat de l'Université Claude Bernard Lyon I

(spécialité informatique)

par

Arnaud Stuber

Composition du jury

Président :	Amal El Fallah Seghrouchni, Professeur, LIP6, Université P.&M. Curie Paris VI
Rapporteurs :	Ramon López de Mántaras, Research Professor CSIC, IIIA, Barcelona, Spain Jean-Paul Sansonnet, Directeur de recherche CNRS, LIMSI-CNRS, Orsay
Examinateur :	Catherine Garbay, Directeur de Recherche CNRS, CLIPS-IMAG, Grenoble
Directeurs :	Salima Hassas, Professeur, LIESP, Université Claude Bernard Lyon I Alain Mille, Professeur, LIRIS, Université Claude Bernard Lyon I

Ordre de Thèse n° 305.2007

Laboratoire d'InfoRmatique en Images et Systèmes d'information — UMR 5205

LIRIS

Mis en page avec la classe thloria.

Remerciements

Je tiens avant tout à remercier les personnes qui ont fait bien plus qu'encadrer mes deux travaux de recherche. En premier lieu, mes remerciements vont à Alain MILLE, directeur de ma thèse, pour les très nombreux échanges, ô combien enrichissants, que nous avons eu ensemble ainsi que pour son précieux soutien.

Mais mes remerciements vont également à Salima HASSAS, co-encadrante de ma thèse, qui a très largement influé sur ma réflexion ; sans elle, ce travail ne serait pas devenu ce qu'il est. C'est donc en ce sens que je me permets de présenter Salima Hassas et Alain Mille comme mes deux directeurs de thèse, qui ont su articuler au travers de mon travail leurs intérêts de recherche propres et complémentaires.

D'autre part, je tiens à préciser que ce travail de thèse n'a pu exister que grâce au stage[1] de DEA effectué avec Christine SOLNON. En ce temps, Christine a fait bien plus que me donner le goût de la recherche et je ne peux donc que lui réitérer mes profonds remerciements pour tout ce qu'elle a fait.

Mes remerciements vont ensuite aux membres du jury. Tout d'abord, ils vont à Amal El Fallah Seghrouchni pour avoir accepté de presider mon jury de thèse. Ils vont ensuite à Catherine Garbay, à Ramon López de Mántaras et à Jean-Paul Sansonnet qui ont, de la relecture à la soutenance, grandement enrichi ce travail autant par leurs critiques constructives que par les questions fondamentales, qu'ils ont soulevées lors de la soutenance.

Je remercie également tous les membres du *Projet OSCAR* pour les échanges que nous avons eu au sujet de la problématique de Capitalisation des Connaissances. J'en profite pour remercier la *Région Rhône-Alpes*, qui est a l'origine du Projet OSCAR et donc de ma thèse. Cela me conduit à remercier également l'*UFR* et l'*IUT d'Informatique* de l'*UCBL* pour leurs soutiens et pour les experiences d'enseignement qu'ils m'ont données.

Mes remerciements vont aussi à tous les habitants du bâtiment *Nautibus* pour l'agréable ambiance qu'ils y font régner et l'émulation qui en résulte. Tout particulièrement, mes remerciements s'adressent aux membres de l'équipe *SILEX*, animée par Alain Mille, et aux membres de l'équipe *SyCoSMA*, animée par Salima Hassas.

En dernier lieu, et non des moindres, un immense merci à toutes celles et à tous ceux qui m'ont entouré lors de ces années de thèse. Ne voulant les citer, j'espère qu'ils se reconnaîtront...

[1] travail ayant porté sur l'extraction automatique de l'expérience collective acquise par une colonie de fourmis (ACO) lors de la résolution d'un problème combinatoire, et la réutilisation de cette expérience dans une nouvelle tentative (collective) de résolution, c.f., [Stuber 01]

à Anita,
à Paul.

Table des matières

Chapitre 1

Introduction

La problématique est de concevoir un assistant personnalisé, adossé à une application métier, permettant à chaque acteur humain de réutiliser son expérience individuelle et de la partager avec d'autres acteurs. Cet assistant doit être conçu dans le but ultime de permettre la construction et la réutilisation d'une expérience collective.

Le verrou sous-jacent à cette problématique est l'acquisition des connaissances d'identification de situation d'assistance, qui fassent sens à l'utilisateur et qui soient exploitables par l'assistant personnalisé. Pour convenir de telles connaissances, nous allons transposer les mécanismes d'émergence d'un langage de forme lexicale à la "communication" s'effectuant entre l'acteur humain et son assistant lors des phases d'assistance.

Dans la section suivante, nous allons présenter le projet OSCAR dans le cadre duquel s'est déroulée cette recherche. Nous introduirons ensuite le plan de ce mémoire.

Le projet OSCAR

La recherche présentée trouve ses racines dans le cadre du projet OSCAR (*Organisation des Simulations en Conception par la CApitalisation et la Réutilisation*) de la *Thématique Productique - Génie industriel* de la Région Rhône-Alpes. Ce projet a regroupé les laboratoires *GILCO*[1], *3S*[2], *IMAG*[3] et *LIRIS*[4], en partenariat avec *Schneider Electric*[5], *PCO Technologies*[6] et *Renault*[7]. Il fait suite à une collaboration entre *Schneider Electric* et les laboratoires 3S et GILCO (projet SG3C, avec la thèse de Nadège Troussier, c.f., [Troussier 99]) qui avait pour objectifs d'apporter un certain nombre de réponses très concrètes à des besoins industriels exprimés à la fois en termes d'organisations d'entreprises, de méthodes de travail et d'outils associés. Les travaux réalisés ont permis de dégager une méthode et un outil support (SIMMANAGER) pour favoriser la gestion des multiples calculs générés lors du développement de nouveaux produits techniques. Un dispositif est proposé pour améliorer la coopération entre les analystes qui réalisent les simulations et les concepteurs, dans un contexte d'ingénierie simultanée et distribuée. La méthode et l'outil fournissent un support à une traçabilité documentaire, en vue de la réutilisation des modèles et des démarches de simulations. Après ce premier travail, il est apparu le besoin de disposer d'outils

[1] Laboratoire de Recherche de l'ENSGI-INPG - *http ://gilco.inpg.fr/*
[2] Laboratoire 3S - pôle Conception Intégrée - *http ://www.3s.hmg.inpg.fr/*
[3] LSR IMAG équipe SIGMA – *http ://www-lsr.imag.fr*
[4] LIRIS, Axe 1, équipe SILEX (Supporting Interaction and Learning by Experience) *http ://liris.cnrs.fr*
[5] *Schneider Electric* - BT
[6] *PCO Technologies – http ://www.pcotech.fr*
[7] *Renault – DIEC*

informatiques permettant : 1) le support de la tâche des différents acteurs, en assurant la gestion de documents descriptifs des différentes étapes de l'activité collaborative de conception intégrée, 2) l'assistance proactive pour faciliter l'accès et la réutilisation de l'expérience acquise dans des situations analogues.

Cette thèse s'est effectuée en collaboration avec *PCO Technologies*, dont le rôle était de spécifier un modèle d'application informatique documentaire servant de support à une activité coopérative, sur la base des travaux de modélisation d'activité et de développement d'outils de support qu'ils mènent chez leurs clients.

Plan du mémoire

Le chapitre 2 présente une synthèse de l'appropriation que nous avons fait des résultats du projet SG3C, en collaboration avec les laboratoires 3S et GILCO, et notre partenaire *PCO Technologies*. Cette synthèse est placée dans la problématique générale de la gestion et de la capitalisation des connaissances, et nous permet d'identifier les besoins et les conditions que doivent remplir les conteneurs de connaissances manipulés durant une activité collaborative. Ce chapitre nous conduit à nous intéresser à l'expérience située, dans le cadre de l'interaction entre un acteur humain et un outil informatique.

Le chapitre 3 présente les problématiques fondamentales rencontrées lors de la création de systèmes intelligents, qui résonnent avec celles énoncées au chapitre précédent, et argumente l'intérêt d'adopter une approche intermédiaire que constitue le RàPC. Les limites de ce paradigme sont discutées, et les motivations pour son évolution à des situations non prédéfinies sont introduites.

Dans le but de répondre au besoin de personnalisation de notre assistant, nous faisons au chapitre 4 une analyse des travaux menés sur la personnalisation à base de profil. Le besoin de proactivité de notre logiciel assistant, adossé à une application tierce, est alors mis en relation avec les travaux sur les agents interface. Les difficultés rencontrées par ces approches et leurs dernières évolutions allant vers la prise en compte de l'utilisation, nous conduisent à aborder la question de l'adaptation des applications à leurs utilisateurs sous la forme d'une appropriation en situation ; cela est également dans la continuité des points abordés en conclusion du chapitre précédent sur le RàPC. Nous présentons alors les principes du RàPET, une évolution du RàPC, qui est étudiée par l'équipe SILEX. Nous allons adopter le principe du RàPET, et le modèle MUSETTE qui s'y rattache, pour notre assistant. La problématique fondamentale commune à tous les systèmes d'assistance, qui est de disposer de sens partagés par l'acteur humain et son assistant informatique, afin que ce dernier puisse effectuer des suggestions d'expérience pertinentes, est présentée en 4.5 ; c'est par rapport à cette problématique que nous positionnons celle de la thèse. Ceci nous conduit à présenter les travaux sur l'émergence de langage dans une population d'agents, sans supervision, que nous transposerons à notre problématique.

Dans la deuxième partie, nous présenterons notre contribution. Les principes de notre approche sont présentés au chapitre 5 et nous présentons l'architecture de notre assistant pour faciliter la réutilisation contextuelle de l'expérience tracée ; cet assistant est articulée autour d'un mécanisme de RàPET et manipule les éléments de sens qu'il acquiert selon les principes d'émergence de langage. Le chapitre 6 détaille notre module de RàPET et la représentation de

trace guidée par principes du modèle MUSETTE, dont nous ferons une extension. Le chapitre 7 présente la transposition des mécanismes d'émergence de langage pour assurer l'acquisition de sens d'interprétation des traces ; de telle manière que ces sens soient admis par l'acteur humain et utilisables avec suffisamment d'efficacité par l'assistant. Notre travail s'effectue dans la perspective de permettre le partage d'expérience entre acteurs via le système. Le chapitre 9 porte sur le démonstrateur développé pour illustrer notre approche.

Le chapitre 10 présente les perspectives qu'offrent notre approche pour la construction d'un système d'assistance, soit principalement, 1) l'utilisation de cette transposition pour l'échange et le partage d'expériences individuelles et collectives, 2) le besoin et la possibilité d'ajouter une dimension syntaxique au langage émergent, afin d'augmenter son expressivité. Enfin, nous concluons et discutons in fine du caractère plus général de notre contribution.

Première partie

Référentiel scientifique

Chapitre 2

Partage et réutilisation de connaissance en conception intégrée

Sommaire

Le projet OSCAR (Organisation des Simulations en Conception par la CApitalisation et la Réutilisation) intègre une méthode permettant de gérer et de capitaliser les connaissances de simulation, qui sont mobilisées durant des activités de conception. Dans ce chapitre, nous allons dans un premier temps brièvement aborder la problématique générale de la gestion et de la capitalisation des connaissances, dans laquelle le projet OSCAR prend place. Ensuite, nous traiterons le cas de l'activité de conception intégrée en présentant le modèle SG3C qui est à la base du projet OSCAR. Enfin, nous positionnerons notre travail par rapport à la problématique générale de capitalisation des connaissances, et aux résultats du projet SG3C, introduisant les questions plus théoriques ensuite abordées.

En faisant cela, nous ne cherchons pas à faire un état de l'art des travaux menés dans le domaine de la capitalisation des connaissances, mais à faire une introduction au contexte d'application du travail présenté.

2.1 Gestion et capitalisation des connaissances - problématique

Cette section a pour objectif d'identifier les concepts et difficultés fondamentaux qui sont présents dans divers travaux de *gestion* et de *capitalisation* des *connaissances* menés dans les entreprises. Nous nous appuyons sur l'analyse détaillée faite par [Baizet 04].

Des travaux sur la gestion des connaissances, il apparaît que l'objectif commun est de mettre en place des politiques managériales pour les "connaissances" cruciales qui interviennent au cours des diverses activités de l'entreprise, et ce afin d'améliorer les performances de l'entreprise. Néanmoins, derrière cet objectif communément visé, les définitions données à la notion

de "connaissance" sont divergentes. Ainsi, [Balmisse 02] parle d'identifier systématiquement les *compétences* et les *savoirs* de l'entreprise, afin d'en avoir une vision globale ; quant à [Ballay 97], il met l'accent sur le besoin de faciliter la circulation des *informations utiles*, ainsi que sur l'intérêt qu'il y a à les remobiliser dans le contexte de l'activité en cours. Au travers de ces deux exemples, la difficulté à convenir d'une définition précise et unique pour le terme *connaissance* dans le contexte de la gestion des connaissances apparaît donc clairement ; nous reviendrons par la suite sur les travaux visant à caractériser les connaissances.

[Grundstein 00] place la *capitalisation des connaissances* comme le but de tout un chacun, en la considérant comme une richesse qu'il faut valoriser et fructifier. De ce fait, la gestion des connaissances est un processus global visant à assurer leur capitalisation, et elle doit donc être menée à tous les niveaux dans l'entreprise afin de mettre en place les conditions favorables à un travail coopératif et à un échange de connaissances entre ses acteurs. Cette approche est à rapprocher de celle d'[Ermine 01] qui parle de *gestion du patrimoine des connaissances* afin de construire un *livre des connaissances*, ensemble de connaissances structurées pour un domaine. Il propose de considérer la problématique de gestion des connaissances dans les entreprises comme un processus portant sur le patrimoine des connaissances, et s'articulant autour de la capitalisation, le partage et la création. Ce dernier point met bien en évidence que le patrimoine des connaissances doit être considéré comme un système ouvert, dans lequel l'apprentissage joue un rôle clé pour son évolution. De ces travaux, il ressort que la *connaissance* occupe une position centrale, mais il apparaît à nouveau que sa caractérisation est difficile. Cette difficulté provient du fait que les travaux de recherche sur la gestion des connaissances font appel à des disciplines différentes (telles que la psychologie, les sciences du management, les sciences organisationnelles, la sociologie, l'ingénierie, les sciences stratégiques...), et qu'il en découle des sensibilités différentes, mais aussi du fait que la notion de connaissance est par nature difficile à cerner.

La capitalisation des connaissances est donc un objectif clé qui s'intègre à un effort plus global de gestion des connaissances. Elle a pour but de faire passer la somme des expériences et connaissances individuelles au niveau d'une mémoire d'entreprise, ou organisationnelle. Ce processus requiert l'explicitation des connaissances au niveau de chaque acteur, ce qui constitue souvent un enjeu de pouvoir dans l'entreprise. De plus, une partie des connaissances reste tacite, ce qui constitue une difficulté fondamentale à laquelle toute méthode de gestion des connaissances est confrontée.

2.1.1 La notion de connaissance

La notion de connaissance est une notion complexe qui est abordée suivant différents points de vue adoptés par les auteurs. La nature de la connaissance positionnée par rapport à son degré d'abstraction ou de généralité vis à vis des notions de données, information, savoir et surtout la capacité à les mettre en œuvre en situation, qui sera appelée compétence. Les rôles que tiennent ces connaissances dans l'entreprise aboutissent à certaines typologies. Enfin, au croisement de ces deux approches analytiques, ce situe le processus dynamique complexe permettant à chacun des acteurs humains, effectuant une tâche collective, de réunir différentes sources de connaissances, d'effectuer un raisonnement qui est lui-même générateur de nouvelles connaissances. Un modèle de ce processus est présenté, mais nous attirons le lecteur sur le fait que toutes les approches introduites se heurtent au problème fondamental de la présence d'une partie tacite dans la connaissance, qui est par nature difficile voire impossible à capturer.

Caractérisation de la nature des connaissances : un consensus parmi certains auteurs ([Tsuchiya 95], [Ermine 96]) se dégage autour du statut de *donnée*, *information*, *connaissance* ou

bien *savoir*. Nous résumons la définition de ces notions par :

- les *données* sont les chiffres, les mots, les images... utilisés durant la réalisation d'une tâche ;
- les *informations* sont le résultat d'un traitement sur les données, suivant une logique d'analyse et de synthèse, qui a été adopté en vue de la réalisation d'une tâche précise ;
- les *connaissances* sont basées sur des informations mises en relation avec un contexte permettant leur utilisation. Il apparaît à ce niveau que les connaissances sont soit explicites (formalisables), soit tacites (qui s'inscrivent dans l'expérience et les habitudes des acteurs). Les connaissances peuvent également être collectives ou individuelles ;
- les *savoirs* sont des connaissances complètement formalisées et applicables dans différents contextes. Ceci correspond habituellement aux théories.

En complément de cette catégorisation, [Prax 00] ajoute la notion de *compétence*, comme "un ensemble de connaissances, de capacités d'actions et de comportements structurés en fonction d'un but et dans un type de situation donnés". Cette définition nous intéresse tout particulièrement (nous la prendrons en considération au chapitre 4) car elle met l'accent sur la valeur opérationnelle de la connaissance ; cela va dans le sens de l'intérêt que nous portons à étudier des systèmes d'assistance capables de mobiliser l'expérience en situation, afin d'augmenter les capacités d'action (les *compétences*) de l'acteur humain.

Les différents types de connaissance dans l'entreprise : de nombreux travaux ont été menés pour caractériser les différents types de connaissance qui sont impliqués dans les entreprises. Brièvement, sans l'ambition d'être exhaustif, les critères utilisés pour les distinguer se concentrent sur leur nature par rapport au sujet connaissant, les objets visés et la forme d'expression de la connaissance. Ainsi, il est distingué par [Grundstein 00] :

- pour la nature, les connaissances *individuelles* / *collectives*, et de façon complémentaire, les connaissances *tacites* / *explicites* ;
- pour l'objet, les connaissances *techniques* / *managériales*, qui peuvent elles-mêmes faire l'objet d'une caractérisation plus fine. Par exemple, dans le cadre de la conception abordée par la suite , on parle de modèle physique ou structurel correspondant à différentes logiques d'analyse d'un produit ;
- pour la forme, les connaissances *déclaratives* ou *procédurales*, représentant respectivement : des faits et des situations permettant une application avec une certaine abstraction du contexte d'utilisation, ou inversement, des règles et des conditions d'exécution d'une tâche, ce qui est par nature fonction du contexte.

Concernant la nature de la connaissance, nous allons nous intéresser plus particulièrement à son caractère *explicite* ou *tacite*. Parmi différents auteurs s'étant intéressés à cet aspect, [Grundstein 00] précise cette différentiation pour les connaissances de l'entreprise, en parlant respectivement de *savoir* et de *savoir-faire*.

Dans cette approche, les *savoirs de l'entreprise* :

- se caractérisent par leur formalisation et spécialisation ;
- ils portent sur des données, des procédures, des modèles, des algorithmes, des documents d'analyse et de synthèse... ;
- ils sont hétérogènes, incomplets ou redondants ; ils sont marqués par les circonstances de leur mobilisation ; ils n'expriment pas toutes les informations de leur mobilisation ;
- ce sont des connaissances réparties.

Ils sont complètés par les *savoirs-faire de l'entreprise* qui rendent leur mobilisation efficiente :

- ils peuvent être partiellement explicitables, mais restent difficilement adaptatifs ;
- ils portent par exemple sur le contexte décisionnel, sur des habilités, des tours de main, des secrets de métier, des routines ;
- ils sont acquis par la pratique et peuvent faire l'objet d'un apprentissage collectif selon différents modes ;
- ce sont des connaissances localisées.

Toutes ces connaissances sont représentatives de l'expérience et de la culture de l'entreprise. [Grundstein 00] ajoute quatre critères sur ces connaissances, qui portent sur : la *qualité*, la *profondeur*, l'*étendue* et la *stabilité*. Ainsi, intuitivement, cela permet de prendre respectivement en considération, le caractère certain ou incertain d'une connaissance, le fait que les connaissances portent sur des aspects superficiels ou fondamentaux, qu'ils sont de nature spécialisée ou de sens commun, et enfin, que ces connaissances sont stables ou dynamiques, i.e., en cours de construction.

2.1.2 Le cycle de création et de partage de la connaissance

D'autres travaux se sont intéressé à la dynamique de création de connaissance, avec notamment ceux de [Nonaka 95] qui sont à l'origine d'un modèle faisant référence dans le domaine. Ce modèle est basé sur un cycle de conversion de la connaissance, réalisé par chacun des acteurs de l'entreprise. Ainsi, les savoirs sont vus comme le résultat d'une suite d'opérations, qui engendre le passage alternatif de la connaissance entre un statut tacite et un statut explicite, au fur et à mesure de sa mobilisation. Ce passage résulte des opérations de collecte d'information (*combinaison*) par un acteur pour réaliser sa tâche et cela passe nécessairement par une *internalisation*. Le partage de connaissance entre acteurs est souvent informel, il correspond à la *socialisation* et l'*externalisation* et représente une formalisation et une dissémination de la connaissance. La figure 2.1 illustre ce cycle.

L'objectif de la gestion des connaissances peut être défini par le renforcement voire l'instrumentalisation de ce processus de création de la connaissance, afin de permettre aux connaissances un passage d'un état non formalisé et privé, à un état formalisé et disséminé. Cette approche nous permet de souligner le caractère dynamique et collectif de la création de la connaissance.

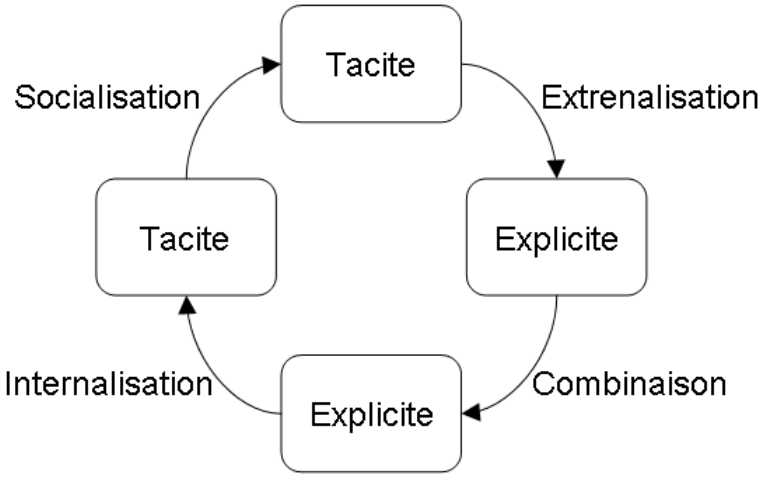


FIG. 2.1 – Le cycle de création de la connaissance selon [Nonaka 95]

2.2 Le cas de la conception intégrée

Cette section a pour objectif de présenter l'activité de conception intégrée, que l'on peut définir par l'intégration des différents métiers (calcul, simulation, dimensionnement, essais) dans le processus de conception. Le projet OSCAR s'est particulièrement intéressé à l'intégration du métier "calcul". Nous allons tout d'abord présenter les enjeux industriels de cette approche. Puis nous aborderons la problématique liée à l'utilisation de la simulation numérique durant la conception. Enfin, nous présenterons le modèle SG3C, qui est à la base du projet OSCAR et qui constitue donc un important point de départ de la thèse présentée dans ce mémoire.

2.2.1 Présentation

La maîtrise de la simulation numérique constitue un facteur clé de compétitivité des entreprises industrielles. Elle contribue simultanément à la qualité des produits commercialisés mais aussi largement au raccourcissement des cycles de développement, qui est conditionnée par la pertinence des quelques prototypes physiques strictement nécessaires. Dans tous les cas, elle est un élément de la *maquette numérique* dont la problématique ne peut être réduite au partage et à l'intégration de modèles géométriques 3D hétérogènes.

Cependant, la multiplicité des simulations numériques dans le processus de développement des produits pose des problèmes de traçabilité et de cohérence des simulations réalisées tout au long d'un projet. D'autre part, la coopération entre "monde des études" et "monde du calcul" est souvent mal assurée. Ainsi, il est souvent observé que la réutilisation des modèles ou des démarches de calcul est difficile au sein d'un même projet ou à travers les différents projets d'une entreprise, tout comme l'identification des connaissances générées par les simulations.

Le besoin des industriels est d'avoir une lisibilité fonctionnelle sur la masse d'informations et de connaissances issue des différents outils de simulation utilisés. Ainsi les questions suivantes se posent : comment stocker les informations de la simulation ? Comment les retrouver efficacement ? Comment gérer l'utilisation de la simulation au sein du processus de conception ? Comment améliorer la communication entre concepteurs et spécialistes ? Comment favoriser le partage d'informations et de connaissances issues de la simulation dans un projet et entre projets ?

D'autre part, le recentrage des grands groupes sur leur cœur de métier les conduit à sous-

traiter aux PME des parts importantes de la conception de leurs produits. Cela entraîne une délocalisation de leur savoir-faire, et les échanges de type client-fournisseur évoluent vers une ingénierie répartie et collaborative ayant pour but d'assurer les fonctions du produit au meilleur rapport coûts/qualité/délais. De cette tendance, il résulte une montée en compétences de la PME, qui devient responsable de la conception de fonctions entières sur le produit. Le partage d'information "produits" et de connaissances sur les processus associés devient alors un vecteur fondamental pour cette collaboration, qui se répercute au sein même de l'activité de conception. Cette évolution conduit de nombreux grands groupes industriels à partager leurs connaissances et savoir-faire avec ces PME partenaires. Ces dernières sont par conséquent au centre des stratégies de Maquette Numérique des grands groupes. Toutefois, il apparaît que seules les PME aptes à franchir ce pas, tant au niveau technologique que humain, peuvent faire partie des partenariats avec ces grands groupes industriels. Faciliter cette collaboration fait partie intégrante des objectifs du projet OSCAR.

2.2.2 La simulation durant la conception

La simulation numérique en bureau d'étude (BE) est aujourd'hui largement utilisée chez *Schneider Electric*, tout comme chez certains autres industriels, tant au niveau des concepteurs que des spécialistes. De nombreux outils existent et apportent des réponses tant sur le comportement (mécanique, thermique, magnétique, etc) des produits en cours de conception que sur les processus industriels employés pour leur fabrication (usinage, rhéologie, assemblage, etc). Tous ces produits permettent aux acteurs du BE de produire des informations et des connaissances qui viennent enrichir le modèle produit.

La simulation numérique est utilisée dans quatres types de contexte :

- aide aux meilleurs choix de conception ;
- validation de solutions conçues au regard d'une spécification ;
- compréhension de phénomènes techniques mal maîtrisés ;
- ajustement et recalage de modèles en vue d'essais.

Dans les nombreux travaux menés sur l'intégration de la simulation numérique en conception, il apparaît que l'aspect de loin le plus traité est le passage des données de conception vers les outils de simulation numérique. L'idéalisation et la discrétisation automatique du modèle de conception en un modèle spécifique au calcul est une approche couramment adoptée ; celle-ci nécessite néanmoins d'ajouter des informations d'analyse (par exemple sur les matériaux impliqués) en complément des caractéristiques géométriques. Ces approches ont l'inconvénient de se concentrer principalement sur le passage des informations dans une seule direction, de la conception à la simulation, alors qu'un retour d'information dans l'autre sens est d'un intérêt certain. D'autre part cette volonté d'automatisation pose certains problèmes ; en effet, afin d'améliorer le temps de développement d'un produit, une réorganisation du processus de conception est nécessaire, ainsi que de faire participer les concepteurs dans les activités de calcul. Il en résulte qu'on ne peut automatiser l'ensemble des tâches de simulation, en mettant l'acteur en dehors de ce processus. Bien au contraire, ce dernier est au centre de l'activité de calcul, et il doit par conséquent disposer de connaissances dépendant de son activité pour agir efficacement dans ce processus.

2.2.3 Le modèle SG3C

Le projet SG3C correspond à une collaboration entre *Schneider Electric* et les laboratoires 3S et Gilco ayant pour objectifs d'apporter des réponses concrètes pour des besoins industriels, tant au niveau de l'organisation de l'entreprise, des méthodes de travail que d'outils supports.

Cette approche a la particularité de se concentrer sur les besoins des acteurs, chargés de faire une analyse experte permettant le passage non automatique des modèles géométriques pour la conception aux modèles pour le calcul. Elle formalise une démarche pour l'utilisation du calcul numérique dans le cadre de la conception industrielle, et laisse la place à la capitalisation des connaissances impliquées en introduisant le concept de *cas d'école*. Elle a également pour objectif de favoriser la coopération entre les concepteurs et les spécialistes du calcul.

Structuration du processus : l'événementiel

[Troussier 99] propose une formalisation du processus de simulation numérique qui est basée sur une structuration en cycles de calcul, où chaque cycle est composé d'étapes successives regroupant les informations spécifiques à ces dernières en vue du calcul. Ces étapes, au nombre de six et spécifiques à cette activité, sont : le *Modèle de conception*, le *But de simulation*, le *Modèle Mécanique*, le *Modèle de Simulation*, les *Résultats* et la *Conclusion*. La figure 2.2 donne un exemple concret de l'application de cette démarche descriptive, où apparaît l'enchaînement chronologique entre les différentes étapes impliquant de fait un passage de connaissances entre les étapes d'un cycle, tout comme entre deux cycles consécutifs.

Avec cette méthode, un processus de simulation est représenté par une succession de cycles. Il est également possible de réaliser des cycles en parallèle, ceux-ci s'effectuant généralement afin de comparer des approches alternatives ; de tels cycles ont un but de simulation commun et une conclusion commune, exprimant le résultat de la comparaison. Cette représentation sous forme d'événementiel est conviviale et suffisamment fonctionnelle pour permettre aux acteurs de naviguer au travers des cycles et accéder à leurs étapes.

Le cas d'école

Le concept de *cas d'école* naît du besoin de permettre la réutilisation d'un ensemble de calculs qui sont estimés suffisamment génériques pour pouvoir être appliqués dans d'autres situations. Cette approche constitue une forme de capitalisation des connaissances produites lors d'un calcul numérique. Notons que la réalisation de cette synthèse est une activité propre à chaque entreprise. Par exemple, chez *Renault*, un service de Calcul Avancé est chargé de synthétiser l'expérience de calcul, de standardiser des méthodes et d'anticiper des besoins de calcul.

Pour illustrer le concept de cas d'école introduit dans SG3C, nous allons donner un exemple. Soit un calcul de flèche sur une poutre donnant pour résultat une flèche de 1 mm ; cela aboutit à la conclusion que le modèle poutre choisi est inadapté. Soit, à l'opposé, une règle de calcul qui exprime que le modèle poutre est inadapté dès lors que le rapport $longueur/hauteur$ est inférieur à 6. Le cas d'école est une connaissance intermédiaire, qui s'exprimerait par exemple par : l'erreur due au modèle poutre diminue quand l'élancement augmente.

De cette manière, la notion de cas d'école peut être vue comme une connaissance locale, qui est plus facile à formaliser qu'une règle, mais qui demande un effort supplémentaire par rapport à l'exemple. Il est important de noter que le cas d'école ne peut être complètement séparé du contexte dans lequel il est apparu.

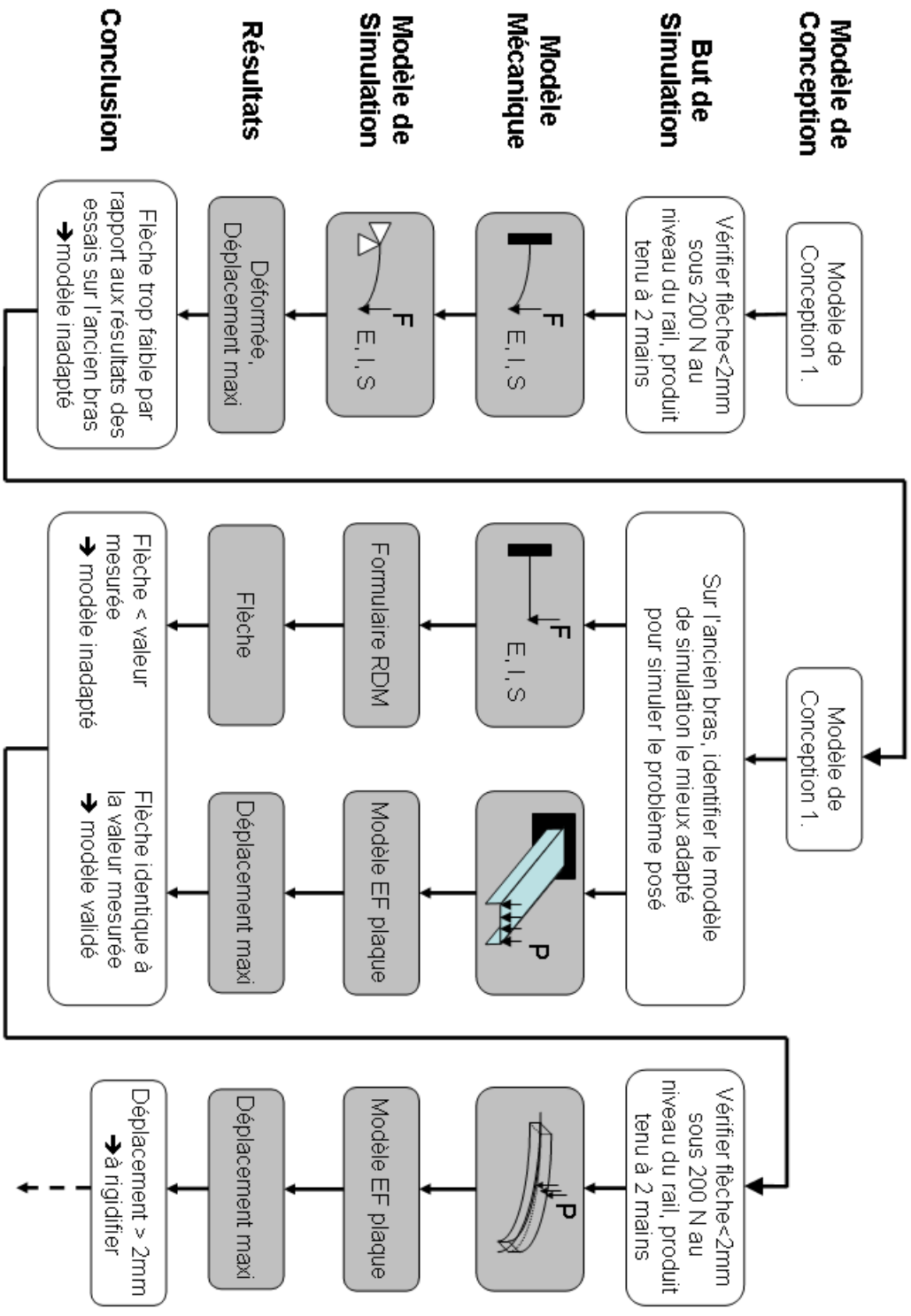


FIG. 2.2 – L'événementiel SG3C : une description chronologique fondée sur la structuration de [Troussier 99]

MemSim et SimManager : deux applications support

MemSim a été développé par [Troussier 99] dans le cadre du projet SG3C, pour supporter la gestion documentaire liée à l'événementiel proposé, et cette application a fait l'objet d'un retour d'expérience. *PCO Technologies* a développé une maquette, nommée SimManager, reprenant les principes de décomposition de l'activité du projet SG3C. Cette maquette illustrait concrètement les besoins de navigation dans les cycles d'activité, afin de permettre aux acteurs de consulter les situations enregistrées. Comme il n'était pas possible pour nous d'utiliser directement cette maquette, en raison de ses nombreuses spécificités, nous avons choisi d'illustrer simplement ce type d'application à base documentaire par un démonstrateur, c'est ce que nous présenterons en 9.1. Ce démonstrateur illustre les opérations (édition, consultation) qui peuvent être effectuées par les différents acteurs sur des entités documentaires, supports d'une activité collective, afin de réaliser leurs tâches respectives.

2.3 Positionnement

Ce premier chapitre a introduit différents travaux visant à gérer et à capitaliser la connaissance. Cela implique des modifications organisationelles dans l'entreprise, suite à une étude détaillée des connaissances manipulées, et nécessite des outils de support à cet effort. Les travaux sur la caractérisation de la connaissance insistent sur le caractère contextuel de celle-ci. Sa valeur réside dans sa capacité à être réutilisée, et ce en situation, afin de limiter les efforts demandés à l'acteur humain. Nous allons donc plutôt parler d'*expérience* que de connaissance, en s'attachant à sa nature personnelle et co-évolutive. Notre approche est de proposer un assistant, adossé à une application telle que MemSim ou SimManager, et qui favorise la réutilisation en contexte de l'expérience, afin d'étendre les "compétences" des acteurs.

Dans ce sens, le chapitre suivant va discuter des problématique liées à l'utilisation de "systèmes intelligents", et présentera le paradigme de RàPC comme une approche alternative. Le chapitre 4 va quand à lui traiter de la notion d'expérience individuelle, en présentant un modèle pour sa réutilisation.

Chapitre 3

Des systèmes experts aux systèmes d'assistance

Sommaire

Ce chapitre traite dans un premier temps des problématiques fondamentales liées à la création de "systèmes intelligents". Les travaux sur les systèmes à base de règles sont discutés, et il est argumenté l'intérêt de recourir à une approche alternative que constitue le RàPC. Ce paradigme est alors présenté, en faisant référence de façon non exhaustive à différents travaux menés sur ce thème, afin de dresser un tableau des dimensions de recherche déjà étudiées et de leurs limites.

3.1 Motivations et difficultés premières des applications de l'intelligence artificielle

Un des premiers objectifs de l'IA est de concevoir des systèmes capables d'acquérir et d'utiliser automatiquement les connaissances propres à un domaine d'application. Ce modèle dit de *système expert* tente de s'approcher des résultats que fournirait un expert humain, voire même de les dépasser. Néanmoins, plusieurs difficultés fondamentales découlent de cette approche selon les objectifs visés pour le système.

L'acquisition des connaissances Un premier problème, d'ordre social, apparaît lors du nécessaire échange de connaissances avec l'expert. Ce dernier peut craindre d'être remplacé par la machine, et donc se montrer réticent à divulger ce qui le rend important pour l'entreprise. Mais avant tout, se pose un problème cognitif : de nombreuses connaissances d'un expert qui doivent être acquises sont des connaissances *tacites*, issues soient d'un apprentissage explicite qui a été complètement approprié, soient d'une acquisition elle-même tacite. [Nonaka 95] appellent cela la "socialisation".

La représentation des connaissances Les langages employés dans les systèmes experts sont conçus pour automatiser le raisonnement, mais ne sont pas nécessairement adaptés à la représentation des connaissances pour un certain domaine d'expertise. Pour réduire cet écart, [Euzenat 95; Pachet 97] proposent des langages de représentation des connaissances par objets, ou encore [Kifer 95; Napoli 97] qui intègrent des éléments de l'approche objet à des langages logiques. D'un autre côté, [Sowa 99] propose les *graphes conceptuels* qui offrent une formalisation des connaissances sous la forme de réseaux sémantiques, et permettent une représentation graphique.

L'exploitation des connaissances Pour les nouveaux langages de représentation de connaissance, l'enjeu ne porte pas seulement sur leur capacité à représenter les connaissances de l'expert ou bien leur faciliter d'appropriation, mais aussi sur les mécanismes permettant de raisonner sur ces descriptions. Ces mécanismes doivent présenter les propriétés de *complétude* et de *correction*, tout en ayant une complexité algorithmique qui permette leur utilisation. Pour les logiques de description, le compromis entre expressivité et efficacité a été étudié dans [Napoli 97], il est possible d'évaluer la complexité des algorithmes d'inférence introduits pour permettre la mise en œuvre des opérateurs nécessaires au raisonnement.

L'ambition inférentielle la plus forte est représentée par les "Systèmes à base de connaissances" définis par [Truong 86]. Ces systèmes ont été abondamment étudiés au travers de différentes techniques de l'IA, qui ont mené à de beaux succès mais aussi à beaucoup de limites dans leur application à des situations réelles, dynamiques par définition. De nombreux efforts sont menés pour prendre en compte formellement cette dynamique dans les langages de représentation utilisés pour décrire les connaissances, tout comme dans les algorithmes devant fournir une interprétation convenable lors de l'exploitation. Des travaux actifs sont menés pour prendre en compte les différentes modalités dans lesquelles les connaissances sont manipulées : c'est dans ce sens que les logiques modales sont introduites afin d'ajouter aux descriptions et aux mécanismes qui permettent de les exploiter des dimensions de temps, de l'espace, du point de vue, du contexte, de la précision, de la certitude, de l'incertitude, de la croyance, etc... Dans tous les cas, l'effort d'ingénierie de la connaissance est d'autant plus grand que l'expressivité est grande, car il faut traiter chaque modalité. C'est dans cette problématique que l'*ingénierie des connaissances* s'est développée, afin de décrire la connaissance avec un degré d'expressivité adapté.

Face à ces difficultés fondamentales, il apparaît alors un changement d'ambition que constitue le paradigme de RàPC, présenté ci-dessous. Dans cette approche, il n'est plus cherché à construire un expert capable de répondre de façon autonome à une question bien posée, mais plutôt un assistant capable d'aider l'utilisateur à répondre à des questions complexes et souvent mal posées. L'ambition inférentielle est beaucoup moins forte, l'utilisateur peut tenir une place à part entière dans le processus de raisonnement.

3.2 Le raisonnement à partir de cas

3.2.1 Fondements cognitifs

Il apparaît à la fin des années 70 en psychologie cognitive que la mémoire *épisodique* tient un rôle prépondérant par rapport à la mémoire *sémantique*, ce qui se caractérise dans la pratique par le fait que le sujet manipule plus aisément des connaissances contextualisées, acquises sous forme d'*expériences* (des souvenirs), que des connaissances abstraites telles que des règles. Alors

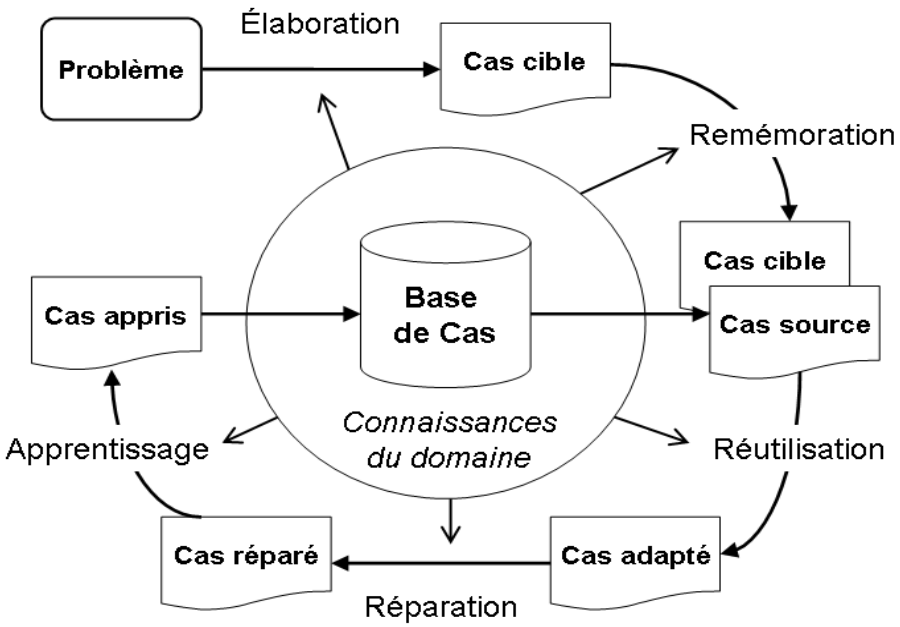


FIG. 3.1 – Le cycle du raisonnement à partir de cas, d'après les principes de [Aamodt 94] et étendu par [Mille 98]

qu'il était considéré que la mémoire sémantique tenait un rôle majeur au cours des mécanismes de raisonnement, [Schank 77] proposent une approche différente selon laquelle un sujet raisonne à l'aide de *scripts* qui guident son comportement durant la réalisation de sa tâche. Les scripts sont tirés de situations antérieures. Nous adoptons couramment cette démarche, par exemple, lorsque nous rédigeons une lettre, nous suivons des démarches distinctes selon qu'elle soit pour notre banquier ou bien pour notre grand-mère ; mais nous adaptons leurs contenus et nous révisons notre script dans certains cas particuliers comme un colis en pièce jointe.

Des mécanismes d'abstraction de l'expérience interviennent également afin d'améliorer son accès. Ceci constitue une problématique fondamentale qui va être à la base de nombreux travaux en vue d'une application informatique. Avec une approche unificatrice, [Whittlesea 87; Rousset 00] proposent des modèles de mémoire fondés sur des systèmes uniques au sein desquels la mémoire sémantique est un résultat émergent de la mémoire épisodique.

Ces travaux ont donné de nombreux modèles qui font l'objet d'implantations informatiques à des fins de validation et de simulation. C'est ainsi que le modèle de [Schank 82] permet d'indexer dans la même structure des expériences concrètes et des connaissances abstraites. C'est sur la base de ce modèle que les premiers systèmes de RàPC sont proposés par [Lebowitz 83; Kolodner 83].

3.2.2 Le cycle du Raisonnement à partir de cas (RàPC)

Ce principe selon lequel il est apparemment plus facile de résoudre un problème à partir de solutions connues pour des problèmes similaires plutôt que de refaire tout le raisonnement, est valable pour nous, c'est pourquoi il est tentant de l'appliquer à l'IA. [Aamodt 94] passent en revue les travaux se rattachant à cette idée, et proposent une formalisation faisant référence dans le domaine pour l'ensemble des mécanismes nécessaires à ce type de raisonnement : le cycle du RàPC (figure 3.1). Appuyé sur une base de connaissance, ce cycle se compose de cinq actions : *élaborer*, *remémorer*, *réutiliser*, *réparer* et *retenir*. Nous allons les détailler par la suite.

Base de connaissances Le raisonnement à partir de cas à pour particularité de s'intéresser à des connaissances *concrètes*, caractéristiques de situations réelles. Elles sont généralement représentées par une partie problème et une partie solution, le tout formant un cas qui est stocké dans une *base de cas* (au centre de la figure 3.1). Les connaissances sur lesquelles reposent ces cas sont liées à l'expérience qu'ils transcrivent, ce qui est difficilement représenté avec des approches à base de règles.

Néanmoins, pour s'exécuter, le RàPC fait appel à d'autres connaissances, et c'est ainsi que [Richter 95] introduit quatre type de "conteneurs de connaissances" qui sont : la base de cas, la mesure de similarité, les mécanismes d'adaptation, et le vocabulaire sur lequel s'appuie la description des trois précédents. Cette position peut s'étendre à toutes les connaissances nécessaires pour effectuer chacune des étapes du cycle. Ces connaissances permettent d'expliquer les cas, les rendant ainsi porteurs d'autres connaissances. Sans elles, seule une approche statistique permettrait d'exploiter la base de cas, bien que cela soit en complète opposition avec un intérêt principal du RàPC qui est de ne pas nécessiter un ensemble de taille suffisamment importante d'exemples d'une situation avant de proposer une solution.

Ainsi, le langage utilisé pour décrire les cas est la première connaissance mobilisée et elle est intrinsèquement liée à la base de cas qui en résulte ; Richter appelle cela le *vocabulaire*. Les différentes étapes du RàPC sont dépendantes de cette description, qui est d'autant meilleure qu'elle les facilite. Une représentation sous la forme attribut-valeur est souvent adoptée, on parle de cas-vecteurs. Mais des représentations plus structurées ont également été proposées [Plaza 95; Smyth 95b].

Les étapes du cycle

Élaborer : cette première étape est absente à l'origine du cycle proposé par [Aamodt 94], elle a été introduite par [Mille 98]. Elle a pour but d'effectuer la transformation du problème considéré dans sa forme "brute", en un cas *cible*. Cette opération ne se réduit pas au simple codage du problème dans le formalisme de représentation des cas : les connaissances du domaine interviennent pour collecter les éléments nécessaires. Cette phase fait déjà l'objet d'une certaine expertise de l'utilisateur d'un système de RàPC, car c'est lui qui détient de façon privilégiée les connaissances pour cette opération. C'est sur cette étape que se concentre le système que nous présenterons dans la deuxième partie de ce mémoire.

Remémorer : le but de cette étape est de retrouver, dans la base de cas, au moins un cas réutilisable pour résoudre le cas cible. Les cas remémorés sont appelés cas *sources*. Cette étape a fait l'objet de nombreuses études pour différents types d'applications. Une classification des différentes mesures de similarité est proposée par [Bouchon-Meunier 96], celle-ci est basée sur les travaux en psychologie de [Tversky 77].

Pour évaluer une mesure de similarité, trois critères sont à considérer : la pertinence, l'efficacité d'exécution et la réutilisabilité des cas obtenus.
La pertinence a tout d'abord été considérée au regard des traits de surface des cas par [Chi 81]. L'utilisation de techniques d'apprentissage automatique sur la base de cas pour obtenir une mesure de similarité a été étudiée par [Sebag 93], et la sélection de mesures de similarité selon leur "pouvoir de discrimination" par [Rifqi 00].
L'étape de remémoration est souvent appelée à comparer le cas cible à un nombre relativement important de cas. Une méthode proposée par [Gentner 91] consiste à effectuer un premier filtrage de la base de cas à l'aide d'une mesure approximative mais moins coûteuse, et d'appliquer la

mesure de similarité détaillée sur l'ensemble résultant.
Une condition importante portant sur les cas obtenus à l'aide d'une mesure de similarité est qu'ils puissent être réutilisés dans l'étape suivante du cycle de RàPC. Dans ce sens, [Lieber 98] voient une dualité entre les connaissances de similarité et les connaissances d'adaptation, ou bien encore, [Smyth 95b] qui avec d'autres effectuent une remémoration guidée par les connaissances d'adaptation.

Réutiliser/adapter : cette étape consiste à transformer de manière plus ou moins automatique les éléments de solution du cas source pour apporter une solution au cas cible. Cette opération est également appelée *adaptation*. Les connaissances nécessaires à cette opération sont souvent difficiles à obtenir et à formaliser, c'est une opération qui est parfois entièrement laissée à la charge de l'utilisateur du système. Toutefois, des travaux tentent de formaliser cette phase [Voss 96; Wilke 98] et des types d'adaptation se distinguent. On peut tout d'abord noter le cas de l'adaptation nulle (recopie des valeurs de solution du cas source dans le cas cible) étudié par [Lieber 02]. Ensuite, [Fuchs 01] proposent un algorithme générique permettant d'adapter la solution source au cas cible par transformation plus ou moins profonde. L'approche générative consiste à appliquer au cas cible le cheminement ayant conduit à la solution source, moyennant une description adéquate de ces connaissances. Tout comme pour la remémoration, des travaux se sont intéressés à extraire des connaissances d'adaptation de la base de cas. Par exemple, [Hanney 96] s'intéresse aux techniques d'apprentissage pour extraire des règles d'adaptation, ou encore [Leake 97] qui font de l'adaptation un cycle de RàPC imbriqué.

[Cordier 06] décomposent les "conteneurs de connaissance", impliqués dans la mise en oeuvre d'un système de RàPC, en des connaissances de cas, de domaine, de similarité et d'adaptation. Ces conteneurs se caractérisent à la fois par l'ingénierie nécessaire pour leur mise en œuvre, et par les modalités de leur acquisition, i.e., les techniques d'apprentissage qui peuvent être utilisées. [Cordier 06] se concentrent sur les connaissances de similarité et d'adaptation, car elles sont de première importance, difficiles à acquérir et à modéliser lors de la conception. Ces difficultés motivent l'introduction d'un processus d'apprentissage pour raffiner ces conteneurs de connaissance. Ces auteurs argumentent qu'il faille une remémoration guidée par l'adaptation, ce qui conduit naturellement à imbriquer les conteneurs de connaissance d'adaptation et de similarité. Ils proposent une formalisation de l'adaptation afin de mettre en évidence les différentes unités de connaissance qui doivent être apprises, sous la forme de dépendances et d'influences entre les descripteurs de la partie problème et ceux de la partie solution. Leur approche se caractérise également par le fait qu'elle est *centrée utilisateur*, i.e., l'utilisateur joue un rôle central dans l'accomplissement des étapes d'apprentissage.

Réparer/Réviser : afin d'enregistrer des cas qui soient valides pour une prochaine utilisation, certains systèmes soumettent les cas adaptés à une expérimentation ou bien une simulation donnant lieu à une évaluation, et les cas sont corrigés selon les résultats. On peut citer les travaux de [Hammond 90] qui tiennent compte des connaissances issues de l'évaluation pour corriger une solution testée, ou plus récemment [Cordier 07b] qui s'intéressent à l'acquisition interactive de connaissances d'adaptation, en adoptant une démarche opportuniste, revenant à intégrer un cycle d'adaptation en interaction avec l'expert qui *révise* les solutions proposées.

Retenir : un des avantages du RàPC est qu'un cas résolu, après l'exécution des étapes détaillées ci-dessus, vient s'ajouter à la base de cas afin d'être lui-même réutilisé plus tard. Ce principe permet au RàPC de revendiquer une certaine capture de l'expérience. A cela revient

directement le besoin d'indexer la base de cas de manière la plus adéquate possible. D'autre part, la base de cas ne peut pas croître indéfiniment sans quoi la phase de remémoration serait trop longue. Il convient donc d'effectuer une maintenance de la base de cas, que [Roth-Berghofer 01] définissent comme un processus en parallèle au cycle de RàPC. [Smyth 95a] proposent la notion de *compétence* d'un cas pour assurer le maintient de la base de cas avec un nombre minimal de cas tout en ne dégradant pas la compétence du système.

Le RàPC distribué et collaboratif

Les étapes du cycle de RàPC sont menées pour chaque utilisateur, qui dispose ainsi de sa propre base de cas et de ses propres fonctions de similarité et d'adaptation. La distribution des cas et le partage des fonctions de similarité ont été étudiés par [Plaza 97] dans le cadre d'un système de RàPC homogène partagé par plusieurs acteurs humain. Il apparaît une amélioration de performance du système (pertinence des cas rapprochés et qualité de la mesure de similarité obtenue par coopération), ce qui est corroboré par les résultats sur les "agents interface", présentés au chapitre suivant. Ceci nous intéresse particulièrement dans notre approche présentée dans la deuxième partie de ce mémoire ; nous referons référence à ces travaux dans nos perspectives, c.f., chapitre 10.

3.2.3 Intérêt et limites du RàPC

Pour discuter des avantages et des limites du RàPC, nous allons dans un premier temps l'analyser au regard des problématiques fondamentales de l'IA développées en 3.1. Ensuite, nous introduirons les motivations qui ont conduit, au travers des travaux de l'équipe SILEX, à faire évoluer le RàPC vers un Raisonnement à partir d'Experience Tracée (RàPET).

Pour la capture des connaissances, nous avons insisté sur le fait que de nombreuses connaissances, qui doivent être acquises, sont tacites pour l'expert. Sur ce point, le RàPC présente l'avantage qu'il ne requiert pas une abstraction ou généralisation des compétences de l'expert, mais au contraire il permet de les acquérir sous une forme proche du récit, qui acquiert un statut général ou singulier uniquement en fonction de sa capacité à être réutilisé. Pour les approches plus formelles, cette difficulté a été étudiée du point de vue de la gestion des exceptions au sein de différents systèmes logiques.
D'autre part, dans la pratique, le RàPC montre généralement l'avantage de requérir des connaissances de surface plutôt que d'avoir besoin d'une théorie complète du domaine ; celles-ci sont plus faciles à acquérir et cela peut être réalisé de façon automatique ou semi-automatique. Le RàPC dispose en plus, par principe, de la capacité d'acquérir des connaissances supplémentaires en retenant des cas, bien que cette opération constitue une problématique en soi comme nous avons pu le détailler plus haut.
Les approches formelles ont pour difficulté fondamentale l'acquisition des connaissances du domaine et leur transposition dans le modèle logique qui les caractérise ; le RàPC a, quant à lui, pour difficulté fondamentale le choix d'une représentation qui soit adaptée à l'utilisation qu'on souhaite faire des cas et qui est donc fortement dépendante des différentes phases du cycle.

L'exploitation des connaissances est contrainte par les qualités et la complexité algorithmique des opérations recherchées. Cette difficulté existe pour le RàPC tout comme dans des approches logiques. Néanmoins, le RàPC est fondé sur l'hypothèse communément admise selon laquelle il est plus facile de résoudre un problème en réutilisant la solution connue d'un problème proche que de générer une solution complètement nouvelle. En terme informatique, il convient de connaître

dans quelles situations la complexité du processus de RàPC est plus avantageuse que celle des approches formelles. Empiriquement, les performances du RàPC varient selon la nature du domaine d'application : pour des domaines homogènes (présence de régularités, de symétries), il se montre souvent moins performant, à l'inverse de domaines hétérogènes dans lesquels la remémoration est plus payante. Cette comparaison n'a de sens que tant qu'il existe des connaissances sur les méthodes permettant de générer une nouvelle solution. Dès que cela n'est plus le cas, les approches formelles s'appliquent difficilement ; l'activité de conception intégrée considérée dans le projet OSCAR est un exemple de ce type de situation ; le RàPC peut encore s'utiliser mais les capacités d'adaptation d'un cas seront contraintes par les mêmes limites.

Disposant de connaissances incomplètes, il n'est plus possible pour les systèmes de RàPC d'adapter totalement un cas. Dans cette situation, les systèmes de RàPC sont amenés à intégrer plus encore l'utilisateur dans le processus de raisonnement. D'autre part, les outils informatiques manipulés pour ces tâches sont généralement des systèmes ouverts donnant à l'utilisateur de nombreuses possibilités de combiner leurs fonctionnalités. Pour être efficace, l'accès à l'expérience doit donc idéalement se faire de façon contextuelle ; toutefois, il n'est pas possible de définir a priori quelles seront les situations qui feront l'objet d'une demande, et il est donc impossible de définir une structure de cas appropriée. Ceci conduit l'équipe SILEX à s'intéresser aux *traces d'utilisation*, considérées comme des entités à part entière dans le système, et qui sont laissées par l'utilisateur lors de son interaction avec ce système. Un modèle d'assistant, le modèle MUSETTE, est proposé pour gérer et faciliter la réutilisation de ces traces. Nous reviendrons sur ces motivations et détaillerons les principes du *Raisonnement à partir d'expérience tracée (RàPET)* au chapitre suivant, c.f., 4.4.1.

Chapitre 4

De la personnalisation à l'appropriation

Sommaire

Au chapitre 1, nous avons introduit le besoin d'une assistance personnalisée et proactive pour notre assistant. Ce chapitre présente les travaux sur la personnalisation d'applications à base de *profil*, menés dans le contexte du World Wide Web. Les travaux sur les assistants proactifs personnalisés, que constituent les *agents interface*, sont ensuite présentés et mis en perspective par rapport à la notion d'*aide*.

Dans la perspective de ces travaux, nous nous intéresserons aux *profils d'utilisation* ; cela nous permet d'introduire le RàPET, et le modèle d'assistance Musette. Pour garantir la pertinence des rapprochements effectués, une problématique fondamentale apparaît alors, qui est de disposer de sens partagés entre l'utilisateur et le système ; cette problématique s'illustre particulièrement dans notre système, car aucune description précise des situations devant donner lieu à une aide ne peut être définie a priori. Nous définissons ainsi la problématique centrale de cette thèse, et nous argumentons l'intérêt de s'intéresser aux travaux sur l'émergence de langage. Enfin, nous récapitulons cette partie de référentiel scientifique et positionnons notre travail.

4.1 La notion de Profil

Avec le développement d'internet, d'importantes quantités d'information sont mises à la disposition de l'utilisateur, à tel point que cela constitue un frein à leur utilisation. De nombreux travaux menés au sein de la communauté IA ont abouti à des plateformes pour des systèmes intelligents distribués apportant des réponses pour palier cette surcharge d'information. Parmi ces approches ont peut distinguer les moteurs de recherche personnalisés, les agents logiciels intelligents ou encore les systèmes de recommandation qui sont apparus pour rechercher, trier, classifier, filtrer et partager ces quantités d'information disponible. Ces opérations croisent la problématique de modélisation des préférences de l'utilisateur, celle de la modélisation des ressources disponibles au regard de leur contenu, et celle de la modélisation de structures récurrentes au sein des communautés d'échange. Deux grandes directions ont ainsi été étudiées individuellement et en combinaison l'une avec l'autre : les approches à base de contenu (*content-based*) et celles s'appuyant sur la collaboration dans une communauté (*collaborative-based*). [Delgado 99; Delgado 00] proposent une analyse récapitulative des travaux sur ce thème, en les mettant en relation avec les théories formelles d'apprentissage automatique. En effet, il voit la tâche de filtrage de documents, consistant à séparer ceux qui sont intéressants ou appropriés de ceux qui ne la sont pas, comme une opération de classification de texte personnalisée basée sur des profils d'utilisateurs ; les profils représentent d'une manière plus ou moins complète le concept cible de l'apprentissage : les *préférences de l'utilisateur*. Cet apprentissage peut en principe être réalisé en examinant les exemples positifs et négatifs donnés par l'utilisateur, mais dans la pratique ce dernier ne veut ou ne peut passer trop de temps à cela. Une approche complémentaire a été étudiée pour obtenir d'autres exemples d'apprentissage, ce sont les systèmes de filtrage collaboratif où les exemples additionnels sont considérés comme des *recommandations*. Les recommandations sont extraites sur la base de régularités d'ordre social et d'analogies entre les différents utilisateurs du système.

Dans ces deux approches, il apparaît que l'apprentissage des profils d'utilisateur est une tâche non triviale qui a déjà rassemblé de nombreux chercheurs en IA. Dans la section suivante, nous allons passer en revue les différentes démarches de construction de profil adoptées dans une logique à base de contenu, et nous introduirons les travaux sur le filtrage collaboratif qui sera traité dans le cadre d'autres approches ("agents autonomes"), en 4.2, appliqué à des problématiques connexes.

4.1.1 Approches existantes

Actuellement, les systèmes Internet qui intègrent un modèle des préférences de l'utilisateur le font sur la base de liste d'attribut-valeur et/ou sur la base de règles. La finesse de la description de l'utilisateur conditionne évidemment les actions de personnalisation que le système peut apporter. Cependant, ce besoin du système est très vite confronté au fait que l'utilisateur souhaite passer un temps limité à donner ces informations, surtout quand cela représente un surcroît de travail sans relation directe avec sa tâche en cours. Cette difficulté est accrue lorsque les centres d'intérêt de l'utilisateur évoluent avec le temps : l'acquisition doit être d'autant plus rapide et la maintenance des profils devient difficile. Pour ces raisons, l'initialisation et la maintenance des profils d'utilisateur nécessitent des méthodes, qui influent sur la conception et le développement de fonctions de personnalisation d'un système.

La personnalisation est par essence à l'interface du système et de l'utilisateur et nous présenterons et discuterons en 4.2 des travaux visant à diminuer l'effort de l'utilisateur en automatisant des parties de son action. D'une façon générale, l'acquisition de données provenant de l'utilisateur (pouvant servir entre autres à la construction de profils) s'étend de la saisie manuelle

par l'utilisateur à la reconnaissance automatique par le système, en passant par des procédures semi-automatiques. Si cela est possible, le choix entre les deux extrêmes est évident : en termes d'effort supplémentaire demandé à l'utilisateur, l'automatisation est à privilégier ; cette situation doit toutefois être nuancée car l'utilisateur accorde plus de confiance en une saisie qu'il contrôle complètement plutôt qu'en une procédure automatique, et il reste prêt à y consacrer le temps si cette acquisition est faite dans le contexte de sa tâche. C'est donc plutôt sur la base de ce compromis que la méthode d'acquisition est à déterminer. D'autre part, dans le cas des systèmes de filtrage, la personnalisation est généralement faite à l'aide d'une interface donnant à l'utilisateur le moyen de rentrer explicitement ses préférences ; ces approches ont pour inconvénient d'être sensibles aux erreurs de l'utilisateur dans la définition de son profil, car elle requiert des connaissances sur le domaine de connaissance et sur les actions réalisées par le système.

Nous allons passer en revue différentes catégories d'approches étudiées, qui ont recours aux techniques d'IA et souvent à l'apprentissage automatique, pour apprendre des profils d'utilisateur. Nous nous basons sur l'analyse faite par [Delgado 99] :

- **Apprentissage à l'aide d'exemples** : l'utilisateur doit explicitement donner des exemples d'éléments intéressants ou bien répondre à une série de questions. Après un nombre suffisant d'informations récoltées, le système construit un profil sur la base d'un mécanisme propre de compilation de ces informations. Cette approche d'acquisition a le double avantage d'être facile à prendre en main par l'utilisateur et de fournir des informations contrôlées au système. Elle repose néanmoins, comme cela est le cas pour le RàPC (c.f., 3.2.3), sur une logique de conception (le schéma d'interrogation permettant de construire une série de questions) qui n'a que peu de chance de pouvoir capturer toutes les logiques d'utilisation (les préférences de l'utilisateur). Nous reviendrons en 4.4 sur cette problématique

- **Stéréotypage de profil/d'utilisateur** : dans cette approche, il est fait l'hypothèse qu'il est possible de classifier un utilisateur dans une catégorie, en appliquant un mécanisme à base d'arbre de décision. Ce modèle est simple et il a l'avantage de nécessiter un nombre réduit d'interactions avec l'utilisateur. Mais ses performances résident sur un nombre approprié de catégories, et sur une sélection attentive des caractéristiques utilisées pour la discrimination.

- **Apprentissage par observation** : l'agent assure la création automatique de profil utilisateur en procédant en deux étapes : 1) une observation du comportement de l'utilisateur, et 2) une action adaptée aux besoins de l'utilisateur sur la base de cas antérieurs. La première étape se termine lorsque suffisamment d'informations ont été récoltées par l'agent pour identifier un schéma de comportement et conclure sur les actions à effectuer. Dans un deuxième temps, les informations identifiées sont utilisées pour mettre en relation les besoins de l'utilisateur avec les actions de l'agent. Cette proactivité dans l'acquisition des informations nous permet de parler d'agent et elle a par principe le grand avantage de ne pas requérir de saisie directe par l'utilisateur. Cette approche dispose d'un caractère adaptable car les relations entre les besoins et les actions peuvent évoluer avec les préférences de l'utilisateur. Son principal inconvénient est de nécessiter généralement un temps d'apprentissage long.

- **Filtrage collaboratif** : les systèmes présentés jusqu'ici, peuvent être qualifiés de système de filtrage à base de contenu. Dans ces systèmes, les ressources filtrées et les profils d'uti-

lisateur sont souvent représentés avec le même modèle (par exemple, un vecteur de mots clés), afin de pouvoir appliquer des mesures de similarité représentatives de la pertinence d'une ressource au regard d'un profil.
Une autre approche existe pour représenter les profils d'utilisateur ; elle est basée sur un schéma d'évaluation lui permettant d'effectuer une prédiction statistique de l'évaluation de nouveaux éléments par l'utilisateur. Dans ces systèmes, les utilisateurs partagent leurs descriptions d'intérêt et obtiennent des recommandations sur la base de l'avis des autres utilisateurs portant sur des éléments semblables. Nous reviendrons plus loin sur ces approches, qui s'intègrent dans une autre problématique, celle de la construction de systèmes d'assistance proactifs, appelés *agents interface*.

[Delgado 99] analysent les approches existantes au regard des modèles théoriques d'apprentissage automatique (Mistake-bound model ou Probably Approximately Correct) qui garantissent une convergence en un nombre limité d'itérations. Toutefois cette garantie n'est plus valide dans le cas où l'objet à caractériser est de nature dynamique, ce qui est le cas par essence pour une activité exploratoire voire créative comme la conception/simulation, à laquelle nous nous intéressons. Ils proposent une formalisation du problème d'apprentissage des préférences de l'utilisateur sur la base du *modèle de Co-training* permettant de combiner les informations de contenu et les informations sociales.

4.1.2 Intérêts et difficultés des profils

Les modèles théoriques d'apprentissage automatique à base statistique ont un intérêt certain pour apprendre des comportements réguliers de l'utilisateur, permettant même de les classifier. Néanmoins, ils ne peuvent plus s'appliquer lorsque les situations devant faire l'objet d'une personnalisation sont singulières ; ceci est généralement le cas pour l'activité de conception à laquelle nous nous intéressons.

Nous voyons que les techniques de profilage ne peuvent pas directement être appliquées dans notre cas. Néanmoins, les descriptions à base de contenu nous intéressent car nous nous plaçons dans le cadre d'une activité effectuée en interaction avec une application métier à base documentaire. D'un autre côté, les travaux sur le filtrage collaboratif nous intéressent par nature, le collaboratif étant le cadre de notre travail.

Enfin, comme nous avons pu déjà le mettre en évidence ci-dessus, la création d'agents assistants personnalisés (ici, agents de filtrage, de recommandation) est confrontée à une problématique d'interface homme-machine. En dehors des critères d'ergonomie, dès le développement, le concepteur est confronté à des choix de mode d'acquisition des informations qui ont finalement une grande incidence sur la facilité d'utilisation du système. Les agents interface abordés ci-dessous sont proposés dans l'optique de réduire cet écart.

4.2 Les agents interface

La section précédente vient de présenter les travaux menés autour de la notion de profil, dans le cadre de la problématique du filtrage d'information. Cette problématique va de paire avec la présence toujours croissante des outils informatiques dans les activités quotidiennes, et le nombre toujours croissant d'utilisateurs ayant une formation réduite, entraînent certaines limitations dans l'efficacité d'utilisation de ces outils.

Pour l'activité de recherche d'information sur internet, dans laquelle chacun d'entre nous a un statut de novice lorsqu'il entame l'exploration d'une nouvelle thématique, [Lieberman 95]

propose le système désormais bien connu : *Letizia*. Celui-ci est un "agent autonome", embarquant une automatisation de tâches routinières et doté d'une certaine proactivité exploratoire. Il joue le rôle de support pour l'"assistance" à l'utilisateur dans sa tâche d'exploration documentaire. Nous reviendrons dans la section suivante sur la notion d'*assistance* et de façon plus générale sur celle de l'*aide*, mais avant cela nous allons détailler celle d'"agent autonome", conçu dans l'optique d'apporter une aide.

La métaphore traditionnelle basée sur la manipulation directe, et s'effectuant souvent par le biais d'opérations élémentaires devant être correctement effectuées, est un frein à l'efficacité des utilisateurs. Pour palier cela, [Maes 94] fait partie de ceux qui pensent que les techniques d'IA, et plus particulièrement les "agents autonomes", peuvent être employés pour introduire des fonctions complémentaires de manipulation, on parle alors de "gestion indirecte". Dans cette perspective, de tels "agents autonomes" seront appelés par la suite *agents interface*. La métaphore correspondante est celle d'un assistant personnalisé collaborant avec l'utilisateur, et basé sur l'apprentissage des habitudes de ce dernier. Dans cette approche, le respect des procédures de l'utilisateur est important : il convient de ne pas s'interposer, et dans l'absolu, les tâches envisageables sont :

- l'exécution d'opérations sous la demande de l'utilisateur,
- sa formation et son perfectionnement,
- le support au travail collaboratif entre utilisateurs,
- et enfin, la centralisation d'information.

Lors de la conception de tels agents, deux problèmes principaux sont à considérer :

1. la "compétence" de l'agent, ou : comment se fait l'acquisition des connaissances nécessaires à un agent, pour savoir quand, avec quoi et comment aider l'utilisateur.
2. la "confiance", ou : comment garantir qu'un utilisateur estime confortable de déléguer une tâche à un agent.

Une première approche d'"agents semi-autonomes" a été proposée : l'utilisateur saisit une série de règles pour traiter certaines tâches ; mais cette méthode ne remplit pas de façon satisfaisante le critère de compétence, car elle nécessite de l'utilisateur qu'il identifie le besoin d'un agent, le crée, lui donne des connaissances et les maintienne à jour. La confiance dans l'agent est plus légitime, mais cela fait appel à des compétences de programmation chez l'utilisateur, qui arrive rarement à obtenir un comportement adéquat.
La deuxième approche, appelée "à base de connaissances", consiste à introduire un agent doté d'ontologies spécifiques sur le domaine et sur l'utilisateur. Ainsi l'agent cherche à reconnaître les intentions de l'utilisateur et à contribuer à l'action qui en découle. La garantie de la compétence et de la confiance constitue une difficulté dans cette approche : une grosse quantité de connaissance doit être introduite, spécifique au domaine et à l'application, et donc de faible généricité. La maintenance et la personnalisation de ces connaissances constituent des difficultés supplémentaires. De plus, l'affectation à l'utilisateur d'une interface sophistiquée, qualifiée et autonome ne participe pas à la mise en confiance de l'utilisateur, car cela est source d'un sentiment de perte de contrôle et d' incompréhension, renforcé par une faible personnalisation.
Enfin, il a été proposé de construire des agents interface en introduisant des techniques d'apprentissage automatique. Cela consiste à apprendre le comportement régulier d'un utilisateur, directement ou bien par le biais d'agents homologues plus expérimentés. Pour cela, il est nécessaire d'avoir des quantités suffisantes d'exemples d'un certain comportement. Mais il est permis de considérer que les comportements répétés d'un utilisateur ont des particularités propres à

ce dernier, qui sont distinctes de celles des autres utilisateurs. Cette approche est un moyen de répondre aux attentes de confiance de l'utilisateur, car il apporte une certaine transparence dans son processus d'apprentissage, permettant également à l'utilisateur d'avoir une représentation plus fine de des capacités de l'agent. Cette méthode a l'avantage, par rapport aux deux précédentes, de réduire les efforts de l'utilisateur ainsi que ceux de développement pour la mise en place de l'outil. Son fonctionnement contient de manière intrinsèque la capacité de prendre en compte les particularités et les habitudes de l'utilisateur.

[Lieberman 97] s'intéresse également à des agents en relation directe avec l'interface utilisateur. Ces agents sont "autonomes", i.e., proactifs, en opposition avec l'approche conversationnelle, qui correspond à une prise de contrôle alternée entre les agents et l'utilisateur.
Il argumente en faveur des agents interface, présentés comme des assistants, ayant une activité perceptible et évaluable relativement simplement par son utilisateur. L'autonomie est dans ce cas un plus, allant dans le sens de l'assistance, et dans celui de la reconnaissance par l'utilisateur pour cette fonction.
Les agents interfaces sont considérés comme un moyen important d'élargir les possibilités fonctionnelles des interfaces utilisateur, car par exemple, l'apprentissage et la contextualisation permettent de réduire le nombre d'accès directs aux fonctionnalités, et d'en simplifier l'utilisation.
De ses travaux, [Lieberman 97] énonce les conseils de conception d'agents interface suivants :

- préférer la suggestion plutôt que l'action. Il convient donc de ne pas utiliser les agents interface dans des actions critiques ;
- chercher la collaboration entre agents interface ;
- utiliser les réactions de l'utilisateur via l'interface pour déduire des informations ;
- profiter des moments de non activité de l'utilisateur, car ils offrent une grande disponibilité des ressources aux agents ;
- prendre en compte l'irrégularité des moments d'attention de l'utilisateur ;
- prendre en compte le fait que les mécanismes de raisonnement impliqués par les agents interface sont plus ou moins adaptés aux démarches cognitives des utilisateurs.

Dans la pratique, [Lieberman 98] souligne que pour la majorité des applications devant utiliser des agents interface, il a été nécessaire de répéter les tâches d'implementation depuis le début ou bien de modifier le code d'une application existante, afin de permettre la communication entre les applications et l'agent. A l'inverse, il serait intéressant de pouvoir "attacher" un agent à une application déjà existante, avec un besoin minimal d'informations de programmation. Depuis peu, les applications gèrent de plus souvent des "interfaces de programmeurs" ou encore des languages de script ; mais est ce que cela est suffisant pour réaliser la communication avec les applications ? [Lieberman 98] présente des travaux expérimentaux visant au développement des agents capables de communiquer avec des applications commerciales non modifiées, mais aussi un ensemble d'applications. Pour cela, il introduit une catégorie d'agents, le *ScriptAgent* : un agent avec des fonctions d'apprentissage d'actions basées sur un langage de script, permettant la communication et l'inter-opérabilité entre applications. Ces travaux soulignent les limites qu'ont les applications existantes à être examinées et manipulées par une application extérieure.

Les agents interface collaboratifs

[Lashkari 94] soulignent que les performances des agents interface, basés sur un mécanisme d'apprentissage, sont limitées par un temps d'initialisation, et par la similarité des nouvelles situations avec celles déjà rencontrées. Tout comme cela a été étudié pour le RàPC (c.f., 3.2.2),

une démarche collaborative peut aider à traiter les situations où l'expérience d'un agent est inadaptée, en lui permettant de solliciter celles des autres agents. La possibilité d'accès aux expériences accumulées par les autres agents permet d'éviter le ré-apprentissage. Dans cette approche, un second niveau d'apprentissage est introduit permettant à chaque agent d'établir un niveau de confiance avec ses pairs. Dans le but d'effectuer une meilleure prédiction lors de la rencontre de nouveaux cas, deux modes de collaboration existent :

1. communication en cas d'incompétence : faire appel aux autres agents pour connaître le comportement de leurs utilisateurs dans des situations similaires.
2. communication exploratoire : les agents entreprennent de découvrir d'autres agents dont les capacités de prédiction, en regard du comportement de leurs utilisateurs, sont meilleures que celles des agents courants.

Ces modes de communication sont orthogonaux : pour un agent donné, le premier mode intervient en cas d'incompétence et gère le cas de nouvelles situations en se basant sur l'expérience des agents de confiance ; le second mode se base sur l'expérience passée pour évaluer la confiance dans d'autres agents.
En dehors de son contexte d'application, cette approche adopte une mesure de "confiance" qui prend en compte ces deux modes de communication. Elle est propre à chaque agent pour chacune des situations. Son initialisation peut être aléatoire ou sur recommandation d'un autre agent. Sa variation, qui est bornée à l'intervalle $]0;1]$, est définie par :

$$confiance \leftarrow confiance + \Delta(p,a) \cdot \gamma \cdot \ confiance \cdot \ certitude$$

où : $\Delta(p,a) = +1$ si la prédiction p de l'agent a est correcte, -1 sinon.
γ : coefficient d'apprentissage.
$certitude \in]0,1]$: niveau de certitude de l'agent prédicteur dans sa réponse.

Cette approche nous intéresse dans les perspectives de notre travail. En effet, elle permet la création dynamique d'un réseau de confiance entre agents interface, sur la base de leurs interactions. Nous reviendrons en perspectives, c.f., chapitre 10, sur ces travaux et l'intérêt qu'ils constituent pour guider la formation dynamique de communautés de travail.

4.3 La notion d'*aide*

Dans cette section, nous allons traiter de la notion désignée communément par le terme "assistance", sous ses formes les plus diverses et les typologies qui en résultent. En ce sens, nous quittons un instant les spécificités des tâches de filtrage ou de recherche d'information, bien que celles-ci soient de toute première importance dans notre travail.

D'un point de vue des sciences cognitives, [Gapenne 06] définit brièvement la problématique de l'aide conçue comme "une relation asymétrique et instrumentée, entre une personne ayant un projet d'action (souhaité ou suggéré voire imposé) dont les modalités de réalisation sont ignorées (ou oubliées) et une technologie censée rendre explicite ces modalités, d'une façon telle qu'elles soient appropriables par la personne sollicitant l'aide". Il met également en évidence le fait que "l'une des propriétés importante de la relation d'aide est d'être porteuse d'une certaine forme de reflexivité que le couplage à l'instrument doit rendre possible." Il propose donc de considérer l'aide comme "un aspect particulier de la dynamique d'apprentissage", laquelle se trouve par

nature "porteuse, ou plus précisément annonciatrice, de transformations cognitives et plus généralement phénoménologiques".

Dans ce sens, [Gapenne 06] distingue quatre types pour une telle relation :

1. la *substitution*, qui qualifie toute relation permettant la délégation à un système tierce d'un projet d'action par l'opérateur, laissant à ce dernier la possibilité de s'affranchir de tout ou partie d'une tâche donnée. De telles relations relèvent des problématiques d'automatisation et nécessitent des interfaces dont l'ergonomie vise la simplicité, l'efficacité et la modularité ;

2. la *suppléance*, qui qualifie toute relation s'instaurant entre une personne et son environnement, matériel ou humain, et qui implique l'apparition d'horizons d'actions et d'expériences inédites par rapport à cet environnement. Une technologie est dite de suppléance lorsque son usage modifie le pouvoir d'action de son utilisateur, par essence, cela appelle et génère l'apparition de nouveaux schèmes d'action ;

3. l'*assistance*, qui est présentée comme une relation dont le rôle est "annexe", comparativement aux autres, et ce dans le sens où "[elle n'est] pas cruciale à l'effectivité de l'engagement dans une activité ni à son déroulement". De telles relations ont pour rôle principal de faciliter ou d'améliorer l'utilisation, constituant néanmoins une large famille d'outils hétérogènes ;

4. l'*aide*, qui est positionnée comme une "relation spécifique entre deux agents dont la mise en oeuvre et la dynamique ont pour effet l'appropriation et l'usage d'un schème nouveau pour l'aidé et la mise à l'épreuve d'un parcours didactique pour l'aidant". Et, par conséquent, il est permis de parler de "reflexivité" émancipatrice, mais dont le potentiel émancipateur sous-tend intrinsèquement la possibilité de s'affranchir, à terme, de l'aide.

Enfin, en complément de cette typologie, [Gapenne 06] souligne le besoin de distinguer la notion de *situation d'aide* de celle de *système d'aide*. Nous n'allons pas traiter en détail la suite de cette analyse, ce que nous souhaitons mettre en évidence est la diversité des situations et des moyens qui peuvent faire l'objet et qui permettent d'apporter une aide. Par souci de simplicité et parce que cette typologie n'est pas un consensus reconnu, nous continuerons à désigner, après cette section, l'ensemble sous le terme commun de situation et de système d'*assistance*, en donnant ainsi à l'assistance un sens plus large (mais plus flou), que celui que nous venons de présenter.

Concernant le domaine des artefacts informatiques pour l'assistance, [Egyed-Zsigmond 03] effectuent un état de l'art conséquent sur les approches et outils existants pour les taches liées à l'exploitation des documents numériques. Leur travail s'attache à distinguer les aspects suivants :

- une première distinction est proposée par [Selker 94], qui sépare les systèmes de type *conseiller* et les systèmes de type *assistant*. Dans cette distinction, le *conseiller* donne des informations, propose des solutions, mais n'intervient pas directement dans la tâche. A l'inverse, les *assistants* sont dédiés à l'exécution de tâches répétitives. Notons que cette définition du terme "assistant" est en contradiction avec celle de [Gapenne 06], qui parle-

rait de *substitution* pour une telle relation (d'usage) ;

- les systèmes d'aide *conversants* et *autonomes* sont ensuite respectivement distingués par [Selker 94] et [Lieberman 97]. Les systèmes conversants s'illustrent sur les systèmes d'aide actuels, et sont conversants dans le sens où ils requièrent, de la part de l'utilisateur, l'expression de questions, auxquelles sont associées des expressions logiques, e.g., des requêtes. D'autre part, les systèmes autonomes fonctionnent en tâche de fond et disposent par nature d'une proactivité dans les suggestions ;

- la capacité d'un système à s'enrichir est soulignée par [Lieberman 96] et [Trousse 99]. Certains systèmes disposent de la capacité d'adapter leurs actions, par exemple selon des vues, mais n'effectuent pas nécessairement un enrichissement de leur savoir.
  Les techniques étudiées pour l'enrichissement couvrent tout le spectre allant du manuel à l'automatique. Ces techniques font appel à certaines déjà introduites, e.g., RàPC et construction de profil, ou encore à celles de fouille de données.

[Egyed-Zsigmond 03] traitent ensuite la situation particulière de l'assistance à l'annotation et à la recherche, pour laquelle il met en évidence : 1) des assistances structurées (à base d'ontologie), 2) des assistances au cas par cas (analogique), mais également 3) des approches mixtes. Ils finissent sur un ensemble de travaux mettant en avant la nécessité de prendre en compte les usages pour une assistance analogique.

Dans la suite des travaux de Pattie Maes, que nous avons présentés dans la section précédente, [Wexelblat 99] ont étudié, conçu et expérimenté différents outils de support à la navigation dans des pages web en y incluant l'"histoire de leur utilisation". Ils partent en s'intéressant à la notion de l'"histoire d'utilisation" qui accompagne chaque objet du monde réel, et argumentent que les histoires, dont il reste des "traces" sur les objets que nous rencontrons, guident de façons plus ou moins conscientes nos actions. Par conséquent, ils effectuent l'étude de modèles et d'outils pour doter les objets numériques d'une telle "histoire d'utilisation". Les objets numériques considérés étant les pages web et les utilisations cibles étant celles liées à la tâche de navigation.

[Wexelblat 99] basent leur travail sur une analyse selon les six notions suivantes, qu'ils considèrent comme cruciales dans la construction d'"interfaces dotées d'histoire" :

1. ils empruntent à l'urbanisme et à l'anthropologie sociale, les adjectifs *proxémique* et *distémique*, pour qualifier la "proximité" des relations entre des gens et des espaces cibles. Cette "proximité" est vue comme doublement croissante selon la distance physique et la "distance cognitive" entre ces gens et entre ces espaces cibles. Intuitivement, un espace proxémique est un espace transparent à ses utilisateurs et où les indications qu'il leurs fournit sont intelligibles. Inversement, un espace distémique est opaque à l'utilisateur, par déficit de connaissance ou d'expérience.

2. le caractère *passif* ou *actif*, de l'utilisateur, lors de l'acquisition des éléments d'histoire est discuté. Dans le cadre de l'assistance à la navigation, cette notion est illustrée sur la base des mécanismes existants, i.e., les bookmarks (actif) et les historiques (passif). Les modes passifs sont privilégiés afin d'éviter de solliciter l'utilisateur.

3. la nature stationnaire ou dynamique des éléments d'histoire est ensuite discutée au regard de son rôle dans la construction d'une histoire, mais également au regard leur "durée de vie". Les points et perspectives soulevés rentrent dans le cadre de la problématique de l'étape de *rétention* du cycle de RàPC, c.f., 3.2.2.

4. la notion de "degré d'infiltration" est introduite pour désigner le degré d'imbrication et d'interdépendant entre un objet et ses traces historiques. D'autre part, ils écartent volontairement toute possibilité d'avoir des éléments d'histoires capturés lors de la création d'un nouvel élément d'histoire.

5. concernant l'origine uniquement personnelle ou sociale des éléments d'histoire considérés à un instant donné, en accord avec des travaux antérieurs, il est mis l'accent sur les bénéfices d'une approche collective ; ceci est fait, en premier lieu, pour favoriser les nouveaux arrivants.

6. une catégorisation du "type d'information" que constitue un élément historique est adoptée selon sa réponse aux questions *quoi*, *qui*, *pourquoi*, *comment*. Une catégorisation qui peut convenir dans le cadre de l'assistance à la navigation, mais qui se confronte aux problématiques présentées au chapitre 2.

Sur ces fondements, [Wexelblat 99] présentent trois outils complémentaires : les *cartes*, les *chemins* et les *indications*. Afin d'éviter les problématiques de gestion de données privées, les données utilisateur sont rendues anonymes et fusionnées.

Les cartes construites sont des cartes de trafic entre les pages, afin de mettre les usages en évidence[1]. L'acquisition des cartes est passive pour l'utilisateur, et il apparaît que l'effort d'appropriation des cartes en fait des objets d'interaction plutôt distémiques.

Les chemins représentent les parcours suivis par un utilisateur. Leur acquisition est passive et ils sont représentés avec des objets graphiques standards pour la gestion des arborescences. Apparemment distémique, ces objets d'interaction sont familiers aux utilisateurs et l'appropriation qui est faite est donc quasi-transparente.

Un mécanisme d'annotation automatique est introduit autour du traçage dans chaque page des statistiques concernant les directions de navigation (i.e., choix parmi les liens présents) prises par les utilisateurs. Cet outil est proxémique et local. Bien que d'un rôle annexe, ce mécanisme s'est révélé générateur de nouveaux usages.

Enfin, les indications sont en quelque sorte des liens, ou plutôt une suite de liens, ajoutés par l'utilisateur qui leur attribue des étiquettes en langue naturelle. Elles constituent le moyen privilégié d'échange et de partage entre les utilisateurs ; un moyen qui est actif, proxémique et social. Toutefois, l'utilisation à grande échelle et le partage illimité finissent par poser des difficultés de gestion et de surcharge d'information.

Les expérimentations de ces outils montrent que les utilisateurs arrivent à effectuer les mêmes quantités d'activité avec un effort réduit.

[1] le moteur de recherche *kartoo* (http ://www.kartoo.com/) est un exemple d'utilisation de ce modèle de représentation d'information, où la sémantique des relations semble a priori prendre en compte les liens hypertextes et des similarités portant sur le contenu.

Une approche théorique des modes d'interaction, des modèles et des langages sous-jacents est effectuée dans les travaux menés par l'équipe de Jean-Paul Sansonnet, et ce en adoptant des approches multi-agents. Les travaux de [Valencia 00] étudient les mécanismes de résolution de l'hétérogénéité sémantique entre des agents dialogiques, i.e., des agents dotés d'ontologies propres et agissant individuellement en conséquence mais confrontés à des situations d'interaction ; ces agents sont basés sur des mécanismes d'inférence de nature logique. Les travaux de [Sabouret 02] traitent de la prise en compte du "sens commun" dans la formulation d'expressions logiques, en l'occurrence des requêtes. [Sansonnet 04] effectue une synthèse de ces travaux et traite d'aspects de développement logiciel qui en incombent.

Dans la continuité, avec une approche ergonomique et cognitive, [Leray 07a] étudient la notion d'assistance, telle qu'elle est faite dans les systèmes actuels, et plus particulièrement son inadéquation dans le cadre d'une utilisation novice. Ils introduisent et illustrent, sur des exemples précis, la notion de "fossé sémantique" (pour un utilisateur novice) qui existe dans ces systèmes ; cette notion est en relation directe avec les situations qualifiés de "distémiques", qui ont été abordées auparavant. Ils défendent l'idée que la conception de l'assistance doit être envisagée dès le début de la conception d'un environnement informatique.

En s'intéressant aux situations dialogiques médiées, [Leray 07b] proposent de prendre en compte et de représenter les éléments cognitifs d'un utilisateur novice au cœur du processus de conception logicielle, afin d'augmenter les performances des fonctions d'assistance. Il est ainsi parlé d'assistance "orientée utilisateur", en opposition aux assistances "orientées code" actuelles.

Nous présenterons en 8.2 un dernier travail, qui se base sur une approche conversationnelle et couplée avec un traitement de la langue naturelle. La problématique de ce travail est particulièrement connexe à la notre, et elles relèvent toutes les deux plus généralement de la problématique d'aide.

4.4 Des profils d'utilisateur aux profils d'utilisation

Les techniques de construction de profil, abordées en 4.1, proposent diverses méthodes qui sont intéressantes pour ajuster rapidement un modèle descriptif de l'utilisateur. Toutefois, ces approches font toutes l'hypothèse que le modèle préétabli, introduit lors de la conception d'un système personnalisé afin de représenter l'utilisateur, sera capable de représenter toutes les situations rencontrées par ce dernier. Or, comme cela est mis en évidence par [Mille 06b], dans la pratique, cette hypothèse ne peut être vérifiée, car un système informatique est conçu suivant une *logique de conception* dépendante de l'analyse de l'activité qui a été faite pour le développer, et que l'utilisateur manipule cet environnement informatique avec sa *logique d'utilisation*. Le terme *logique* est utilisé pour désigner la rationalité respectivement adoptée lors de la conception et lors de l'utilisation, et nous voyons bien ici qu'il y a peu de chance pour que ces deux logiques soient confondues. Cet écart entre ces deux logiques est accru par le fait que les environnements informatiques sont ouverts et que l'utilisateur tente toujours d'appliquer sa logique d'utilisation en exploitant les possibilités d'inter-action qui, elles, relèvent de la logique de conception.

De cette manière, l'environnement informatique constitue par lui-même un terrain de confrontation entre les logiques de conception, qui s'expriment par le biais des possibilités d'inter-action mises à disposition de l'utilisateur, et les logiques d'utilisation, qui s'expriment par les interactions effectivement menées par l'utilisateur dans le contexte de sa propre tâche. C'est l'acteur humain qui est chargé d'atteindre ses objectifs, et qui suit donc sa logique d'utilisation confronté aux possibilités d'inter-action proposées par les différents états de l'environnement, ce qui découle de la logique de conception. Dans l'état actuel des choses, seul l'utilisateur peut combler

cet écart. Dans la pratique, cet écart est réduit par de nouvelles versions de l'environnement informatique prenant en compte un certain retour d'expérience ; mais, pour les mêmes raisons, cette évolution ne parviendra pas à représenter a priori toutes les logiques d'utilisation, surtout si l'environnement est enrichi.

L'idée principale de l'approche proposée par [Mille 06b] est de fonder un principe d'accompagnement de l'appropriation d'un environnement informatique pour réaliser une tâche, en donnant à l'utilisateur le moyen de visualiser la suite d'inter-actions qu'il mène, ce qui lui permet d'interpréter comment sa démarche s'inscrit dans la logique de l'environnement. Dans la pratique, l'utilisateur se voit doté d'un dispositif de *réflexion* de ses interactions, ce qui illustre la confrontation que l'utilisateur doit gérer entre sa logique d'utilisation et la logique de conception.

Pour permettre cette réflexion, [Mille 06b] proposent de mettre à disposition de l'utilisateur une trace de ses inter-actions, telle qu'elle peut être acquise par un *agent* de traçage. La terme agent est utilisé ici, car l'acquisition de la trace est faite de manière proactive, sans effort supplémentaire pour l'acteur humain. Pour permettre à l'utilisateur de gérer la confrontation entre logiques d'utilisation et logiques de conception, une telle trace doit être lisible et manipulable symboliquement par l'utilisateur. En inter-agissant avec elle en tant qu'objet informatique particulier, l'utilisateur adoptera également une démarche d'appropriation des objets trace et des symboles la constituant. L'environnement informatique considéré évolue de facto vers un environnement étendu, intégrant la réflexion. Ce nouvel environnement peut lui même être enrichi en permettant l'utilisation de traces de l'environnement initial dans l'environnement étendu. Il s'agit donc bien d'une dynamique de l'utilisation liée à la mise à disposition de traces gérées par l'utilisateur ; ainsi, la connaissance de l'acteur humain sur l'utilisation en situation de l'environnement se construit inter-activement en étant représenté symboliquement à l'utilisateur afin de faciliter sa propre tâche.

4.4.1 Raisonnement à partir d'Expérience Tracée

Comme le présente [Mille 06a], l'expérience est la source d'une façon plus ou moins directe de toute connaissance, et le raisonnement consiste à mobiliser des connaissances existantes pour produire un résultat satisfaisant le but fixé. Ainsi, l'expertise et la pratique sont deux supports d'inscription de l'expérience, en ayant, pour l'expertise, été compilée et explicitée, ou, pour la pratique, figurant implicitement dans des rapports d'activité ; dans tous les cas un raisonnement peut être vu comme étant *à partir de l'expérience*. Dans cette approche, c'est donc le niveau d'explicitation de l'expérience qui ferait la différence, ce qui reste compatible avec les classifications de la connaissance abordées en 2.1. L'inscription de cette expérience sensible est difficile par nature, car elle est incorporée dans l'environnement soutenant la tâche tout comme chez le sujet vivant l'expérience, et seule les traces persistantes laissées dans l'environnement restent des inscriptions d'une connaissance émergente et/ou mobilisée lors d'une nouvelle expérience. Les systèmes permettent généralement de réinjecter les expériences acquises, et favorise (avec l'intervention de l'acteur humain) le processus de génération de nouvelles expériences. Notons que [Bachimont 04] pointe cette notion d'inscription, et propose en argumentant le concept d'ingénierie des inscriptions de la connaissance.

Cette approche laisse la place à un spectre assez large de manières d'inscrire les connaissances à l'aide d'un système informatique, en fonction de l'ambition que l'on donne au mécanismes de calcul réalisant une interprétation afin de donner un sens.

Dans l'approche de [Mille 06a], il est adopté l'idée que l'expérience humaine, temporelle par essence, est mieux représentée par une trace temporelle inscrivant l'observation d'un processus sous-jacent implicite. Le RàPC, présenté en 3.2.2, s'inscrit partiellement dans cette logique car il

s'intéresse à des épisodes de résolution de problème, mais cela relève de l'analyse préliminaire lors de la construction de cas avec les limites que cela comporte (c.f., 3.2.3). Les systèmes de RàPC exploitant des particularités épisodiques temporelles ne sont pas nombreux, et les descripteurs ne sont pas obligatoirement situés les uns par rapport aux autres. De plus, un épisode de résolution de problème est manipulé en dehors du contexte, ou encore des "histoires", dans lesquelles il a été impliqué, ce qui constitue de fait une restriction a priori des connaissances devant "pour toujours" être mobilisées pour définir le problème à traiter ; un cas étant décrit à un niveau de granularité fixe, dans une temporalité déterminée.

[Mille 06a] propose "d'exploiter les traces d'utilisation d'un dispositif informatique comme sources possibles d'inscription de la connaissance mobilisée par un utilisateur dans sa tâche médiée par le dispositif." Ses travaux proposent une théorie permettant de définir précisément la notion de *trace*, un principe de base pour son codage, et une méthode générale permettant de retrouver des épisodes d'utilisation. "Lorsque les traces sont exploitées sur la base de calculs de similarités autorisant un processus d'adaptation d'épisodes repérés comme similaires, alors nous proposons de parler de *Raisonnement à partir de l'expérience tracée* (RàPET) vu comme une évolution généralisante des principes du RàPC".
Ainsi, par analogie avec le cycle du RàPC (figure 3.1), il est proposé un cycle du RàPET (figure 4.1) au cours duquel le système identifie et recherche dynamiquement des épisodes potentiellement utilisables, en effectuant un rapprochement, qui est lié à l'identification, entre la trace en cours de construction et des traces antérieures. Ce sont les traces qui sont mémorisées en tant que conteneurs d'épisodes potentiels pour des tâches non complètement prévues à l'avance, ainsi que les connaissances servant à l'identification des épisodes.

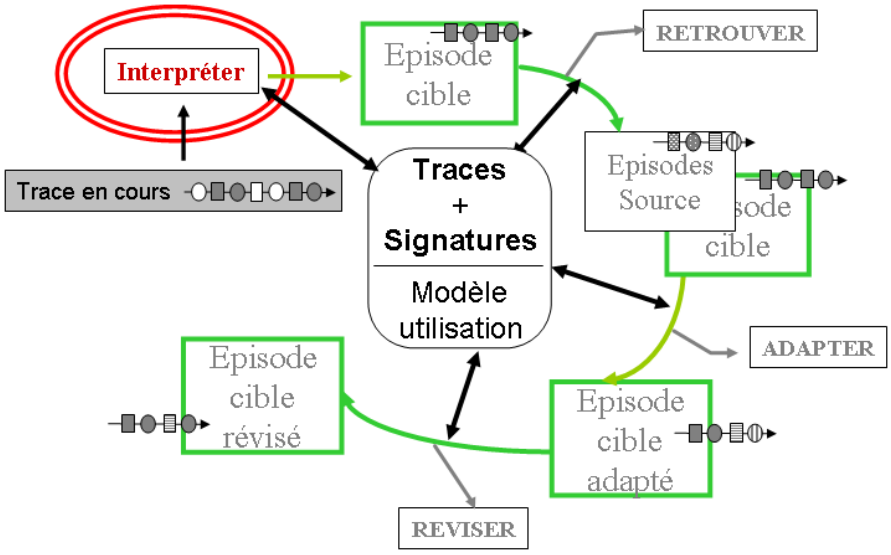


FIG. 4.1 – Cycle du RàPET

La section suivante présente une modélisation de la trace en vue de la réutilisation, que constitue l'approche MUSETTE. Ce modèle ne fait pas l'hypothèse de l'utilisation d'un mécanisme de RàPET, mais est particulièrement adapté pour ce type de réutilisation. C'est l'approche que nous adopterons dans notre travail présenté au chapitre suivant.

4.4.2 Une approche de la trace par les *modèles d'utilisation* : MUSETTE

Le modèle MUSETTE ([Champin 03]) est issu d'une synthèse de travaux déjà menés sur ce thème dans l'équipe SILEX, il est proposé pour répondre à des tâches non complètement définies à l'avance.

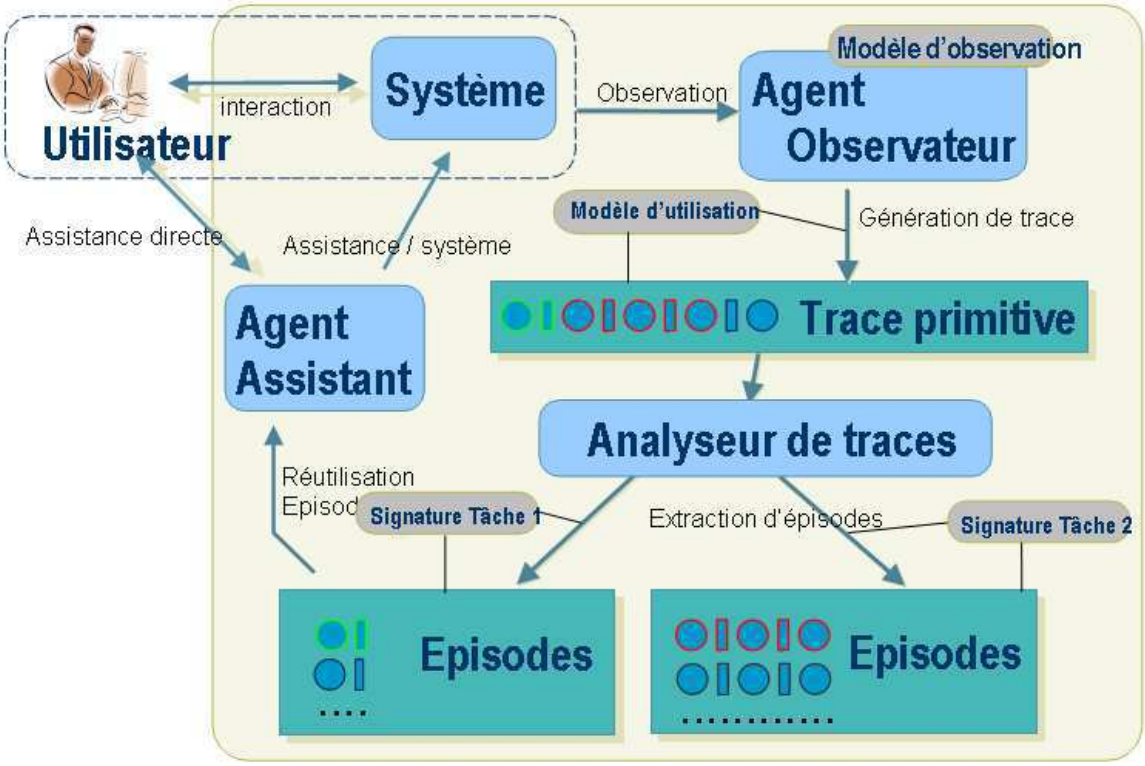


FIG. 4.2 – Modèle MUSETTE : architecture générale d'un système exploitant les traces d'utilisation à des fins d'assistance

La figure 4.2 illustre les principes de l'approche MUSETTE. Un utilisateur est en interaction avec l'environnement informatique, entrainant des modifications de cet environnement. Un *agent observateur* est chargé de décrire les opérations successives de l'utilisateur en construisant une *trace primitive* ; celle-ci est basée sur un *modèle d'utilisation*, une ontologie définissant les *objets d'intérêt* (OI) pour l'utilisateur, i.e., des descriptions des objets de connaissance manipulés par l'utilisateur durant son activité. Nous pouvons inclure à cette ontologie, comme nous le ferons au chapitre 6, des informations relatives à la sémantique des fonctionnalités que l'application observée propose à l'utilisateur, situant explicitement la trace par rapport à l'application. Cette trace primitive peut alors être analysée pour produire des *épisodes d'utilisation* conformes à différentes *signatures expliquées de tâches*, représentant le lien entre un sens d'interprétation de la trace en cours, significatif pour l'utilisateur, et un ensemble de régularités, exprimées dans un certain formalisme (règles, arbres, graphes..), permettant l'identification de la situation.
Ce sont ces épisodes qui constituent les conteneurs d'expérience expliquée exploitables par un *agent facilitateur* qui peut alors les re-présenter à l'utilisateur (usage réflexif des traces) et l'accompagner dans sa tâche en cours. Ainsi, l'environnement cible est enrichi par le fait qu'il donne à l'utilisateur le moyen d'analyser ses propres pratiques, mais l'utilisateur est également informé

sur la perception que le système dans son ensemble (environnement cible plus assistant MUSETTE) a de sa tâche par le biais des signatures de tâches ; ceci constitue un avantage pour faciliter l'appropriation du système par l'utilisateur.

Cette approche résonne avec la problématique des agents interface présentée en 4.2. En effet, elle est fondamentalement centrée sur l'utilisateur, proactive, et est confrontée aux mêmes difficultés que ces derniers. Pour son déploiement, elle est contrainte par la granularité des captures possibles sur l'application cible, ce qui rejoint les problèmes de scriptabilité énoncés par [Lieberman 98].

Principes

Pour construire une trace primitive conforme au point de vue de l'utilisateur, il est nécessaire de capter les interactions en s'appuyant sur les outils de l'interface graphique de l'environnement informatique cible. Cette capture est faite par un automate, l'agent observateur, sous la forme d'un "espionnage" pour acquérir des informations brutes desquelles pourront être distinguées les objets d'intérêt manipulés et désignés par les interactions de l'utilisateur. La trace primitive est une suite d'états et de transitions contenant respectivement des captures de l'état de l'environnement informatique dans une forme considérée comme stable à l'instant de la capture, et des événements déclenchés par l'appel d'une fonctionnalité par l'utilisateur. D'une manière générale, la trace primitive est constituée des objets d'intérêt ainsi repérés et inscrits dans le flux de traçage. Cette trace primitive constitue un premier niveau lisible par l'utilisateur, toutefois l'utilisation directe de cette représentation peut s'avérer difficile : il est aisé d'imaginer une surcharge d'information dans le cas d'un traçage avec une granularité fine, des mécanismes de filtrage et/ou de représentation synthétique peuvent vite s'avérer nécessaires. Le modèle MUSETTE appuie sa construction de trace sur une ontologie de haut niveau, illustrée par la figure 4.3, qui doit être spécialisée lors de son application à un environnement cible. La classe *Relation* est destinée à ajouter des informations entre les OI, au moment de la capture, et sera utilisée dans l'expression des signatures de tâche.

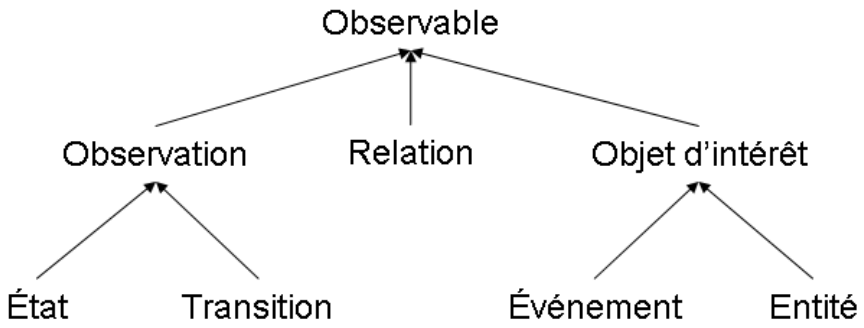


FIG. 4.3 – MUSETTE : ontologie de haut niveau

La figure 4.4 illustre une spécialisation de l'ontologie MUSETTE en définissant les entités et événements qui constituent les objets d'intérêt considérés, et donne schématiquement un extrait de trace qui pourrait être construit en observant l'utilisateur.

Un épisode est une séquence repérée dans la trace en cours sur la base de sa capacité à rendre compte d'interactions réalisées dans le cadre d'une tâche particulière de l'utilisateur, et qui pourrait servir de base à une exploitation par cet utilisateur à des fins de réutilisation ou encore de

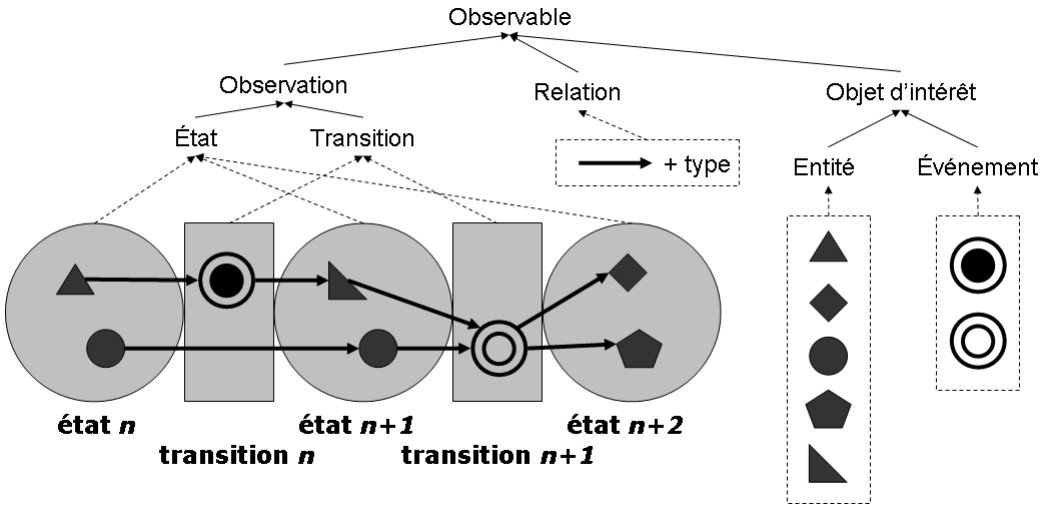


FIG. 4.4 – Exemple de trace MUSETTE et de spécification de l'ontologie de haut niveau (c.f., figure 4.3)

partage d'expérience. L'outil informatique permettant d'identifier un épisode est appelé *signature expliquée de tâche* dans le le modèle MUSETTE. Il s'agit d'un motif composé d'observables, qui est exprimé à l'aide d'un certain formalisme, représentant la façon dont l'utilisateur s'attend à les retrouver ensemble dans le cadre de sa tâche, et il sera utilisé pour l'opération de mise en correspondance approximative du flux de la trace en cours avec des traces antérieures. Ce processus de mise en correspondance approximative peut être composé d'un appariement du graphe exprimé par la signature avec des graphes-séquences contenus dans les traces. La complexité de cet appariement peut varier de manière importante : du simple appariement de surface jusqu'à un appariement fin prenant en compte les valeurs des propriétés des observables. Cet appariement est qualifié d'approximatif car c'est une mesure de similarité qui permet de sélectionner et de classer les épisodes candidats, afin d'exprimer le comportement lié à la tâche considérée par une certaine signature.

4.5 Problématique : disposition de sens partagés

Le modèle MUSETTE qui vient d'être présenté ne fait pas l'hypothèse qu'une situation d'assistance soit définie dès la conception d'un système, mais il reste comme tout système informatique confronté à la problématique d'acquisition des connaissances nécessaires à l'identification d'une situation. Les connaissances permettant cette identification sont contenues dans les signatures de tâche. Pour que les rapprochements effectués par le système soient pertinents et intelligibles pour l'utilisateur, il est indispensable de disposer de signatures de tâches dont la signification est partagée par l'acteur humain et son assistant artificiel.

Dans notre travail présenté en deuxième partie, nous cherchons à construire un agent assistant personnalisé, qui suive les principes d'assistance du modèle MUSETTE avec un mécanisme de RàPET, pour faciliter la tâche d'un utilisateur en lui proposant des rapprochements en contexte avec des situations antérieures. La problématique d'acquisition des signatures de tâche, avec les conditions mentionnées ci-dessus, nous pousse à nous intéresser aux travaux portant sur la construction d'un système de communication entre agents, sans supervision (car elle ne pour-

rait être faite que par l'utilisateur) ni apport initial. Ce rapprochement est à la base de notre contribution, il s'appuie sur l'idée qu'un agent assistant peut bénéficier de ses interactions avec l'utilisateur pour convenir plus ou moins implicitement d'éléments de sens communs, utilisables par le systèmes et adaptés aux besoins de l'utilisateur.

La section suivante présente les travaux d'émergence de langage, qui sont menés avec une approche constructiviste et située.

4.6 Co-construction de sens partagés : l'émergence de langage

La problématique d'avoir à disposition des sens partagés entre l'acteur humain et son agent assistant, nous conduit directement à nous intéresser aux travaux menés sur la dynamique de construction d'un langage entre plusieurs acteurs, sans supervision ni apport initial. Ce que nous appelons langage est, à ce niveau, composé d'abstractions de la réalité et d'éléments de communication perceptibles et reproductibles, permettant *in fine* de désigner communément une certaine abstraction avec un élément de communication partagé ; c'est donc une forme strictement lexicale de langage de communication qui est considérée. [Steels 00] fait une synthèse des travaux menés dans ce domaine, en présentant un langage comme un système adaptatif complexe, dont l'émergence est basée sur des principes fondamentaux issus de la biologie : l'apprentissage par renforcement, l'auto-organisation, le sélectionisme, le co-évolution via le couplage structurel, et la formation de niveaux.

Cette section va présenter cette synthèse. Il est tout d'abord important de considérer qu'un langage émerge dans un contexte situé, chaque agent dispose de capacités de perception et d'action, et qu'il est engagé dans la réalisation d'une tâche nécessitant la communication avec d'autres agents pour être effectuée. Un langage partagé est obtenu en effectuant des itérations de jeux de langage entre deux agents, leur permettant de construire de manière co-évolutive les différents éléments nécessaires à la communication. Plus que les mécanismes de perception et d'action donnés aux agents pour remplir leur tâche, il est important de considérer les mécanismes fondamentaux sur lesquels s'appuient ces différents jeux. Ces premiers travaux mettent bien en évidence une dynamique complexe permettant l'émergence et l'évolution des éléments de base d'un langage, mais l'émergence de langage ayant le pouvoir expressif des langages humains reste encore un mystère ; dans ce sens, l'émergence de grammaire est la principale direction de recherche.

4.6.1 Approche adaptative située

Les travaux synthétisés par [Steels 00] se différencient des travaux antérieures sur les langages dans lesquelles les capacités permettant leur manipulation sont souvent rattachées à des capacités issues de l'évolution, voire propres à l'espèce humaine. Ces approches parviennent à énumérer les raisons pour lesquelles un langage évolue, sans pour autant parvenir à en expliquer les mécanismes. C'est dans cette optique, en précisant les structures cognitives impliquées chez les agents et en définissant les comportements interactifs de chaque agent, qu'est étudiée l'obtention d'un langage par la communauté.

Ainsi, les travaux qui nous intéressent adoptent la même base. Ils impliquent une population d'agents (artificiels), souvent des robots. Les agents s'engagent dans des interactions situées dans un environnement spécifique, une interaction est vue comme une itération de jeux. Chaque agent dispose d'un appareil sensori-moteur, une architecture cognitive et un script déterminant la manière dont il interagit avec les autres agents. Les agents sont plongés dans un environnement, constitué de situations qui sont idéalement ouvertes.

4.6.2 Jeux de langage : principes et mécanismes fondamentaux

La construction d'un système de communication partagé par une communauté d'agents repose sur l'obtention d'un accord entre les acteurs. Cet accord mène chaque agent à construire son répertoire de symboles (par exemple des sons pour le cas de la langue parlée), son répertoire de sens (des conceptualisations de la réalité), et son répertoire de paires symbole-sens (le lexique). Ces répertoires donnent à un agent le moyen de "parler" d'un objet de l'environnement en utilisant des symboles, et d'être correctement compris par un autre agent. L'établissement de ces répertoires est basé sur des itérations de trois jeux de langage, qui sont respectivement : le *jeu d'imitation*, le *jeu de discrimination* et le *jeu de nommage*.

4.6.2.1 Jeu d'imitation

Le travail de [De Boer 97] constitue un bon exemple pour illustrer la manière dont un répertoire de voyelles (constituant le répertoire de symboles) est convenu au sein d'un groupe distribué d'agents, sans qu'un répertoire ne soit prédéfini, et sans supervision. Le cas des voyelles est intéressant car une tendance universelle à produire les systèmes de voyelles des langues humaines a été observée, dès lors que le système producteur de voyelles modèlise le fonctionnement et les contraintes des organes impliqués dans la parole humaine. De façon plus générale, les voyelles sont influencées par les contraintes fonctionnelles et sensori-motrices des acteurs.

Les simulations réalisées se font dans le contexte de la robotique : un agent dispose pour percevoir d'un analyseur acoustique, qui extrait les principales composantes d'un signal, et pour émettre d'un système articulatoire synthétique.

Une itération de jeu d'imitation s'effectue de la manière suivante :

1. Un agent émet aléatoirement un son, qui fait partie de son répertoire ;
2. un autre agent tente de reconnaître le son émis, au regard de son propre répertoire, et reproduit le son qu'il a reconnu ;
3. le premier agent tente de reconnaître le son qui a été reproduit et le compare au son qu'il a initialement émis.

L'itération est un succès si le son finalement reconnu par le premier agent est identique à celui qu'il avait initialement émis, sinon c'est un échec.

En cas de réussite d'une itération, le score associé au son considéré augmente. En cas d'échec, deux situations sont distinguées. Soit le son émis ne présente aucune similitude avec un son déjà existant dans le répertoire, il y est alors ajouté et un moyen de le produire est cherché individuellement. Soit le son émis est proche d'un son déjà existant dans le répertoire mais sa reproduction n'est pas satisfaisante, ce qui traduit une limite dans les capacités de distinction de l'imitateur. Ce dernier intègre ce son à son répertoire et cherche à améliorer sa production. Pour ajouter des sons à leurs répertoire, les agents produisent de nouveaux sons en faisant varier les paramètres de leur système articulatoire ; les signatures acoustiques de ces sons sont ajoutées au répertoire. Enfin, les sons dont les scores sont significativement bas sont éliminés, et les sons suffisamment proches sont fusionnés.

Les expérimentations montrent que les phénomènes suivants apparaissent après une série d'itérations de jeu d'imitation entre les agents :

- un répertoire de sons partagés émerge par auto-organisation ;
- le répertoire s'étend aussi longtemps qu'une contrainte existe ;
- dans le cas des voyelles, celles qui émergent ont les mêmes caractéristiques que les systèmes naturels de voyelles.

Ainsi, en plus de démontrer que l'émergence d'un système partagé de voyelles peut être obtenu de manière distribuée, ces expérimentations illustrent qu'il est aussi possible de capturer les propriétés essentielles des systèmes naturels.

Ce jeu s'appuie sur trois principes :

1. l'*apprentissage par renforcement* constitué par la rétroaction après chaque itération, et utilisé pour renforcer ou inhiber un son selon son succès. Cependant, ce principe ne suffit pas à expliquer la formation d'une solution partagée ; à cela s'ajoute :
2. l'*auto-organisation*, qui apparaît lorsqu'une boucle de rétroaction positive est présente dans un système non-linéaire ouvert. Dans le cas du jeu d'imitation, cette boucle positive apparaît lors du succès de l'utilisation d'un symbole (ici, une voyelle). Ce principe explique l'apparition d'un répertoire partagé, mais n'explique pas pourquoi certains sons apparaissent plutôt que d'autres. Pour cela, un troisième principe est nécessaire :
3. le *sélectionisme*. Les capacités sensori-motrices des agents rendent certains sons plus faciles à reconnaître et à émettre ; ces sons ont donc tendance à être renforcés et donc à survivre dans la population. La même tendance s'applique aux sons créés aléatoirement.

4.6.2.2 Jeu de discrimination

D'autres expérimentations ont montré comment un répertoire de sens peut émerger chez un agent, membre d'une population, suite à une série d'itérations de jeux. Les jeux sont typiquement des jeux de discrimination.

Un agent perçoit une certaine partie de la réalité, sélectionne un des objets présents dans ce contexte et le considère comme sujet. En utilisant ses canaux de perception, l'agent essaie de trouver une catégorisation qui permette de distinguer le sujet des autres objets. L'agent a la possibilité de faire évoluer les descripteurs de ces catégories. Un score est associé à chaque catégorie : les catégories utilisées avec succès sont renforcées, sinon elles sont inhibées. Les catégories avec des scores significativement bas sont supprimées. Les expérimentations montrent la formation de répertoires de catégories stables, qui continuent à s'étendre ou à se contracter lorsque l'environnement évolue. Les répertoires ne sont pas nécessairement identiques entre les agents.

Plus que les mécanismes de catégorisation utilisés, ce qui est important est l'idée d'organiser le processus de formation des répertoires comme une série d'itérations de jeu. Les deux principes fondamentaux du jeu de discrimination sont :

1. l'apprentissage par renforcement, car les catégories distinctives sont renforcées, sinon elles sont inhibées. Cela conditionne leur persistence dans le répertoire.
2. le sélectionisme, car les catégories crées spontanément ou celles issues de l'évolution d'autres catégories sont soumises aux pressions de l'environnement, ainsi qu'à celles de la tâche de perception suivie de la discrimination.

Il n'y a pas d'auto-organisation car chaque agent construit individuellement son propre répertoire. Les expérimentations montrent un raffinement des mécanismes de catégorisation.

4.6.2.3 Jeu de nommage

Enfin, d'autres travaux se sont intéressés à la construction collective de relations symbole-sens dans une population d'agents. Les agents ont une architecture cognitive composées d'un mécanisme de catégorisation et un lexique qui est constitué d'une mémoire associative bi-directionnelle

leur permettant de stocker des paires symbole-sens. Les agents n'ont pas de vue globale, ni d'accès aux lexiques des autres agents.

Une itération de jeu de nommage se déroule de la façon suivante : lorsqu'un agent à un sens à exprimer, il recherche dans son lexique le symbole le plus approprié. Un autre agent, qui est à son écoute, utilise son propre lexique pour retrouver le sens le plus probablement associé. L'itération est un succès lorsque le sens retrouvé par le second agent est compatible avec celui exprimé par le premier agent. La compatibilité s'exprime indirectement au travers des actions qui sont effectuées : une réaction adaptée traduit un sens correctement exprimé. Il y a inhibition bilatérale : les scores des associations utilisées augmentent et ceux des associations concurrentes diminuent. En cas d'échec, les scores des associations impliquées diminuent. Les expérimentations montrent clairement une dynamique selon laquelle une association prend le dessus sur toutes les associations concurrentes, après une phase de compétition.

De la même manière que pour les deux jeux précédents, les principes mis en oeuvre sont importants. On a :

1. l'apprentissage par renforcement, basé sur le succès ou l'échec du jeu dans son ensemble. L'itération est un succès si le deuxième agent parvient à identifier le sens que voulait exprimer le premier agent.
2. l'auto-organisation dûe à la boucle de rétroaction positive entre succès et utilisation. Les agents préfèrent le mot qui a le score le plus élevé pour produire et interpréter, ainsi les mots ayant le plus de succès seront utilisés, ce qui conduira à encore plus de succès.
3. le sélectionisme qui joue aussi un rôle, car les capacités sensori-motrices des agents et les situations qu'ils rencontrent agissent comme une pression sélective.
4. le *couplage structurel*. Ce quatrième principe est largement utilisé en biologie. Il mène à la co-évolution entre répertoires de sens et lexiques. La formation du répertoire de sens est couplée à la formation de lexique de deux manières : de nouvelles catégories peuvent apparaître à tous les instants, qui sont utilisées avec succès pour la discrimination. Ceci conditionne les paires symbole-sens qui vont apparaître. Inversement, le succès du jeu de langage influence le score des catégories utilisées, et le processus de catégorisation s'adapte mieux au langage dans l'environnement des agents.

Les expérimentations montrent que, pour un sens et de multiples symboles candidats à son expression, une relation symbole-sens finit par l'emporter sur toutes les autres relations.

4.6.3 Vers l'émergence de grammaire

[Steels 00] cite plusieurs études qui ont été menées pour tenter d'expliquer comment des langages, avec une complexité grammaticale de l'ordre de celle observée dans les langages humains, peuvent émerger sans supervision ni apport initial. Il en ressort que ces travaux sont confrontés à des questions ouvertes, et que cet effort de recherche n'en est qu'à ses débuts. Toutefois, ces travaux utilisent les mêmes principes que ceux utilisés pour l'émergence d'un lexique : l'apprentissage par renforcement pour exprimer l'adoption par un individu des conventions présentent dans la population, l'auto-organisation due à la boucle de retro-action positive entre l'utilisation et le succès afin de gagner de la cohérence, le sélectionisme lié au contraintes de l'environnement, et la co-évolution entre la syntaxe et la sémantique du fait d'un couplage structurel. Il s'ajoute un phénomène propre aux grammaires : la formation de niveaux, résultant de structures hiérarchiques issues d'opérations de composition. Les niveaux constituent des îlots de stabilité (réutilisables avec succès) au sein de structures plus larges.

Plus récemment, avec une approche constructiviste, [Steels 05] propose un formalisme d'émergence de grammaire, appelé *Fluid Construction Grammar*, pour l'établissement d'une syntaxe partagée par un ensemble d'agents. Nous ne détaillerons pas cette approche à ce niveau, car les mécanismes que nous allons utiliser pour notre contribution n'y font pas référence. Néanmoins, nous reviendrons sur ce formalisme et sur les questions qui s'y rattachent dans les perspectives de notre travail, traitées au chapitre 10.

4.7 Positionnement

Nous avons conclu dans le chapitre 2 sur le besoin de s'intéresser à l'expérience plutôt qu'à une forme synthétisée de connaissance, car nous argumentons que la connaissance est contextualisée et qu'elle trouve sa valeur dans sa capacité à être réutilisée. Dans le but de construire un assistant intelligent, les travaux d'IA sont discutés au chapitre 3 ; cela nous conduit à présenter une approche intermédiaire, le RàPC. Nous détaillons ses principes qui permettant de soutenir un acteur humain dans la résolution d'un problème, sur la base de situations antérieures, i.e, les *cas*, dont une acquisition incrémentale est effectuée. La formalisation a priori du problème à résoudre, et l'utilisation des systèmes de RàPC pour des tâches autres que celles pour lesquelles ils ont été conçus, nous a amené à argumenter qu'il faut étendre le RàPC à des situations non prévues à l'avance.

Afin que notre assistant fournisse des suggestions personnalisées, nous avons passé en revue les travaux sur la personnalisation par profil dans le cadre du WWW (c.f., 4.1). Les agents interface, qui ont été pour beaucoup également appliquées aux problématiques du WWW, sont ensuite discutés pour traiter du besoin de proactivité de notre assistant ; cela nous permet d'identifier les besoins et les difficultés pour leur mise en œuvre. Les agents interface et notre projet d'agent assistant nous conduisent à traiter au sens général des travaux sur les systèmes d'aide. Nous avons argumenté ensuite qu'une grande difficulté, commune à toute application informatique, venait de l'écart entre la logique de conception et celle d'utilisation d'un système, car cela ne permet pas à l'application de répondre correctement à toutes les demandes de l'utilisateur. Les systèmes de RàPC ou de personnalisation à base de profil n'échappent pas à cette problématique. Pour réduire cet écart incontournable, nous nous intéressons à la capture et à la réutilisation en situation de "profils d'utilisation", c'est dans ce sens que nous avons présenté le RàPET et le modèle Musette. De cette approche, il est apparu le besoin d'acquérir des sens partagés par l'utilisateur et son assistant, ce qui s'inscrit dans la problématique plus générale de l'explicitation des connaissances par l'utilisateur. Nous introduisons l'idée principale de notre contribution qui est d'obtenir ces sens partagés sur la base d'un processus de négociation. Cela nous conduit à présenter les travaux sur l'émergence d'un langage entre agents, sans supervision ni apport initial.

Le partie suivante présente les principes de notre agent assistant personnalisé, basé sur un mécanisme de RàPET. Nous allons proposer une transposition des mécanismes d'émergence de langage (de forme lexicale) pour que notre agent assistant puisse acquérir des éléments de sens partagés avec l'utilisateur, ces sens lui servant à identifier une situation d'assistance et à effectuer des rapprochements en conséquence.

Deuxième partie

Faciliter la réutilisation en situation de l'expérience individuelle tracée

Chapitre 5

Principes et Architecture de l'agent assistant

Sommaire

Ce chapitre a pour objectif d'introduire les principes d'*aide* que nous retenons pour notre agent assistant, ainsi que d'introduire les trois comportements fondamentaux de cet agent qui guident intrinsèquement son architecture. Les mécanismes que nous associons à ces comportements vont être décrits dans la suite de cette deuxième partie, en finissant sur la présentation du démonstrateur que nous avons développé pour illustrer nos principes et évaluer leur faisabilité.

La section suivante définit le cadre de travail, en positionnant nos choix fondamentaux par rapport aux aspects soulevés dans la partie précédente. La section 5.2 détaille notre démarche pour apporter de l'aide à l'utilisateur, sur la base de l'expérience acquise et des "indications de sens" qu'il a pu y donner auparavant. Notre méthode de travail sera présentée en 5.3, cela nous permettra de discuter des intérêts et des difficultés de notre approche. Nous conclurons ce chapitre sur un *Guide de Lecture* pour cette deuxième partie, en s'appuyant sur les éléments que nous aurons introduits.

5.1 Cadre de travail et Démarche adoptée pour l'assistance

L'objectif de notre travail est d'étudier la notion d'agent assistant (au sens des agents interface, c.f., 4.2) assurant la capitalisation interactive de l'expérience individuelle et facilitant la réutilisation de cette expérience par celui qui l'a expérimentée. L'assistance à la réutilisation est faite "en contexte" pour des activités non complètement définies à l'avance.

Plutôt que de chercher à généraliser des connaissances complexes et multiples, nous nous attacherons à capturer des descriptions contextualisées des situations rencontrées par l'acteur humain. L'expérience individuelle est captée sous la forme de séquences d'interactions avec un

système informatique, et elle est représentée par des *traces*. Notre approche se place dans la logique du RàPET, présenté en 4.4.1 et proposé pour apporter une réponse aux limites du RàPC discutées en 3.2.3. Notre assistant s'appuie sur le modèle MUSETTE, présenté en 4.4.2, qui base la construction des traces sur une modélisation des objets d'intérêts (potentiels) pour l'utilisateur. L'assistance est possible grace à l'identification d'une signature de tâche. Nous avons argumenté en 4.5 que l'obtention de signatures significatives est une problématique clé du RàPET. Pour répondre à cela, nous proposons de représenter ces signatures à l'aide d'un language émergent, que l'agent assistant utilisera et maintiendra afin de représenter les sens d'interprétation des traces qu'il partage avec l'utilisateur. Cette approche demande à l'agent de disposer d'un mécanisme de négociation du sens des éléments de langage manipulés par l'utilisateur.

Notre agent assistant observe la manipulation d'un système informatique à base documentaire par un acteur, impliqué dans une tâche coopérative. L'activité coopérative que nous considérons à titre d'exemple est la réalisation de la procédure administrative de remboursement des frais de mission dans un laboratoire de recherche universitaire ; en effet, les acteurs de ce processus (le missionnaire, le directeur de laboratoire, les secrétaires des différents services administratifs...) ne tiennent pas les mêmes rôles, n'ont pas toutes les connaissances, et n'ont pas une vision globale du processus. Nous cherchons par là à illustrer simplement le type d'application SG3C proposé par [Troussier 99] et décrit en 2.2.3. Notre description de la procédure administrative suit les principes de décomposition en entités, représentées par des documents composés de divers champs, et qui se succèdent en principe les unes aux autres. Dans la pratique, pour diverses raisons, le déclenchement, l'enchaînement voire la répétition des étapes ne suivent pas l'ordre prédéfini. La trace représente directement les actions sur les documents telles qu'elles ont été effectuées, car nous considérons que cela traduit en soi une certaine expérience en situation.

Suite au contexte et aux besoins énoncés dans le cadre du projet OSCAR, nous considérons dès le départ l'acteur au sein d'une communauté intervenant dans une activité coopérative complexe, i.e., une activité impliquant différents acteurs qui n'ont pas les mêmes rôles et les mêmes compétences et qui n'ont pas une vue d'ensemble sur cette activité.

Nous adoptons une approche descriptive de l'expérience individuelle, car nous considérons qu'elle constitue, de fait, le point de départ à la construction de traces d'expérience collective. Nous discuterons en perspective de la problématique de la construction et du partage d'expérience individuelle et collective, et argumenterons les possibilités de notre approche dans un tel contexte.

Compte tenu de cela, nous démarche peut se résumer à étudier les moyens informatisables pour que notre agent assistant soit en mesure de reconnaître une situation et d'effectuer des rapprochements à partir d'éléments de sens, que nous modélisons par un langage lexical, et de les présenter à l'utilisateur de telle manière que la situation fasse sens à l'utilisateur, à l'instant t.

En fait, l'agent alter ego a acquis ces éléments de sens, avec une certaine confiance, lors d'interactions passées avec le même utilisateur, schématiquement à l'instant $t-1$; le rôle de notre agent assistant est en quelque sorte de colporter à l'instant t le discours tenu à l'instant $t-1$ par un même utilisateur. Notons que cela implique de doter l'agent assistant de proactivité : dans le choix des discours à colporter, dans la recherche d'illustrations analogues à partir des traces d'expérience antérieures, dans la négociation avec l'utilisateur lorsque des ambiguïtés de sens surviennent et dans la synthèse de ces négociations et de nouvelles indications afin de construire et d'utiliser un nouvel état de langage.

Du point de vue des formes d'assistance définies par [Gapenne 06], nous pouvons dire d'emblée que nous ne visons pas directement la *substitution*, i.e., la délégation de tâches systématiques, principalement parce que nous nous intéressons ici qu'à l'identification de situation ; nous reviendrons là-dessus en fin de chapitre, c.f., 5.3. Autrement, nous visons à proprement parler l'*assistance* et l'*aide*, i.e., faciliter la tâche et assurer un transfert de compétence ou encore une "formation", si ce n'est au regard de l'oubli de la part de l'utilisateur. Enfin, nous espérons la *suppléance*, dans le sens où la manipulation à part entière de l'expérience tracée et des éléments (avec leurs propriétés informatiques et les mécanismes proactifs associés) venant la dénoter peut constituer un nouvel espace d'action pour l'utilisateur. Cependant, tous ces aspects de la notion d'*aide médiée* ne serons pas évalués dans la suite ; en effet, n'ayant eu la possibilité d'effectuer des expérimentations réelles de nos principes, nous limiterons leur illustration à un démonstrateur.

Notre travail constitue une situation d'agents dialogiques interagissants, dont les éléments de sens respectivement manipulés peuvent conduire à des hétérogénéités sémantiques, et qui relève donc directement des travaux de [Valencia 00]. Toutefois, il nous a semblé plus facile de capter ce qui peut être réutilisé avec succès que de capter la logique interactionnelle. Cependant, cette logique interactionnelle deumeure de première importance, et notre approche n'est pas contradictoire avec elle, elles pourraient même profiter l'une de l'autre ; c'est pourquoi nous traiterons encore de ces travaux en 8.2.

D'autre part, nous privilégions un moteur d'inférence analogique, i.e., le RàPET, dans la continuité de ce que nous avons présenté au chapitre 3.

Notre approche à base de trace est connexe à celle "riche d'histoire" de [Wexelblat 99]. En effet, dans notre cas, l'expérience tracée peut être vue comme une première dimension "historique" dont la forme (des séquences de captures) exprime un ordre temporel, et les éléments de sens venant la dénoter comme une seconde dimension "historique" d'une autre nature. Nous allons donc situer notre travail par rapports aux six notions déjà présentées en 4.3.

Concernant les notions d'espace d'interaction *proxémique* ou *distémique*, dans notre cas, les mécanismes de suggestion d'expérience et de négociation de sens que nous allons introduire sont d'un caractère *distémique*, car ils s'intègrent dans un ensemble dont l'appropriation n'est pas immédiate ; une formation minimale devrait accompagner son introduction. Toutefois, le coeur de nos mécanismes portent sur des dénotations dont la sémantique est exprimée en langue naturelle ; les travaux de [Wexelblat 99] ont illustré l'intérêt de tels approches (dans le cas des "indications") et le caractère proxémique d'un tel mode d'interaction. Notre acquisition de traces d'interaction se fera avec une approche *passive*, i.e., systématique, à l'inverse de l'interaction avec les éléments de sens qui est *active* par nature ; nous suivons l'idée de privilégier tant que possible les modes passifs, cependant nous illustrerons dans les chapitres suivants les problèmes qui résultent de cette catégorie d'approches. Le "degré d'infiltration" de nos éléments d'histoire (traces et sens de dénotation) dans les objets d'activité est quasi nulle, i.e., les traces sont totalement dissociées des objets d'activité, bien qu'elles ne soient valable que par rapport à eux ; a fortiori, les sens de dénotation sont indépendant des objets d'activité, mais résultent des traces tout en étant des éléments distincts. A propos de la porté personnelle ou sociale de nos "éléments d'histoire", nous adoptons une approche personnelle dans un premier temps, car nous considérons que pouvoir réutiliser (ou plutôt, se faire proactivement aider à réutiliser) sa propre expérience est la première étape avant de chercher le partage et l'échange d'expérience. Enfin, nous ne chercherons pas à spécifier des "types d'information" pour le contenu des signatures, car nous tenons compte des difficultés à convenir d'une caractérisation robuste d'une telle catégorisation, en accord avec les

points soulevés au chapitre 2.

La section suivante décrit plus en détail les différents comportements qui composent notre agent assistant.

5.2 Architecture de l'agent assistant

Pour gérer la base d'expériences individuelles, nous utilisons le paradigme de *raisonnement à partir d'expérience tracée* (RàPET) présenté en 4.4.1. Rappelons que dans cette approche, l'expérience individuelle est capturée en observant les interactions de l'utilisateur avec des objets d'intérêts préalablement identifiés dans le système, et elle est représentée dans le système par des traces d'utilisation. Pour retrouver l'expérience passée, un raisonnement par analogie entre la trace en cours de formation et les traces antérieures est effectué en considérant des épisodes similaires dans ces traces, épisodes élaborés selon une signature donnée.

Pour modéliser les négociations de sens entre l'utilisateur et son assistant informatique personnel (et dans la perspective d'autres négociations entre assistants personnels), nous nous appuyons sur le paradigme des Systèmes Multi-Agents ; le système que nous allons considérer est un système hybride, i.e., l'utilisateur et son agent assistant. Dans la suite du document, cet assistant informatique personnel sera appelé "agent alter ego", lorsque nous voudrons désigner l'agent "autonome" d'assistance dont le principal objectif est de "coller" aux mieux aux notions utilisées, en situation, par l'utilisateur ; ou simplement, "assistant", par simplicité ou pour désigner sa fonctionnalité ultime.

Les comportements de l'agent alter ego sont décrits dans les sections suivantes, et nous discuterons brièvement les intérêts de notre approche et les difficultés qui en incombent.

5.2.1 Comportement d'acquisition de trace et de RàPET

L'*acquisition* d'observations des objets d'intérêts manipulés par l'acteur humain au travers de l'application cible, et l'*élaboration* d'un épisode cible sur la trace résultante, tel que cela est illustré sur la figure 5.1.

Les observations des OI sont structurées en trace (une succession d'états et de transitions) et forment la trace d'interaction en cours de construction, nous l'appellerons par la suite *trace en cours*. Cette trace est soumise au module de RàPET, qui est chargé d'élaborer un épisode cible et de retrouver au regard de cette élaboration les épisodes similaires situés dans des traces antérieures.

L'élaboration consiste à reconnaître un motif dans la trace conforme à une signature donnée et à lui donner l'étiquette symbolique de la signature. Il s'agit donc d'une reformulation symbolique, chaque symbole ayant sa sémantique décrite par la signature associée. Comme support de négociation sur les signatures et leur sens, nous proposons de rassembler l'ensemble des signatures disponibles à un instant donné, dans un "état de langage" L_n, n étant l'indice de d'état (on commencera avec un état de langage L_0, théoriquement vide, tout comme la base de trace). Nous définissons formellement les notions de trace, symbole, signature au chapitre suivant.

Notre approche nous conduit à rassembler l'ensemble des signatures, disponibles à un instant, dans un *état de langage* L_n ; nous définirons formellement la notion d'état de langage au chapitre 7. Ce comportement d'acquisition de trace s'effectue en tâche de fond, de façon transparente à l'utilisateur. A ce niveau, l'agent alter ego peut bénéficier des nombreux moments d'inactivité

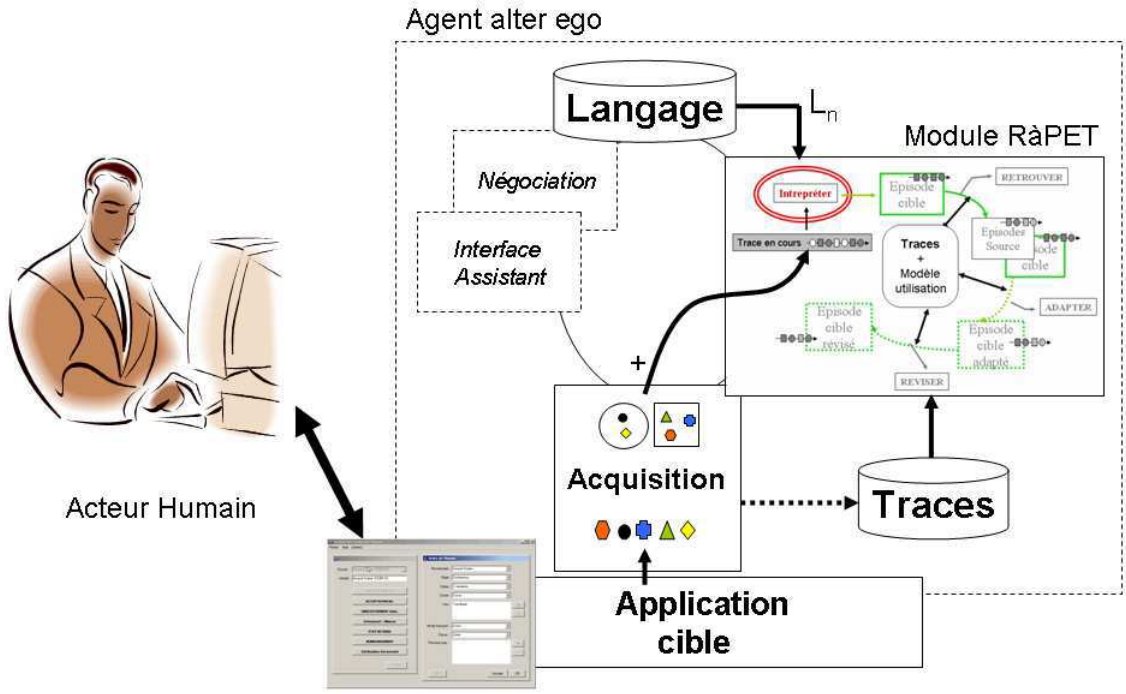


FIG. 5.1 – Architecture de l'agent alter ego : comportement d'acquisition et d'élaboration d'épisode cible

de l'ordinateur pour effectuer ces élaborations et recherches d'épisodes.

5.2.2 Comportement d'assistance à la réutilisation

le comportement d'*assistance à la réutilisation* est déclenché lorsque l'utilisateur fait appel à son assistant comme illustré sur la figure 5.2.

L'assistant soumet à l'utilisateur les différentes élaborations qui ont été effectuées au regard des différentes signatures définies dans l'état de langage L_n en cours. L'utilisateur choisit une élaboration qui lui est proposée ; les différents épisodes étiquetés symboliquement selon des élaborations faites avec la même signature, sur des traces antérieures, sont proposés à l'utilisateur. L'agent alter ego enregistre les réactions de l'utilisateur sur les éléments symboliques d'élaboration (ou "symboles"). De manière générale, une réaction de l'utilisateur vise à indiquer ce qui est à prendre en compte, et de quelle manière, dans une trace ; ceci sera représenté par un symbole de langage. Ce comportement implique la construction d'une interface graphique pour permettre la suggestion d'épisodes à l'utilisateur et la capture de ses réactions.

5.2.3 Comportement de co-construction de sens par négociation

Pour guider l'acquisition de sens commun entre l'agent humain et l'agent alter-ego, nous proposons d'introduire un comportement de *co-construction de sens par négociation* entre les deux agents. Ce processus a pour objectif de faire évoluer les éléments d'élaboration d'épisodes en intégrant les réactions de l'utilisateur, qui peuvent être source d'ambiguïtés pour l'agent alter ego. L'interface graphique, qui est associée à ce processus de co-construction par négociation, est

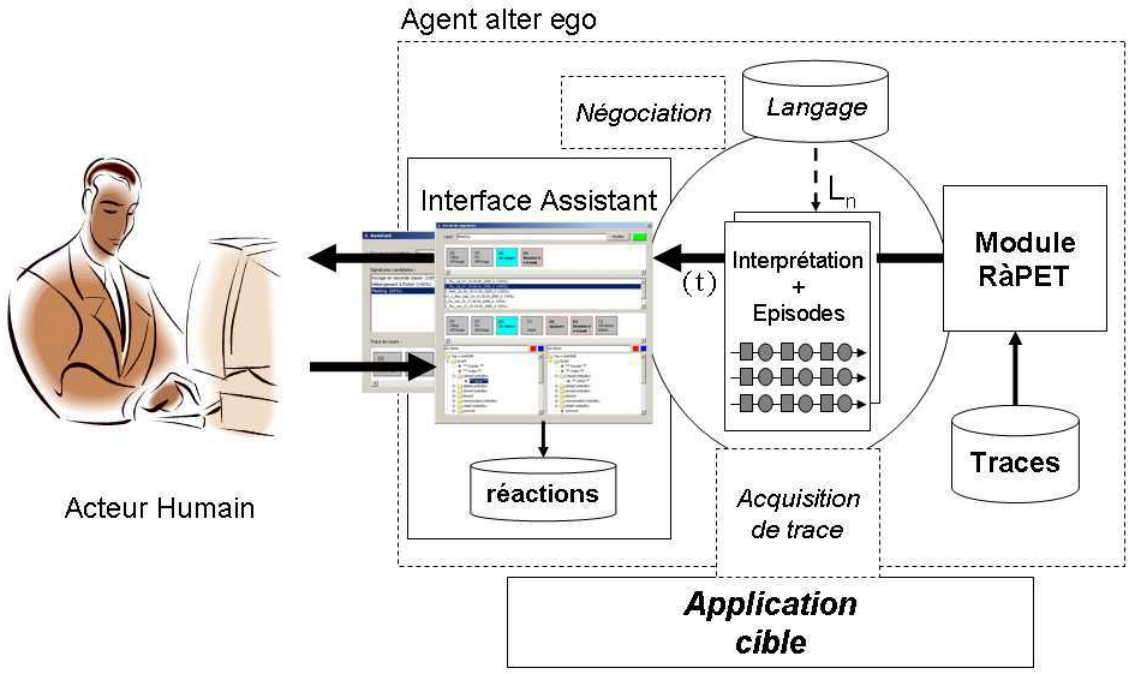


FIG. 5.2 – Architecture de l'agent alter ego : comportement d'assistance à la réutilisation de l'expérience

illustrée sur la figure 5.3.

Notons que l'acteur humain doit passer par une phase d'appropriation des possibilités et des limites d'interprétation de son agent alter ego, et que l'agent alter ego a pour principe d'utiliser les indications données par l'acteur humain comme base pour de futures interprétations.

Plus précisément, afin d'obtenir une élaboration dont le sens convient à l'acteur et qui est utilisable son agent alter ego, nous proposons de transposer les mécanismes d'émergence de langage de forme lexicale, synthétisés par [Steels 00]. Nous faisons cela en considérant les interactions entre l'acteur et son agent alter ego comme un processus de négociation de sens portant sur l'élaboration de la trace. Cette approche nécessite de disposer d'une représentation symbolique de la trace. L'acteur humain construit un symbole en spécifiant parmi les éléments d'une section de trace quels sont ceux qui sont significatifs pour sa tâche en cours. Les symboles sont utilisés comme médium de communication entre l'acteur et son agent alter ego, et ils sont utilisés pour exprimer un sens d'assistance, i.e., une certaine interprétation. Nous proposons ainsi d'appliquer la métaphore et les principes d'émergence de langage pour modéliser les connaissances d'interprétation sous la forme d'un système adaptatif complexe.

La figure 5.3 décrit les composants mobilisés et le cycle suivi, dans le cadre d'une négociation : une élaboration d'épisodes conformes à une signature du langage L_n en cours a été réalisée ; les descriptions symboliques des épisodes retrouvés sont soumis à la réaction de l'utilisateur. Les réactions sont soumises à négociation, ce qui permet d'affiner les réactions et d'envisager la révision de l'état du langage (modification de sens d'un symbole, création de symboles, etc.). Cette

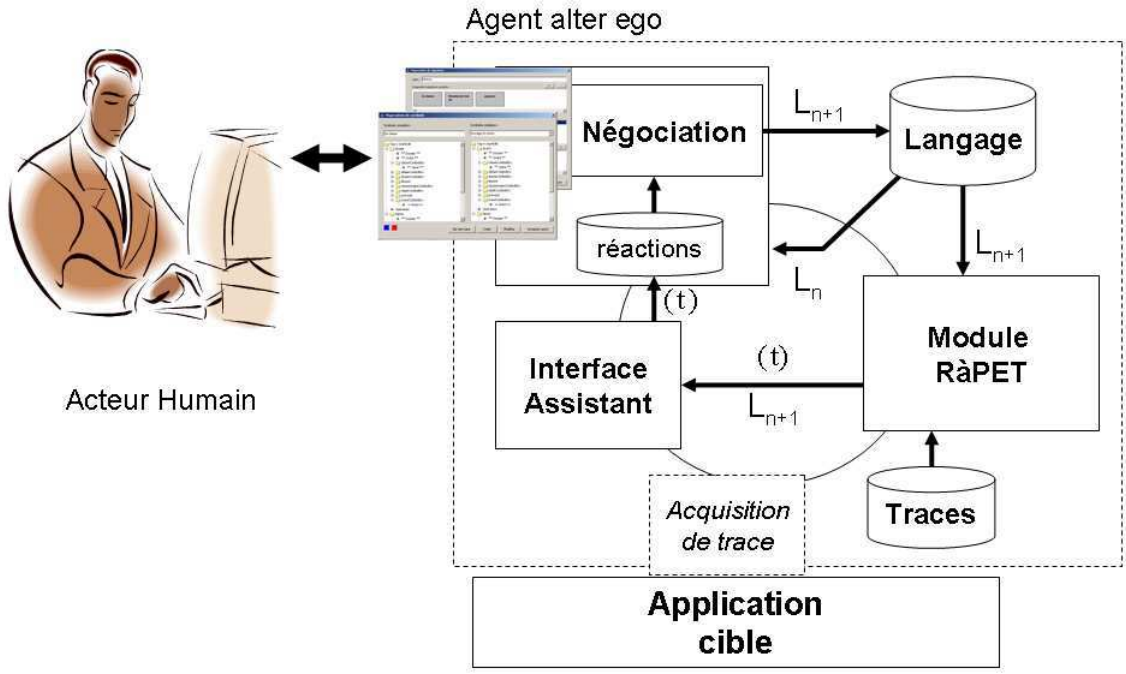


FIG. 5.3 – Architecture de l'agent alter ego : comportement de négociation de sens avec l'acteur humain

construction d'un nouvel état de langage L_{n+1} s'inspire des principes énoncés par [Steels 00]. Une nouvelle élaboration avec recherche d'épisodes conformes à ce nouvel état du langage est menée via le module RàPET, elle permet d'effectuer de nouvelles suggestions à partir de la signature négociée, et réitère une nouvelle tentative d'assistance à la réutilisation de l'expérience.

Ce comportement de négociation de sens est activé à chaque fois que l'acteur humain souhaite une mise à jour de l'élaboration et des rapprochements, qui tienne compte de ses réactions. A chaque itération, un nouvel état de langage est construit.

5.3 Méthode, intérêts et difficultés de notre approche

Notre démarche est générique. Elle se fait sans hypothèse sur un domaine d'activité particulier, et vise à être un support aux acteurs d'une activité collective complexe. L'étude que nous faisons de notre agent part du principe d'un agent autonome, qui doit rester sous le contrôle de l'utilisateur auquel il est destiné, tout ceci ne pouvant se faire qu'avec un surcroît d'activité très limité. Ces deux objectifs nous conduisent, dans un premier temps, à privilégier le contrôle à l'autonomie, bien que ce soit l'autonomie qui est la valeur même du système que nous proposons. C'est pour cela que notre présentation inclue une discussion de cet écart, à différents niveaux, et des conditions qui doivent être remplies pour le combler. Nous traiterons en perspective des moyens d'étendre cette autonomie et de l'accroissement de proactivité qui en résulterait pour l'agent alter ego.

La construction des traces requiert une représentation du domaine et de l'activité, nous proposerons donc un modèle avec certaines propriétés, mais celui-ci n'est pas le centre de notre

travail. A partir de cela, nous allons décrire de manière détaillée et formelle :

- un modèle générique de représentation de trace d'utilisation et les mécanismes génériques permettant d'effectuer des rapprochements selon des situations d'intérêt, puis ;
- un modèle de représentation du sens des situations d'intérêt pour l'utilisateur, selon les principes d'émergence d'un langage de forme lexicale. Ceci inclue des mécanismes génériques d'acquisition de ces sens, utilisés de façon située, en interaction avec l'utilisateur.

Notre étude se concentre sur le fonctionnement interne de l'agent assistant, soit 1) la capture de trace et son rapprochement avec des traces antérieures, selon les sens disponibles à ce moment, et 2) la révision des définitions (informatiques) de ces sens ou l'acquisition de nouveaux, à la suite de réactions de l'utilisateur. Cette deuxième partie sous-tend l'introduction de mécanismes et de composants d'interface graphique ; ces éléments seront principalement vus comme une illustration de ce qui est envisageable avec les objets graphiques les plus courants. Nous faisons cela afin d'étudier les moyens permettant à l'utilisateur de manipuler "intuitivement" les objets (informatiques) que nous allons introduire ; ceux-ci lui permettent d'exprimer ce qui fait sens dans sa situation courante et comment ce sens se caractérise concrètement, i.e., sur les éléments de trace à sa disposition.

De la sorte, nous ne traiterons pas les questions ergonomiques qui vont naturellement avec l'introduction de ces nouveaux éléments d'interface graphique. Nous considérons que c'est principalement l'expérimentation de leurs usages qui permettrait d'évaluer cette dimension, expérimentations que nous n'avons pas été en mesure d'effectuer. Néanmoins, en utilisant des éléments d'interface standards, nous pouvons faire l'hypothèse d'un effort d'appropriation réduit concernant leur fonctionnement ; c'est donc avant tout leur "ergonomie cognitive" dans la tâche de manipulation en situation de l'expérience tracée, qu'il conviendrait d'étudier.

Un premier intérêt de l'approche que nous proposons est d'acquérir en tâche de fond une description des situations rencontrées par l'acteur humain, et de permettre un accès contextuel aux situations passées en appliquant les principes du RàPET. Cette approche est adaptative par nature. Le module de RàPET est limité aux étapes d'interprétation et de recherche d'épisode, nous traiterons dans les perspectives (c.f., 10.5) de la réalisation des étapes restantes.

La contribution majeure de notre travail est de mettre en place un cadre permettant la capture de sens en interaction avec l'utilisateur. Cette opération est indispensable à l'agent alter ego pour être en mesure de faire des rapprochements avec les traces antérieures qui soient porteurs de sens pour l'utilisateur.

Ainsi, nous cherchons à bénéficier des interactions entre l'acteur humain et son agent alter ego afin de capturer une définition de ce qui est significatif pour l'acteur humain dans le contexte de sa tâche en cours. Ne pouvant complètement et précisément définir les éléments significatifs à l'avance, restreignant ainsi l'interprétation à un modèle prédéfini, nous préférons laisser à l'acteur humain le soin de spécifier à l'agent alter ego ce qui est intéressant pour lui. Toutefois, pour des raisons pratiques, nous ne pouvons requérir de l'acteur humain qu'il spécifie à chaque fois entièrement ce qui est significatif, en particulier quand il l'a déjà fait : son assistant doit donc être proactif en réutilisant les définitions antérieures pour proposer une interprétation de la trace en cours. En dehors de cette proactivité, nous suivront dans cette deuxième partie une démarche faisant que l'agent assistant cherche à "suivre" au plus près l'utilisateur ; ceci afin d'éviter que l'utilisateur ne perde le contrôle de son assistant. Les mécanismes de négociation proposés pourront donc sembler proche de la "paramétrisation", bien que cette même négociation impliquera des mécanismes de suggestion d'autres sens par rapprochement ; nous traiterons en

perspective des possibilités de doter l'agent d'une réelle autonomie sur des points que nous laissons, dans un premier temps, à la charge de l'utilisateur.

Une difficulté de fond sera de retranscrire les spécifications faites par l'acteur humain en un processus de négociation de sens conduisant à un langage émergent, de façon la moins intrusive possible et la plus intelligible. Les fondamentaux de l'ergonomie nous limitent à attendre de l'utilisateur qu'un nombre limité d'opérations. Ceci impose des conditions contraignantes à notre transposition des mécanismes d'émergence de langage ; de plus, nous prenons bien en considération que nous ne pouvons requérir un surcroît de travail de la part de l'utilisateur sans contrepartie.

5.4 Guide de Lecture

Dans le chapitre 6, nous allons présenter nos modèles de trace, de symbole et de signature ainsi que les mécanismes de RàPET qui sont au cœur de l'agent alter ego. Nos traces sont issues d'observations qui portent sur des objets d'intérêts, relevant du domaine d'application. Nous présenterons le modèle de domaine que nous retenons, toutefois ces problématiques ne sont pas le centre de nos recherches ; nous nous baserons seulement sur certaines propriétés. Nous présenterons ensuite différentes mesures de similarités permettant in fine d'évaluer la similarité entre deux épisodes extraits de traces à l'aide d'une signature. A partir de tous ces éléments, la section 6.6 décrira les algorithmes que nous considérons pour le fonctionnement du module de RàPET de l'agent alter ego.

Le chapitre 7 détaille notre transposition des principes d'émergence de langage pour porter un processus de co-construction de sens par négociation entre l'utilisateur et son agent alter ego. Ceci passera par une structuration des phases d'interaction entre l'utilisateur et son agent alter ego, qui surviennent lors du passage en alternance du comportement de suggestion d'expérience à celui de négociation, et inversement, tel que cela est le cas lors d'une session d'assistance prolongée. De cette structuration, et du traitement des ambiguïtés de sens par négociation, découle notre transposition des mécanismes d'émergence de langage dont nous ferons une description algorithmique et une illustration sur un exemple.

Ces deux chapitres décrivent les fondements théoriques de notre approche. Ils sont intrinsèquement inter-dépendants, c'est pourquoi des principes relevant du chapitre 7 seront introduits très tôt dans le chapitre 6. De nombreuses références seront ainsi données pour faciliter les rapprochements au lecteur, bien que nous ayons tout fait pour permettre une lecture linéaire.

Le chapitre 8 fait une discussion des principes que nous aurons présentés, pour la représentation de l'expérience tracée tout comme pour les mécanismes de co-construction de sens. Un positionnement par rapport à des travaux connexes sera ensuite effectué.

Le chapitre 9 présentera le démonstrateur que nous avons développé. Une description de ces composants logiciels sera effectuée, ce qui inclue une explicitation des interfaces, dont les rôles et les fonctionnements aurons été largement décrits auparavant. Nous ferons une brève illustration sur un scenario d'activité "jouet". Puis, nous présenterons le premier retour d'expérience issu de ce développement.

Chapitre 6

Représentation des traces d'utilisation et de leur sémantique

Sommaire

Dans ce chapitre, nous allons détailler notre mise en oeuvre du modèle MUSETTE, présenté en 4.4.2. Nous présenterons également les évolutions que nous lui apportons afin de bénéficier d'une description fonctionnelle de l'application cible ; cette évolution est à la base de notre reformulation symbolique des traces, et elle donne la possibilité de situer les indications données par l'acteur humain. Ce positionnement des symboles sera fait vis à vis des fonctionnalités de l'application cible, qui elle-même s'articule autour d'un modèle implicite de l'activité, dont elle assure la médiation.

Dans le modèle MUSETTE, une trace d'utilisation est définie par une séquence alternée d'états et de transitions. Les *objets d'intérêt* observables durant la tâche considérée sont décrits dans un modèle d'utilisation, ce sont soit des entités soit des événements. Une *trace d'utilisation* décrit les changements intervenant dans le système, et qui sont observables par l'acteur humain. Un état est constitué d'objets observables, considérés comme étant dans un état stable. Les transitions représentent la réalisation d'actions par l'utilisateur, sous forme de séquences d'opérations ; ces

dernières sont définies par des événements, ainsi que par d'éventuelles entités supplémentaires. Sous cette forme, une trace d'utilisation est appelée *trace primitive*.

Pour manipuler les traces, l'assistant doit être en mesure de reconnaître certaines situations en s'appuyant sur des signatures, et l'acteur humain est le mieux placé pour juger de ce qui est porteur de sens, de ce qui ne l'est pas, dans le contexte de sa tâche en cours. Pour mettre en oeuvre notre approche basée sur un processus de négociation de sens entre l'acteur et l'agent alter ego, nous devons disposer d'une représentation symbolique de la trace et des signatures. Ceci constitue une évolution du modèle MUSETTE.

Dans la suite, les choix que nous avons effectués (représentation, similarité, algorithme, ...) sont faits sur la base de leur utilisabilité pour les besoins de la recherche et non pas pour leur valeur intrinsèque même. Chacun de ces choix pourrait faire l'objet d'une étude particulière, et venir remplacer nos propositions ; néanmoins, ils font l'hypothèse de certaines propriétés et s'attachent à proposer une trame pour gérer les spécificités de ces propriétés.

Notre recherche s'intéresse principalement à l'étude d'un mécanisme général d'acquisition et de négociation de sens par l'agent alter ego, en interaction avec l'utilisateur.

6.1 Représentation du domaine d'application

La première étape dans la mise en oeuvre du modèle MUSETTE est l'identification des objets d'intérêt, afin de disposer d'un modèle d'utilisation. Dans notre cas, nous nous intéressons à une application cible à base documentaire, qui est support d'une activité collaborative complexe. Pour cela, nous devons considérer les types de documents manipulés par les acteurs humains, ainsi que la terminologie utilisée pour décrire les situations rencontrées. Etant donné que nous nous intéressons, dans un premier temps, uniquement à l'expérience individuelle d'un acteur, nous allons donc considérer simplement qu'il manipule différents types de documents, sans nous intéresser aux relations qui existent entre ces documents. Un document est défini par sa structure interne constituée de différents champs, dans l'absolu une arborescence. Les champs contiennent des mots clés, qui constituent la terminologie spécifique au domaine d'application.

Pour cette terminologie, étant donné que notre démarche est de donner à l'acteur humain le moyen d'exprimer ce qui est significatif pour lui dans le contexte de sa tâche, il est intéressant de lui donner un moyen d'exprimer une abstraction (ou *généralisation*). Dans la structure terminologique définie ci-dessous, il s'agit de donner à l'utilisateur le moyen d'exprimer des abstractions, i.e., de considérer un mot clé comme l'expression d'un concept. La sémantique d'une telle abstraction est que ce mot clé peut être remplacé par un autre qui représente le même concept de généralisation, une proximité entre de tels mots clés est alors calculable, et une telle généralisation est ce qui compose la description d'un *sens de dénotation*, indiqué par l'utilisateur.

A titre d'exemple, nous proposons donc d'organiser les mots clés à l'aide d'une hiérarchie de concepts, tel que cela est illustré sur la figure 6.1. Une pondération $a_k > 1$ est associée à chaque arc.

Cette organisation des mots clés et des concepts qu'ils représentent se résume à un arbre de termes, où les feuilles sont des mots clés (KW_i) et les nœuds des concepts (Cpt_i). Nous faisons par la suite l'hypothèse simplificatrice que seuls les mots clés sont considérés pour décrire les documents. Nous introduisons la mesure de similarité entre termes, qui sera utilisée soit entre mots clés, soit entre concepts. La distance $d(\Gamma_i, \Gamma_j) \in \{0\} \cup]1; \infty[$ entre deux termes Γ_i et Γ_j est

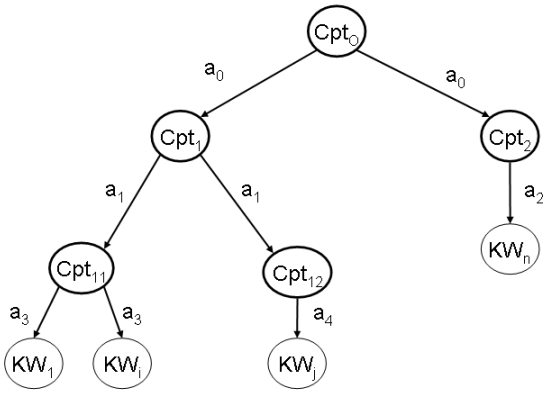


FIG. 6.1 – Représentation des termes du domaine

définie par :

$$\begin{aligned} d(\Gamma_i, \Gamma_j) &= \prod_{chemin\ \Gamma_i \leftrightarrow \Gamma_j} a_k \qquad si\ i \neq j \\ &= 0 \qquad si\ i = j \end{aligned}$$

A partir de laquelle, nous définissons la similarité $sim(\Gamma_i, \Gamma_j) \in]0;1]$ entre termes :

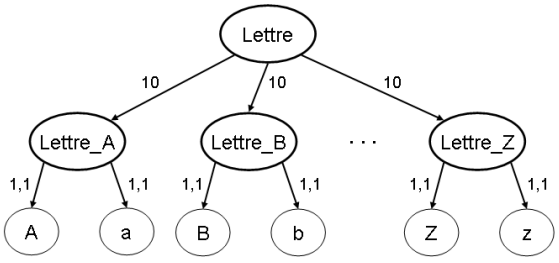
$$\begin{aligned} sim(\Gamma_i, \Gamma_j) &= \frac{1}{d(\Gamma_i, \Gamma_j)} \qquad si\ i \neq j \\ &= 1 \quad si\ i = j \end{aligned} \tag{6.1}$$

Par la suite, nous appellerons *modèle de domaine* l'ensemble des mots clés et des concepts ainsi introduits. Cette structure n'est qu'un exemple rudimentaire d'organisation des termes du domaine, elle peut être remplacée par une ontologie plus fine, ou au contraire être absente. Dans ce dernier cas, la mesure de similarité serait de 1 pour les mêmes mots clés, et de 0 sinon.

Exemple

A titre d'illustration simple de notre modèle de domaine, considérons le "monde des lettres" tel que nous le concevons communément en évoquant les concepts de majuscules, minuscules, consonnes, voyelles, ordre alphabétique.

La figure ci-dessous décrit un premier modèle, qui est minimal, où il est introduit le concept général de *Lettre*, situé à la racine de la hiérarchie de concepts et les concepts fils *Lettre_ ?* pour chacune des lettres *?* de l'alphabet. Ce modèle exprime qu'il convient de rapprocher les lettres (A, a, B, b, ...) en ne tenant pas compte du fait qu'elles soient en majuscule ou minuscule.



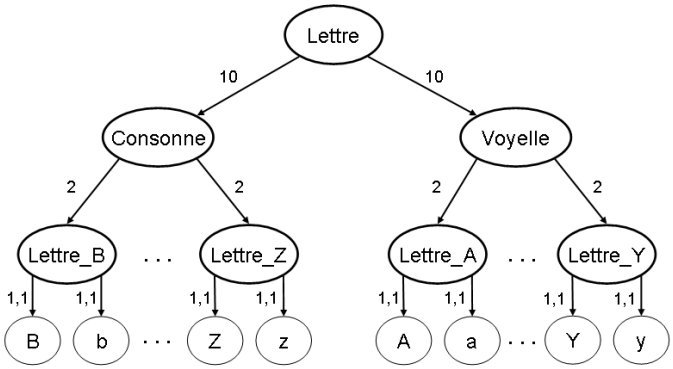
Ainsi, on a :

$$sim(A, a) = \frac{1}{1,1 \cdot 1,1} \approx 0.83$$

et pour tous les autres :

$$sim(A, Z) = sim(a, z) = sim(A, z) = \frac{1}{1,1 \cdot 10 \cdot 10 \cdot 1,1} \approx 0.008$$

Ce premier modèle peut dans certains cas être inadapté, et la connaissance de similarité peut s'exprimer sur un autre modèle. Par exemple, le modèle ci-dessous exprime que la dimension consonne/voyelle doit être ajoutée au précédent.

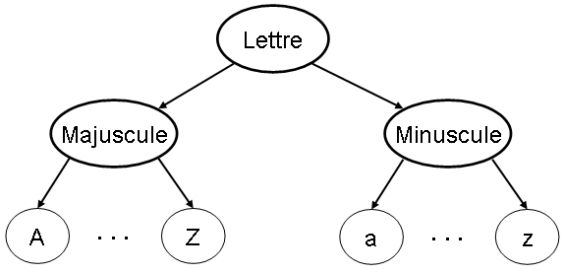


On a alors :

$$sim(A, e) = \frac{1}{1,1 \cdot 2 \cdot 2 \cdot 1,1} \approx 0,21$$

$$sim(a, b) = \frac{1}{1,1 \cdot 2 \cdot 10 \cdot 10 \cdot 2 \cdot 1,1} \approx 0,002$$

Cependant, il n'est pas toujours possible de représenter toutes les connaissances de similarité simplement avec une hiérarchie de concepts. Différentes répartitions des instances (ici, des lettres) peuvent s'avérer incompatibles entre elles, tel que cela est illustré sur le troisième modèle ci-dessous. L'organisation en hiérarchie de concepts n'est pas nécessairement adaptée pour représenter certaines connaissances de similarité, tel que la prise en compte de l'ordre alphabétique.



Dans les systèmes opérationnels de RàPC, il est généralement fait recours à des mécanismes de pondération entre de multiples mesures de similarité, en laissant à l'utilisateur le moyen d'ajuster les poids des différentes mesures pour corriger le classement des suggestions et ainsi améliorer les performances du système. Néanmoins, l'estimation de la similarité est une opération faisant intervenir des connaissances que l'utilisateur est le plus approprié à donner en situation. Dans une certaine mesure, l'ajustement des pondérations de similarité au moment de la suggestion est une expression limitée de ces connaissances avec une valeur locale ; nous pensons qu'il est préférable de laisser à l'utilisateur le moyen d'exprimer explicitement les connaissances qu'il mobilise, par exemple en le laissant indiquer explicitement quelle dimension de comparaison il choisit par le biais d'une généralisation dans cette dimension. Dans notre exemple du *domaine de lettres*, cela reviendrait à choisir un concept généralisant d'un des trois modèles de domaine. Cette question reste un problème ouvert : si l'expression des connaissances de similarité repose sur une combinaison logique et/ou arithmétique plus complexe (par exemple, dans le *domaine de lettres*, deux lettres α et β pourraient être considérée proches si "$Voyelle(\alpha, \beta)$ & $(Majuscule(\alpha)||Minuscule(\beta))$ et d'autant plus que $|\ ordre\_alphabétique(\alpha) - ordre\_alphabétique(\beta)\ |$ est minimal"), une moyenne pondérée entre différentes mesures de similarité n'est pas suffisante pour représenter cette connaissance complexe, et notre approche à base d'interaction avec l'utilisateur nécessiterait le développement d'une interface donnant à ce dernier le moyen d'exprimer ces connaissances, avec les contraintes ergonomiques que cela entraîne.

L'approche, que nous présentons par la suite, ne fait pas d'hypothèse sur le modèle du domaine adopté. Seule la possibilité d'exprimer une généralité nous intéresse et le moyen de calculer la similarité entre instances (dans la suite, des mots clés) respectant la généralisation. Par conséquent, une place importante est laissée à l'utilisation des travaux existants dans le domaine de la représentation des connaissances.

6.2 Représentation des observations de trace et de symbole

Dans le modèle MUSETTE, une trace est une succession alternée d'états et de transitions, représentant respectivement des observations des états stables du système, et des observations décrivant les actions de l'acteur humain. Les observations sont composées d'objets d'intérêt (OI) propres au domaine d'application. L'assistance par rapprochement de la situation en cours avec des situations antérieures similaires, selon les principes du RàPET, nécessite d'avoir des connaissances d'identification des situations d'assistance, qui sont appelées *signatures de tâche*.

Comme il n'est pas possible de définir a priori les signatures de tâche, notre approche est d'acquérir ces connaissances en tirant parti des indications données en situation par l'utilisateur. Il résulte de cette approche que le modèle décrivant les OI observés, dans un état ou une transition, guide fondamentalement le modèle décrivant les indications données par l'utilisateur sur ces OI ; nous les présenterons donc conjointement en 6.2.1.

Avant de détailler ce modèle, nous faisons une brève introduction des principes fonctionnels et des notions clés de notre approche, dans le but de donner le fil conducteur des différents aspects traités dans les chapitres 6 et 7.

Pour que l'agent alter ego puisse mémoriser les indications qui font sens à l'utilisateur lors de ses descriptions de situations d'assistance, nous allons effectuer la transposition des mécanismes d'émergence de langage, présentée au chapitre suivant ; schématiquement, les signatures sont rassemblées dans un *état de langage*, qui va évoluer en fonction des indications de l'utili-

sateur. Cette approche à base de langage émergent passe par une représentation symbolique de l'environnement de l'agent alter ego, soit les traces et les signatures de tâche.

La représentation symbolique des traces sera présentée en 6.3. Elle passera par le "découpage" d'une trace en une suite de *sections* ; sur chacune d'elles, l'agent alter ego va chercher à *percevoir* des symboles définis dans son *état de langage* en cours. De cette perception de symboles résulte la représentation symbolique d'une trace.

Une signature de tâche représente la description d'une situation d'assistance, nous la définirons en 6.4 sous la forme d'un ensemble de symboles. En reconnaissant la perception d'une partie des symboles définissant une signature dans la trace en cours, l'agent alter ego effectue l'*élaboration* de l'épisode cible ; il va le comparer par analogie avec des épisodes sources, issus d'élaborations sur des traces antérieures avec la même signature ; la section 6.6 présentera la mise en oeuvre des mécanismes de RàPET.

Ainsi, la notion de *symbole* joue un rôle central dans notre approche, nous la définirons en 6.3.2. Schématiquement, un symbole sera la paire formée 1) d'une "observation" de ce qui a été déclaré significatif par l'acteur humain dans une section, nous l'appelons le *motif* d'un symbole, et 2) d'un label en langage naturel donné par l'acteur humain, afin d'exprimer sa signification ; c'est ce label qui sera utilisé dans la représentation symbolique d'une élaboration à l'utilisateur. Un motif de symbole aura la même structure qu'une section de trace, i.e., une succession alternée d'états et de transitions. Aux transitions et états d'un symbole seront associés des observations de symbole, appelées par la suite *patterns*. La perception d'un symbole sur une section consiste à trouver un appariement de son motif à cette section ; cela revient à trouver un appariement pour chacun de ses patterns avec l'observation d'une section de trace, qui lui correspond en respectant la structure commune. L'opération d'appariement de pattern à une observation de trace sera décrite en détail en 6.6.1.1. Pour l'utilisateur, la sémantique que nous donnons à la perception d'un symbole dans une section par l'agent alter ego est : "les conditions d'identification portées par le symbole sont vérifiées dans cette section" ou encore "la section représente une situation décrite par le symbole".

La section 6.2.1 présente notre modèle d'observation de trace et de pattern, dans la logique d'un appariement du pattern à l'observation de trace. La section 6.2.2 présente notre mesure de similarité entre deux observations de trace, toutes deux appariées à un pattern ; cette mesure est la point de départ pour la mesure de similarité entre l'épisode en cours et un épisode antérieur, élaborés avec une signature, qui est nécessaire à l'étape de recherche d'épisode du RàPET. La section présente une mesure de similarité entre patterns, qui sera utilisée dans notre transposition des mécanismes d'émergence de langage, c.f., chapitre 7.

6.2.1 Modèle d'arbre d'observation et de pattern

Dans notre introduction de cette section, nous avons indiqué que le modèle décrivant les OI d'une observation de trace, associée à un état ou une transition, guide fondamentalement le modèle décrivant les indications données par l'utilisateur sur les OI de cette observation, i.e., le modèle de pattern. Cela vient du fait que l'utilisateur donne ses indications (i.e., il indique qu'un OI est significatif) en interagissant avec les OI d'une observation de trace. Pour l'acquisition de pattern qui résulte de cette réaction (un ensemble d'indications) de l'utilisateur, nous sommes tout naturellement amenés à décrire ses indications en adoptant la même organisation pour les indications que celle des OI dans l'observation de trace. Une autre motivation de cette approche est de réduire tant que possible la complexité de l'appariement d'un pattern sur une observation de trace, nécessaire à la perception d'un symbole.

Notre application cible est une application à base documentaire, permettant d'afficher des entités documentaires, qui sont structurées en champs et qui contiennent des mots-clés (les objets d'intérêt propre au domaine). Cela nous conduit à adopter une structure hiérarchique pour positionner les OI dans l'observation d'une entité documentaire ; c'est également la structure des OI tels qu'ils apparaissent à l'utilisateur.

Pour exprimer cette structure hiérarchique d'une observation, et donc d'un pattern, nous introduisons la notion de *container*. Une observation ou un pattern est un container dans son ensemble, appelé *container racine*, et ce container contient des OI et/ou d'autres containers. Cette organisation en hiérarchie de composition de containers représente la structure commune à une observation de trace et à un pattern, par la suite nous parlerons d'*arbre d'observation* de trace. Pour intégrer des connaissances de discrimination propres à la mise en place de l'observation[1], et dans le souci de réduire la complexité de l'appariement, nous associons un label à chaque container, en ajoutant la condition que ce label est distinct de ceux des containers frères. La condition de l'appariement d'un container de pattern à un container d'arbre d'observation est qu'ils aient le même label et qu'ils soient dans la même position par rapport aux containers racines respectifs. La condition d'un label distinct pour les containers frères est une contrainte limitant l'expressivité d'un arbre d'observation et d'un pattern, mais elle évite toute alternative lors de l'appariement de containers ; nous discuterons de cela en 6.6.1 lorsque nous présenterons l'algorithme d'appariement d'un pattern à un arbre d'observation.

Pour représenter l'observation d'un OI à un endroit dans la hiérarchie de containers d'un arbre d'observation, nous introduisons la notion de *capture*. De cette manière un OI peut être présent dans plusieurs containers via différentes captures. De la même manière, nous utilisons des captures pour représenter une indication de l'utilisateur sur un OI, à un endroit dans la hiérarchie de containers d'un pattern.

Les captures d'arbres d'observation et les captures de pattern se distinguent par leur sémantique :

- une capture d'arbre d'observation exprime simplement la présence d'un OI (ici, un mot-clé). Des informations de temporalité ou l'expression d'une vue partielle, ou facette, pourraient être ajoutées à la capture ;
- une capture de pattern exprime une indication donnée par l'utilisateur visant à définir que l'OI (le mot-clé) est significatif, soit :
  - en tant que tel. Nous parlerons de capture constante, ou *constante* ;
  - en tant qu'instance d'un concept généralisant, défini dans le modèle de domaine, c.f., 6.1. Nous parlerons de capture variable, ou *variable*, faisant référence au concept d'abstraction indiqué.

L'appariement d'une capture de pattern à une capture d'arbre d'observation, situées dans des containers qui peuvent être appariés, est possible :

- dans le cas d'une capture de pattern constante, si l'OI représenté est le même que celui représenté par la capture d'arbre d'observation ;
- dans le cas d'une capture de pattern variable, si l'OI représenté par la capture d'arbre d'observation est une instance du concept représenté par la (capture) variable de pattern. Dans les exemples qui vont suivre nous utiliserons des abstractions au généralisant immé-

[1] par exemple, qu'il ne convient pas de comparer deux zones d'observation, représentées par des containers, car elles ne représentent pas les mêmes choses

diat ; toutefois, notre modèle s'applique à tous les généralisants (porteurs de sens), nous verrons en 6.6.1.1 les difficultés qu'entraînent des généralisants en compétition dans un même container lors de l'appariement.

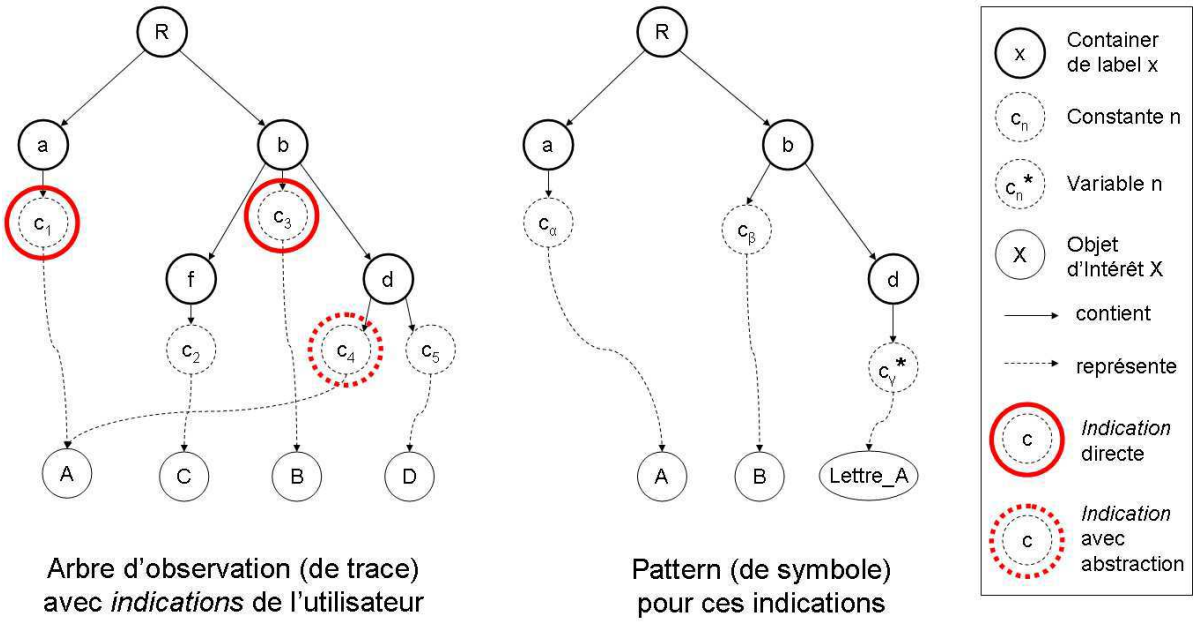


FIG. 6.2 – *Arbre d'observation* de trace et *pattern* de symbole

La figure 6.2 illustre notre modèle, et les notions de container et de capture, en donnant un exemple d'arbre d'observation (de trace) et de pattern (de symbole) ; nous figurons en rouge les indications de l'utilisateur sur cet arbre d'observation et qui sont représentées par le pattern.

Sur la base de l'un des modèles de domaine donné en exemple dans la section précédente, considérons que le pattern a été obtenu en transcrivant les indications suivantes données par l'utilisateur sur l'arbre d'observation : 1) l'OI "A", représenté par la capture c_1, est significatif en soi à l'endroit correspondant au container de label a, 2) de même pour l'OI "B" qui est représenté par la capture c_3, située dans le container de label b, et 3) l'OI "A", représenté par la capture c_4 dans le container de label d, est significatif en tant qu'instance du concept "Lettre_A". Il en résulte que le pattern réplique la hiérarchie des containers de labels R, a, b et d, et contient les captures constantes 1) c_α et 2) c_β et la capture variable 3) c^*_γ dans leur container respectif pour représenter les indications 1), 2) et 3). Inversement, l'appariement du pattern à cet arbre d'observation est possible.

Dans les sections 6.2.2 et 6.2.3 ci-dessous, nous utilisons une représentation simplifiée afin de faciliter la lecture. La figure 6.3 illustre cette représentation simplifiée pour l'arbre d'observation et le pattern décrits sur la figure 6.2 : 1) les containers candidats sont représentés par des boîtes rectangulaires et ils ne sont appariables que s'ils sont à la même position par rapport à la racine, et 2) les captures sont représentées directement par les OI qu'ils désignent, les variables sont notées avec une étoile. Nous avons ajouté en rouge les indications données par l'utilisateur sur

cet arbre d'observation et qui ont permis de construire le pattern. Le concept $Lettre_A$ est noté L_A.

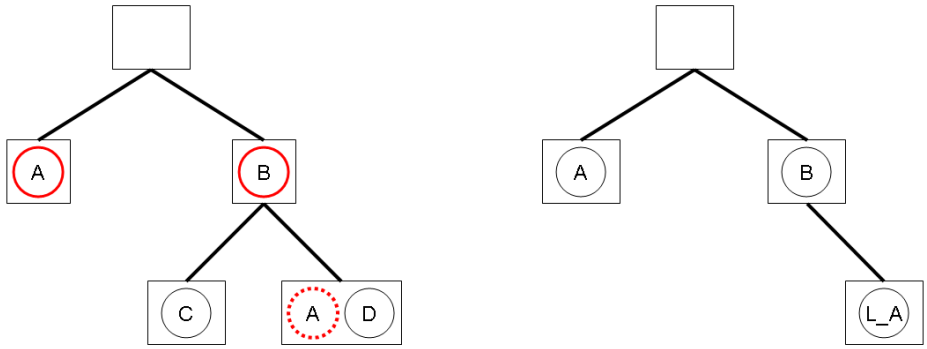


FIG. 6.3 – Représentation simplifiée d'un arbre d'observation et d'un pattern. L'utilisateur a indiqué sur l'arbre d'observation que les lettres majuscules A et B, situées dans les deux containers fils du container racine, étaient significatives en tant que telles dans l'expression d'un sens, ainsi que la lettre A, située à un niveau directement inférieur, mais cette fois en tant qu'instance du concept $Lettre\_A$ (la lettre en minuscule a serait donc également acceptée). De ces indications, données simultanément ou de façon incrémentale par l'utilisateur, l'agent alter ego en construit un pattern.

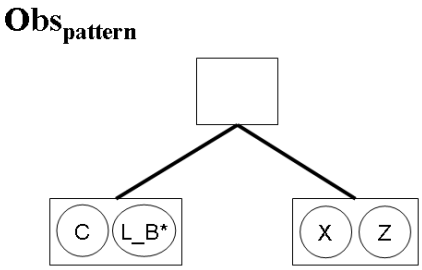
6.2.2 Similarité entre arbres d'observation de trace appariés à un pattern

Nous allons maintenant introduire une mesure de similarité entre deux arbres d'observation de trace, sous condition de leur appariement respectif avec un pattern. Cette mesure est la point de départ pour la mesure de similarité entre l'épisode en cours et un épisode antérieur, élaborés avec une signature, qui est nécessaire à l'étape de recherche d'épisode du RàPET.

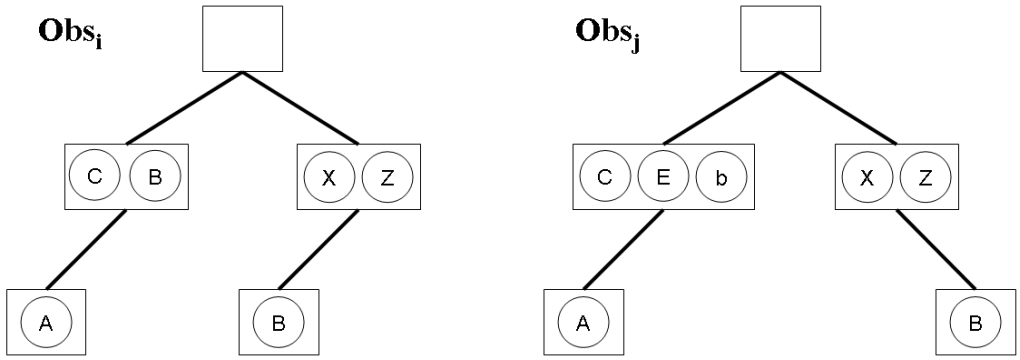
Nous allons nous appuyer sur un exemple pour présenter cette mesure de similarité. Considérons le pattern décrit ci-après, noté $Obs_{pattern}$, dans lequel est présent une variable (L_B^*) pouvant être appariée avec les mots clés "B" et "b" présents dans deux arbres d'observations de trace Obs_i et Obs_j (décrits en dessous de $Obs_{pattern}$). Ce pattern contient également des captures constantes représentant les OI "C", "X" et "Z". La hiérarchie de containers de ce pattern est la suivante : un container racine (vide), et deux containers fils (de droite et de gauche) considérés distinctement lors de l'appariement. $Obs_{pattern}$ est donc appariable à Obs_i et à Obs_j.

On a donc :

– un pattern $Obs_{pattern}$, avec une capture variable L_B^* :

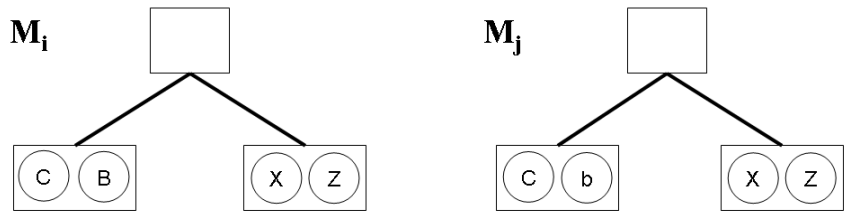


– deux arbres d'observation de traces, Obs_i et Obs_j, appariables à $Obs_{pattern}$:

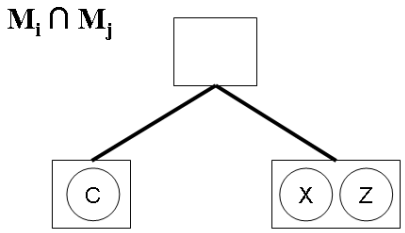


Nous noterons par $\overline{\cap}$ l'opération d'appariement d'un pattern $Obs_{pattern}$ sur un arbre d'observation de trace Obs. Sur cet exemple, introduisons les ensembles suivants, fonction d'une solution d'appariement du pattern à chaque arbre d'observation :

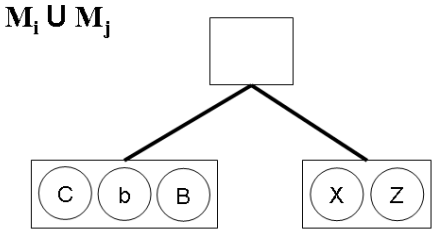
– Parties d'observation de trace appariées : $M_i = Obs_i \overline{\cap} Obs_{pattern}$ (avec $L\_B^* \leftarrow B$) et $M_j = Obs_j \overline{\cap} Obs_{pattern}$ (avec $L\_B^* \leftarrow b$) :



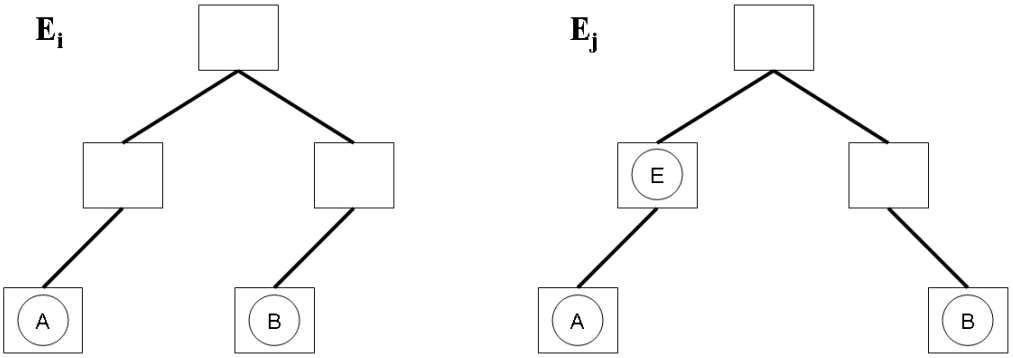
– Intersection entre M_i et M_j : $M_i \cap M_j$.



– Union de M_i et M_j : $M_i \cup M_j$



– Parties d'observation de trace non appariées : $E_i = Obs_i - M_i$ et $E_j = Obs_j - M_j$



En adoptant le principe de similarité cognitive de [Tversky 77], abordé en 3.2.2, selon lequel une similarité entre deux ensembles A et B se ramène à $sim(A,B) = \frac{f(A \cap B)}{f(A \cup B)}$, nous définissons la similarité entre deux arbres d' observations Obs_i et Obs_j sous la condition d'un appariement avec un arbre d'observation de pattern $Obs_{pattern}$, avec pour paramètres de pondération $p_{pattern}$ et p_E, par :

$$sim_{Obs_{pattern}}(Obs_i, Obs_j) = \frac{p_{pattern} \cdot sim_{Obs_{pattern}}(M_i, M_j) \cdot dim(Obs_{pattern})}{p_{pattern} \cdot dim(Obs_{pattern}) + p_E \cdot (dim(E_i) + dim(E_j))} \quad (6.2)$$

où, en incluant l'équation 6.1 :

$$sim_{Obs_{pattern}}(M_i, M_j) = \frac{dim(M_i \cap M_j) + \sum_{v \in variables(Obs_{pattern})} sim(m_i(v), m_j(v))}{dim(M_i \cup M_j) - nombre\_de\_variables(Obs_{pattern})} \quad (6.3)$$

et avec la dimension d'une observation qui est définie par :

$$dim(Obs) = nombre\ de\ captures\ d'OI\ (constantes\ ou\ variables)\ dans\ Obs \quad (6.4)$$

où $m_i(v)$ et $m_j(v)$ représentent respectivement l'appariement d'une variable de pattern v avec une capture de Obs_i et Obs_j, tel que cela a été introduit ci-dessus.

Nous avons introduit les paramètres de pondération $p_{pattern}$ et p_E, dans la définition de $sim_{Obs_{pattern}}(Obs_i, Obs_j)$, dans l'optique de laisser un rôle prépondérant à la partie appariée au pattern dans la comparaison de deux observations ; ceci est d'autant plus important que le pattern est de faible dimension. D'un autre côté, il ne convient pas d'annuler p_E, sinon deux observations distinctes pourraient être de similarité égale à 1, ce qui est cognitivement discutable ; de plus, nous considérons que cela serait dans le principe excessif compte tenu du fait que, dans notre approche, les symboles sont voués à évoluer.

Exemple : sur la base des exemples d'arbre d'observation Obs_i et Obs_j, et du pattern $Obs_{pattern}$ donnés ci-dessus, nous allons illustrer la mesure de similarité $sim_{Obs_{pattern}}(Obs_i, Obs_j)$ en effectuant son calcul. Nous posons $sim(B, b) = sim(match_i(L\_B^*), match_j(L\_B^*)) \approx 0,83$, tel que cela peut être obtenu avec le premier exemple de modèle de domaine donné dans la section précédente.

On a :

$$sim_{Obs_{pattern}}(M_i, M_j) = \frac{3 + 0,83}{5 - 1} \approx 0,96$$

car :

$$\begin{aligned} dim(M_i \cap M_j) &= 3 \\ dim(M_i \cup M_j) &= 5 \\ nombre\_de\_variables(Obs_{pattern}) &= 1 \end{aligned}$$

D'où, avec par exemple $p_{pattern} = 5$ et $p_E = 1$:

$$sim_{Obs_{pattern}}(Obs_i, Obs_j) = \frac{5 \cdot 0,96 \cdot 4}{5 \cdot 4 + 1 \cdot (2 + 3)} \approx 0,77$$

car :

$$\begin{aligned} dim(Obs_{pattern}) &= 4 \\ dim(E_i) &= 2 \\ dim(E_j) &= 3 \end{aligned}$$

6.2.3 Similarité entre patterns

Dans notre transposition des mécanismes d'émergence de langage, détaillée au chapitre 7, nous introduirons un mécanisme de négociation, entre l'utilisateur et l'agent alter ego, pour désambiguïser le sens des indications données par l'utilisateur. Dans ces mécanismes, nous aurons besoin d'évaluer la similarité entre deux patterns, afin de :

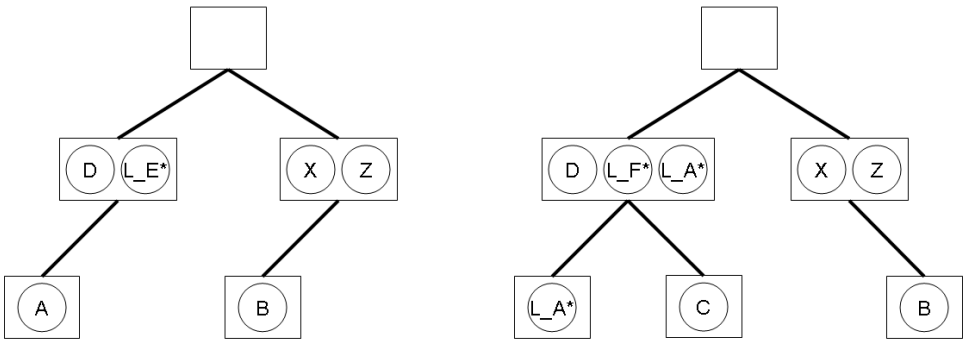
1. savoir si deux patterns (deux indications données par l'utilisateur à des instants différents) sont identiques, i.e., de similarité égale à 1, et ;

2. proposer (sous forme de suggestion) des patterns similaires à l'utilisateur.

Nous allons définir notre mesure de similarité à l'aide d'un exemple de deux patterns Obs_i^* et Obs_j^*, donné ci-dessous. Pour faciliter la lecture, nous utilisons la même représentation simplifiée que dans la section précédente :

- les containers candidats sont représentés par des boîtes rectangulaires et ils ne sont appariables que s'ils sont à la même position par rapport à la racine, et ;

- les captures sont représentées directement par les OI qu'ils désignent, les variables sont notées avec une étoile.

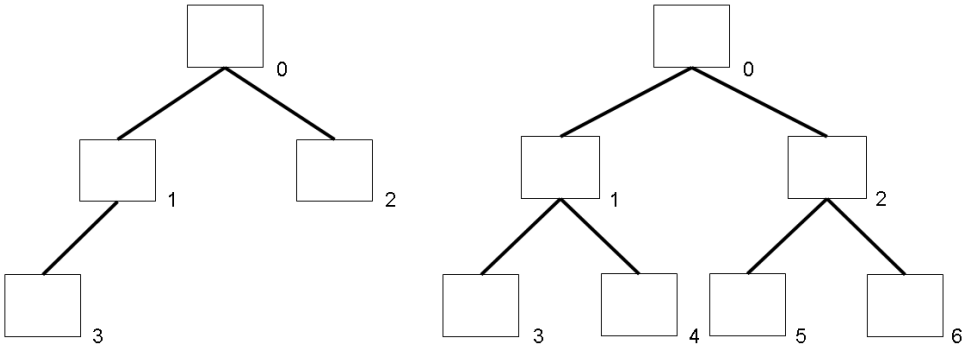


Obs*i **Obs*j**

Notre principe de similarité suit toujours les principes de [Tversky 77], et nous introduisons pour cela les arbres de containers suivants :

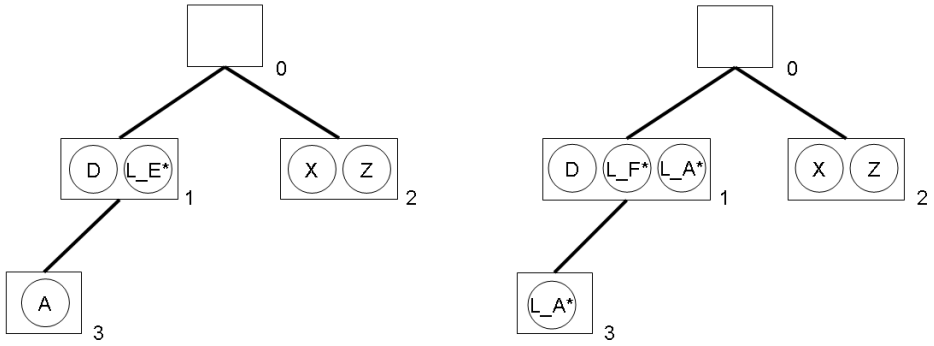
- I, plus grand sous-arbre commun aux arbres de containers de Obs_i^* et Obs_j^* ;
- U, l'union des arbres de containers de Obs_i^* et Obs_j^*.

Une représentation schématique de I et de U est donnée ci-dessous. Afin de les désigner facilement, les containers sont numérotés.



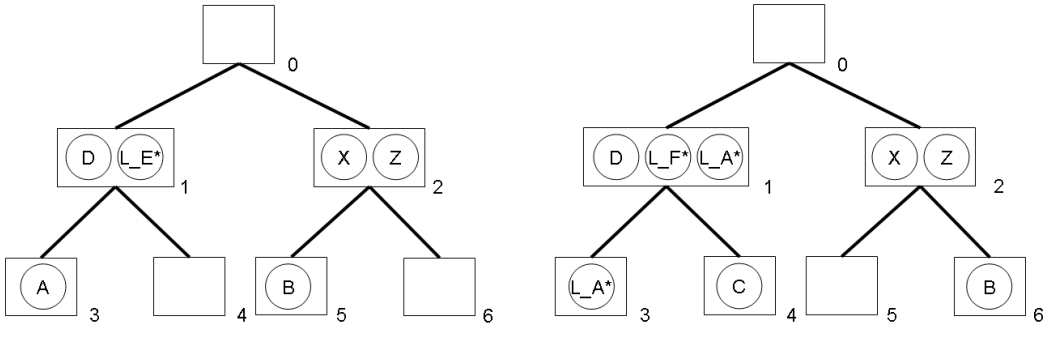
I **U**

L'utilisation de I comme un masque sur Obs_i^* et Obs_j^* nous permet d'introduire respectivement les arbres d'observation I_i et I_j, dont la représentation est donnée ci-dessous.



I_i **I_j**

L'utilisation de U comme un masque sur Obs_i^* et Obs_j^* dans U nous permet d'introduire respectivement les arbres d'observation U_i et U_j, dont la représentation est donnée ci-dessous.



U_i **U_j**

En s'inspirant des principes de similarité de [Tversky 77], une similarité entre deux patterns Obs_i^* et Obs_j^* peut être vue comme un rapport entre la "dimension" des parties *similaires*, qui

sont obtenues à l'aide du masque I, et la "dimension" de la *totalité* qui est obtenue avec le masque U. Cette extension aux parties *similaires*, en plus des parties *identiques*, vient du besoin de comparer les variables entre elles.

Pour justifier cela, considérons un container c de Obs_i^* et un container c' de Obs_j^* à la même position dans le masque I ; soit v une variable de c, non identique à une variable v' de c' : la similarité sémantique entre v et v' peut être proche de 1, et la mesure de similarité entre pattern doit tenir compte de cela. En effet, dans le cas limite où les patterns Obs_i^* et Obs_j^* sont réduits à un seul container avec respectivement v et v' pour unique capture, la partie *identique* serait vide et, sans la prise en compte de la similarité entre v et v', la similarité entre Obs_i^* et Obs_j^* serait aussi nulle, bien que v et v' soient sémantiquement très proches. Le processus de négociation de sens, détaillé au chapitre suivant, demandera à l'agent alter ego d'être en mesure de proposer des patterns similaires à l'utilisateur, c'est pourquoi nous avons choisi de tenir compte de la similarité sémantique entre les variables non identiques dans le calcul de similarité entre patterns.

Sur la base de ces principes, nous définissons la similarité entre deux patterns Obs_i^* et Obs_j^* par :

$$\begin{aligned} sim(Obs_i^*, Obs_j^*) &= \frac{Card(similaire)}{Card(totalité)} \\ &= \frac{Card(identique) + Card(approx.\ variables)}{Card(totalité)} \end{aligned} \tag{6.5}$$

où, en s'appuyant sur les arbres introduits ci-dessus :

$$\begin{aligned} Card(identique) &= \sum_{(c,c')\in I_i\times I_j} |C_i \cap C_j| + |V_i \cap V_j| \\ Card(approx.\ variables) &= \sum_{(c,c')\in I_i\times I_j} \left(\sum_{v\in V_i - V_i\cap V_j} \max_{v'\in V_j - V_i\cap V_j} sim(v,v') \right) \\ Card(totalité) &= \sum_{(c,c')\in U_i\times U_j} |C_i \cup C_j| + |V_i \cup V_j| \end{aligned}$$

et avec :

- en notant $\sum_{(c,c')\in I_i\times I_j}$, respectivement $\sum_{(c,c')\in U_i\times U_j}$, nous exprimons que la somme est effectuée en parcourant les paires de containers (c, c') qui se correspondent dans I_i et I_j, respectivement U_i et U_j. Les ensembles C_i, C_j, V_i et V_j sont fonction d'une telle paire de containers ;
- C_i est l'ensemble des constantes d'un container c de I_i ou de U_i. De même, C_j est l'ensemble des constantes d'un container c' de I_j ou U_j ;
- V_i est l'ensemble des variables d'un container c de I_i ou de U_i. De même, V_j est l'ensemble des variables d'un container c' de I_j ou U_j ;
- pour la somme sur $(c, c') \in I_i \times I_j$ de $Card(approx.\ variables)$, nous mettons la condition que pour chaque paire de containers traitée, on a : $|V_i - V_i \cap V_j| < |V_j - V_i \cap V_j|$, sinon les rôles de i et de j sont intervertis.

Cette condition est mise afin que la somme des similarités maximales pour chaque paire de containers soit majorée par le cardinal du plus petit des deux ensembles de variables non identiques ($V_i - V_i \cap V_j$ et $V_j - V_i \cap V_j$). v et v' sont des variables de l'un et l'autre de ces ensembles.

Nous précisons que cette mesure de similarité ne peut être mise en oeuvre que si les containers sont munis de labels distincts, sans quoi la détermination de I et U est impossible.

La composante $\sum_{v \in V_i - V_i \cap V_j} \max_{v' \in V_j - V_i \cap V_j} sim(v, v')$ fait appel à une mesure de similarité entre concepts d'abstraction, qui est propre à la modélisation du domaine (c.f., 6.1).

Exemple : pour illustrer le calcul de similarité entre patterns, considérons les deux patterns Obs_i^* et Obs_j^* décrits schématiquement ci-dessus. Les abstractions (captures variables) sont faites dans le cadre du deuxième modèle de domaine, pour représenté le "monde des lettres", donné en exemple dans la section 6.1.

Au numérateur, c'est à I que nous nous intéressons. Nous avons pour le container :

- numéro 0, c'est la racine, elle est vide. Un container vide n'a aucune influence sur la valeur de similarité ;
- numéro 1,
  - $|C_i \cap C_j| = |\{D\}| = 1$,
  - $|V_i \cap V_j| = |\phi| = 0$ et
  - $\sum_{v \in \Omega_i} \max_{v' \in \Omega_j} sim(v, v') = max(sim(L\_E*, L\_F*), sim(L\_E*, L\_A*))$
    $= sim(L\_E*, L\_A*) = 0,25$, avec $\Omega_i = V_i - V_i \cap V_j$ et $\Omega_j = V_j - V_i \cap V_j$
- numéro 2, $|C_i \cap C_j| = |\{X, Z\}| = 2$;
- numéro 3, $|C_i \cap C_j| = |V_i \cap V_j| = |\phi| = 0$.

Pour le calcul du dénominateur, nous nous intéressons à U. Nous avons pour le(s) container(s) :

- numéro 0, rien car la racine est vide ;
- numéro 1, $|C_i \cup C_j| = |\{D\}| = 1$ et $|V_i \cup V_j| = |\{L\_E*, L\_F*, L\_A*\}| = 3$;
- numéro 2, $|C_i \cup C_j| = |\{X, Z\}| = 2$;
- numéro 3, $|C_i \cup C_j| = |\{A\}| = 1$ et $|V_i \cup V_j| = |\{L\_A*\}| = 1$;
- numéros 4, 5 et 6, $|C_i \cup C_j| = |\{C\}|_{pour\ 4} = |\{B\}|_{pour\ 5\ et\ 6} = 1$ et $|V_i \cup V_j| = |\phi| = 0$.

D'où :

$$sim(Obs_i^*, Obs_j^*) = \frac{1 + 0,25 + 2}{1 + 3 + 2 + 1 + 1 + 3 \cdot 1} \approx 0,30$$

Notons que pour les containers numéros 3 de I et U, le calcul de similarité ne tient pas compte du fait que $A*$ est une généralisation de A. Dans notre cas, lors de la négociation de sens présentée au chapitre suivant, nous devons absolument être capable d'évaluer si la similarité de deux patterns vaut 1, i.e., s'ils sont égaux. Notre définition de la similarité entre patterns permet cela ; son approximation dans le cas des épisodes similaires (< 1) influe sur la pertinence des suggestions faites par l'agent alter ego.

6.3 Représentation de l'expérience tracée

Le modèle Musette introduit la notion de signature expliquée de tâche comme un motif qui doit être reconnu dans une trace d'utilisation, afin d'être en mesure de construire un épisode cible à partir de la trace en cours (étape d'*élaboration* du cycle de RàPET) et de le rapprocher d'épisodes sources élaborés dans des traces antérieures. Néanmoins, aucune direction particulière n'est adoptée quant à la nature de ce motif, et comme nous l'avons souligné en 4.5, sa capture est une problématique en soi conditionnant la pertinence de l'assistance de l'agent alter ego.

Nous avons introduit, en 6.2, que notre approche pour acquérir ces signatures de tâche sera basée sur la transposition des mécanismes d'émergence de langage. Pour effectuer cela, nous avons besoin d'une représentation symbolique des traces, que nous allons présenter dans cette section. Cette approche peut sembler contraignante, mais elle facilite la lecture d'une trace par l'acteur humain, et cet aspect réflexif constitue en soi un support à son activité méta-cognitive.

La section 6.4 présentera notre modèle symbolique pour les signatures de tâche, qui est également nécessaire à notre transposition.

Dans le modèle Musette, une trace est une succession alternée d'états et de transitions, auxquels sont associés des observations (pour nous, des arbres d'observation) ; sous cette forme, nous l'appelons *trace primitive*.

De façon intuitive, nous choisissons d'axer la représentation symbolique d'une trace primitive à l'utilisateur sur les appels qu'il a faits aux fonctionnalités de l'application cible, auxquels sont associées les variations respectives des états de cette application.

Toutefois, considérons un acteur désirant effectuer un travail en interagissant avec des objets d'intérêt. Pour faire cette action, il adopte une démarche qui lui est propre d'utilisation des fonctionnalités de l'environnement informatique, tel que cela est généralement possible dans les outils informatiques actuels. Par conséquent, la trace d'utilisation a une forte variabilité si nous considérons un acteur effectuant la même opération de différentes façons. Cette variabilité constitue une difficulté pour la manipulation des traces ainsi qu'une limite pour leur comparaison. Nous avons choisi d'orienter la représentation symbolique d'une trace autour des fonctionnalités de l'application cible appelées par l'utilisateur, qui sont les opérateurs de base mis à disposition de l'utilisateur pour réaliser des actions plus complexes. De façon générale, une application cible impose à l'utilisateur certaines opérations, et lui laisse la possibilité d'en effectuer d'autres dans un ordre partiellement libre. Afin de pouvoir parcourir une trace d'utilisation en se servant de ces connaissances, nous devons disposer d'un langage de description exprimant l'utilisation d'une fonctionnalité ainsi que les relations qui existent entre elles. Pour cela, et avec l'intuition qu'une trace constitue un "récit" des opérations effectuées par l'utilisateur dans le contexte de l'application cible, nous proposons ci-dessous une description à base de langage formel pour la

structure de trace. L'appel à une fonctionnalité est représenté par un *marqueur* présent dans la transition associée. L'ensemble de ces marqueurs sont des objets d'intérêt supplémentaires.

La section 6.3.1 suivante présente les règles syntaxiques de construction d'une trace primitive intégrant le modèle de fonctionnalité de l'application cible, via l'ajout des marqueurs. Nous utiliserons une grammaire formelle dont les règles expriment les connaissances de l'application cible, dans le but de décrire en situation la variabilité opératoire dont dispose l'utilisateur. Toutefois, comme nous l'illustrerons sur un exemple, il n'est pas possible d'utiliser directement les propriétés des opérateurs, car les observations contenues dans les états sont temporellement ordonnées et leur permutation ne fait pas toujours sens. Notre utilisation d'une grammaire formelle est faite dans un but uniquement descriptif. A l'aide de cette description, un filtrage de la trace peut être effectué afin d'extraire les actions que nous souhaitons considérer pour l'élaboration ; le besoin d'une telle opération sera discuté en 8.1.1.

Dans le travail présenté ici, nous n'utiliserons pas toutes ces connaissances. Elles constituent néanmoins la base pour enrichir l'expressivité d'une signature par l'ajout de contraintes d'ordre fonctionnel, tel que nous le présenterons en perspective, c.f., 10.4.

Ce niveau syntaxique, guidant la construction de la trace primitive, est introduit pour permettre la représentation symbolique d'une trace. La section 6.3.2 détaille cette opération de représentation symbolique ; elle est faite en intégrant les éléments d'élaboration acquis par l'agent alter ego suite à des indications données par l'utilisateur (et dont la réutilisation lui a déjà suffisamment fait sens) à un niveau symbolique : ce sont les symboles que nous avons introduits en 6.2. Une représentation symbolique d'une trace est donc, par nature, fonction de l'état de langage dans lequel ces symboles sont définis, et elle fait appel à leur perception, i.e., l'appariement de leur motif.

Les points que nous allons présenter dans la section suivante ont été publiés dans [Stuber 03]. Toutefois, nous allons détailler les mécanismes qui permettant de prendre en compte des actions menées conjointement et qui sont représentées chacunes par une phrase distincte de notre grammaire formelle.

6.3.1 Niveau syntaxique : grammaire formelle

Le modèle Musette définit une trace d'utilisation primitive comme une séquence alternée d'états et de transitions. Notre modèle de trace primitive présenté ici est posé dans le souci de respecter ce principe structurel. Nous allons présenter les règles syntaxiques qui permettent d'inclure des connaissances sur les fonctionnalités de l'application cible, sous forme de marqueurs, et de dotter ainsi la trace primitive d'une sémantique supplémentaire ; cette sémantique servira de guide à la reformulation symbolique d'une trace, détaillée dans la section suivante.

Les connaissances sur l'application sont définies dans un *modèle d'application*, qui vient s'inclure au modèle d'utilisation de l'approche Musette. Le modèle d'application est conçu dans le but de définir les entités fonctionnelles "de surface", qui sont mises à disposition de l'utilisateur, et qui par là sont en mesure de lui faire sens. Ce modèle d'application ne vise en aucun cas à être un modèle du fonctionnement interne de l'application cible. Il repose sur une décomposition des fonctionnalités en *actions*, où chaque action est réalisable via une série d'*opérations*. Ce sont ces opérations qui vont être représentées par un marqueur dans la trace, lors de leur appel par l'utilisateur.

Nous proposons de distinguer des actions (ayant un sens pour l'utilisateur) qui sont réali-

sées via une série d'opérations (ayant un sens pour l'utilisateur et l'agent alter ego). Pour les actions que nous considérons dans un environnement documentaire (édition, consultation), nous distinguons parmi les opérations possibles pour une action : les opérations d'***Initialisation***, les opérations *intermédiaires* et les opérations de ***Finalisation***. De plus, nous distinguons parmi les opérations intermédiaires, celles qui sont ***Statiques*** et celles qui sont ***Alternatives***, afin d'exprimer la variabilité opératoire qu'a l'utilisateur durant sa tâche.

Les interdépendances entre actions et opérations, ainsi que leurs conditions d'occurrence, ne sont pas définies dans notre modèle d'application ; l'application cible les impose à l'observation lors de son execution.

Intuitivement, une *séquence* est une portion de trace (une suite de transitions et d'états, avec l'ajout d'un marqueur) qui vient compléter la trace en cours, afin de traduire l'observation de la réalisation d'une opération par l'utilisateur. Une partie de trace correspondant à une action est ainsi composée : d'une *séquence initiale*, d'une suite (éventuellement vide) de séquences intermédiaires représentant l'utilisation de différentes fonctionnalités de cette action, et une *séquence finale*. Parmi les séquences intermédiaires, certaines représentent des opérations qui peuvent être réalisées dans un ordre indifférent, elles sont appelées *séquences alternatives* ; d'autres expriment des opérations au caractère fixe, elles sont appelées *séquences statiques*.

Ci-dessous la définition, sur la base d'un langage formel, de la structure de trace primitive pour décrire la réalisation d'une action. Elle intègre l'ajout des marqueurs, définis dans le modèle d'application :

- **Phrase :** une phrase est une portion de trace primitive représentant la réalisation d'une suite d'actions. Si des actions sont menées conjointement par l'utilisateur, dans plusieurs fenêtres, une phrase est construite pour chaque fenêtre. Nous verrons dans le paragraphe suivant comment ces phrases sont entrelacées pour construire une *trace*.

- **Symboles terminaux :** les transitions (T) et les états (E) sont considérés comme des symboles terminaux. Parmi eux T_I, T_A, T_S, et T_F représentent des transitions contenant respectivement les marqueurs pour la séquence *initiale*, les séquences *alternatives*, les séquences *statiques* et les séquences *finales* d'une action. Un arbre d'observation est associé à chaque état et à chaque transition ;

- **Opérateurs :** les opérateurs sont $\cdot$ et $+$. Une suite d'états et de transitions séparés uniquement par des $\cdot$ est indivisible, la notion de *séquence* est ainsi formellement définie.
  Pour l'opérateur $+$, qui est utilisé entre des séquences, sa commutativité entre séquences est restreinte à une suite ininterrompue de séquences alternatives ; la présence d'une séquence statique rend la commutation impossible.

- **Symboles non terminaux :** une *phrase* est une suite d'actions. une *action* est constituée d'une séquence initiale ($Sequence_I$), d'une suite éventuellement nulle de séquences intermédiaires ($IntermSeq$) et d'une séquence finale ($Sequence_F$). La suite de séquences intermédiaires peut être composée de séquences permutables ($PermSeq$), de séquences statiques ($Sequence_S$) et/ou d'actions imbriquées. Les séquences permutables sont une somme de séquences alternatives ($Sequence_A$), qui peuvent être permutées entre elles car elles constituent une suite ininterrompue.

– **Règles de production :**

$$
\begin{aligned}
Phrase &::= Action^* \\
Action &::= Sequence_I\ [\ +\ IntermSeq]^* + Sequence_F \\
IntermSeq &::= PermSeq \mid Sequence_S \mid Action \\
PermSeq &::= Sequence_A \mid PermSeq + Sequence_A \\
Sequence_x &::= T_x \cdot E\ [\ \cdot T \cdot E]^* \quad avec\ \ x \in \{I, A, S, F\}
\end{aligned}
$$

La forme $T_x \cdot E\ [\ \cdot T \cdot E]^*$ pour une séquence a été introduite pour représenter une opération s'effectuant en plusieurs étapes, tel que cela était le cas dans la maquette SimManager développée par *PCO Technologies*. Une alternative est de décrire ces opérations en plusieurs étapes comme des actions imbriquées ; cela peut permettre d'affiner la description, en rajoutant par exemple les retours en arrière de l'utilisateur, mais une évolution de la grammaire peut s'avérer nécessaire pour exprimer le caractère alternatif ou statique de ces actions imbriquées. Dans la suite, nous considérons que toutes les séquences sont de la forme $T \cdot E$.

Le respect de cette grammaire, lors de la description d'une action, est assuré par le module d'acquisition de l'agent alter ego. Nous soulignons qu'une phrase est grammaticalement correcte seulement lorsque l'action qu'elle décrit est terminée. La question de la correction grammaticale est donc secondaire, car la *trace en cours* est par nature incorrecte, dans la mesure où elle décrit des actions en cours. Ce sont les propriétés descriptives d'une grammaire formelle qui nous intéressent.

Exemple de phrase : la figure 6.4 représente la phrase obtenue pour décrire la réalisation d'une action d'édition. La première séquence représente l'ouverture du document, les deux suivantes sont des séquences permutables représentant la saisie successive de deux OI (un cercle, et un triangle) et la dernière séquence représente la cloture du document. Les arbres d'observation sont simplifiés et se limitent à un seul nœud container.

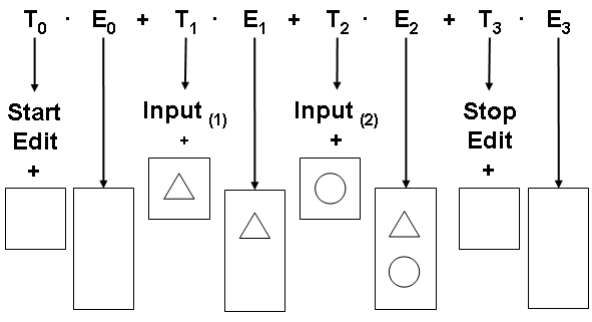


Fig. 6.4 – Exemple de phrase pour une action d'édition

La figure illustre l'impossibilité de permuter simplement les deux séquences de saisie, car les observations associées à E_1 et E_2 ne font plus sens.

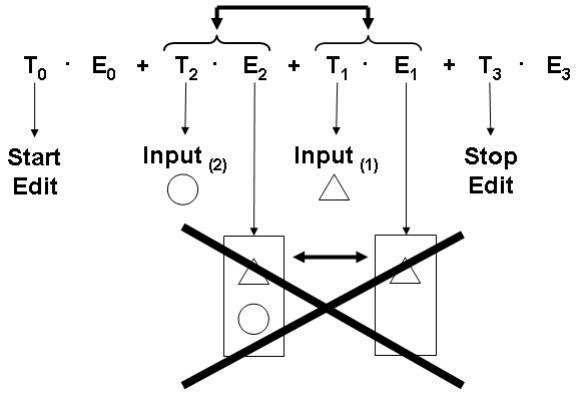


FIG. 6.5 – Illustration de l'impossibilité de permuter des séquences de phrase

Prise en compte des actions menées conjointement

Cette grammaire permet de représenter les enchaînements d'actions et les actions imbriquées, mais elle ne permet pas de représenter des actions pouvant être menées conjointement par l'utilisateur, tel que cela est généralement possible dans les outils actuels disposant d'une interface graphique. Ces outils permettent d'effectuer plusieurs actions dans diverses fenêtres, et l'utilisateur les réalise en basculant d'une fenêtre à l'autre. Cet événement (prise de *focus*) est un événement de base des environnements graphiques, et nous choisissons de le représenter directement dans la trace pour faire le lien entre des actions menées conjointement, décrites chacune par une phrase, selon la grammaire formelle définie ci-dessus. Cet événement de prise de focus vient se rajouter au modèle d'application sous la forme d'un marqueur.

Nous allons présenter l'ajout de ce marqueur de prise de focus dans la construction d'une trace en nous appuyant sur l'exemple de session donné sur la figure 6.6.

L'utilisateur initie cette session par une opération de *login* et il la termine par une opération de *logout* ; ces deux opérations composent une action qui englobe, sous forme d'imbrications, toutes les autres : c'est ce qui constitue une trace. Les fonctionnalités de l'application, qui sont représentées par des actions, peuvent s'imbriquer jusqu'à ce que l'utilisateur ait la possibilité de mener plusieurs actions conjointement. Afin de respecter la forme générale d'une phrase, i.e., une alternance d'états et de transitions, la prise de focus est représentée dans la trace en ajoutant une paire $T_{Focus} \cdot E_{cible}$, où E_{cible} représente l'état de l'action cible du focus. Si les actions sont indépendantes, l'état E_{cible} est le même que le dernier état de l'action cible lors de sa perte de focus ; toutefois, comme nous l'illustrerons dans l'exemple ci-dessous, cela n'est pas toujours le cas et une nouvelle capture est nécessaire.

Exemple de trace : la figure 6.6 décrit schématiquement une session d'utilisation, telle qu'elle peut se dérouler dans le démonstrateur présenté au chapitre suivant. L'utilisateur se connecte à l'application cible (login), puis il ouvre un dossier de remboursement dossier administratif de remboursement de frais de mission. Lors de la consultation d'un dossier, l'utilisateur peut effec-

tuer plusieurs actions sur les formulaires (des documents) de ce dossier en même temps, ainsi, il édite un formulaire tout en consultant un autre. Lorsque ces deux actions sont terminées, l'utilisateur ferme le dossier et se déconnecte.

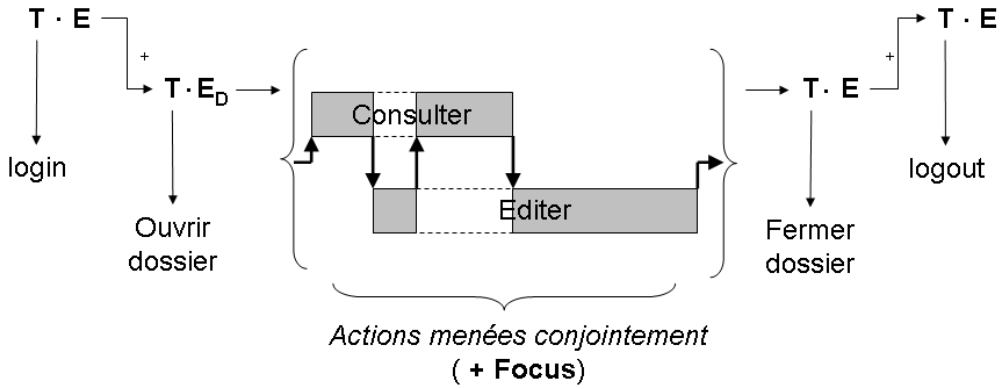


FIG. 6.6 – Structure de la trace correspondant à une session de l'utilisateur

La figure 6.7 donne les séquences composant les phrases des actions menées conjointement, et précise leur entrelacement. Les flèches noires représentent les séquences $T_{Focus} \cdot E$ qui doivent être ajoutées pour représenter l'alternance entre ces deux actions.

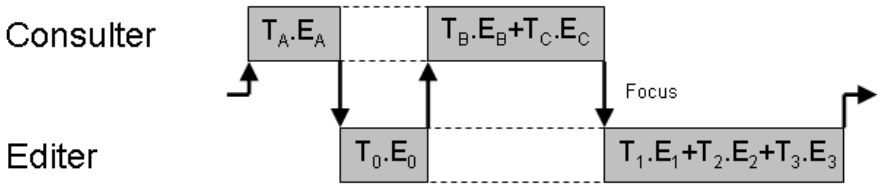


FIG. 6.7 – Description de l'entrelacement des phrases pour les actions menées conjointement

La figure 6.8 met en évidence les marqueurs associés à chacune des séquences composant les deux actions. Les phrases sont présentées séparément. Nous rappelons que des arbres d'observations sont associés aux états et aux transitions de ces phrases, mais nous ne les représentons pas.

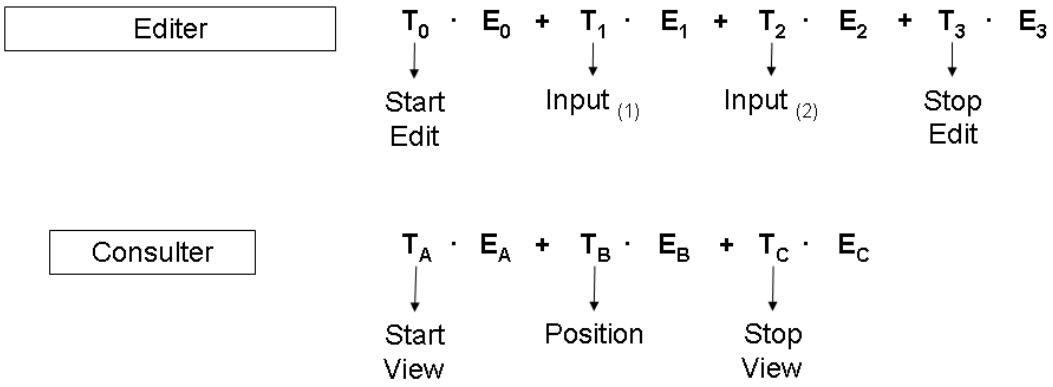


FIG. 6.8 – Détail de la description des actions menées conjointement

La trace représentant la réalisation des actions simultanées est décrite sur la figure 6.9. Elle vient s'intercaler dans l'exemple schématique d'une trace complète présenté auparavant.

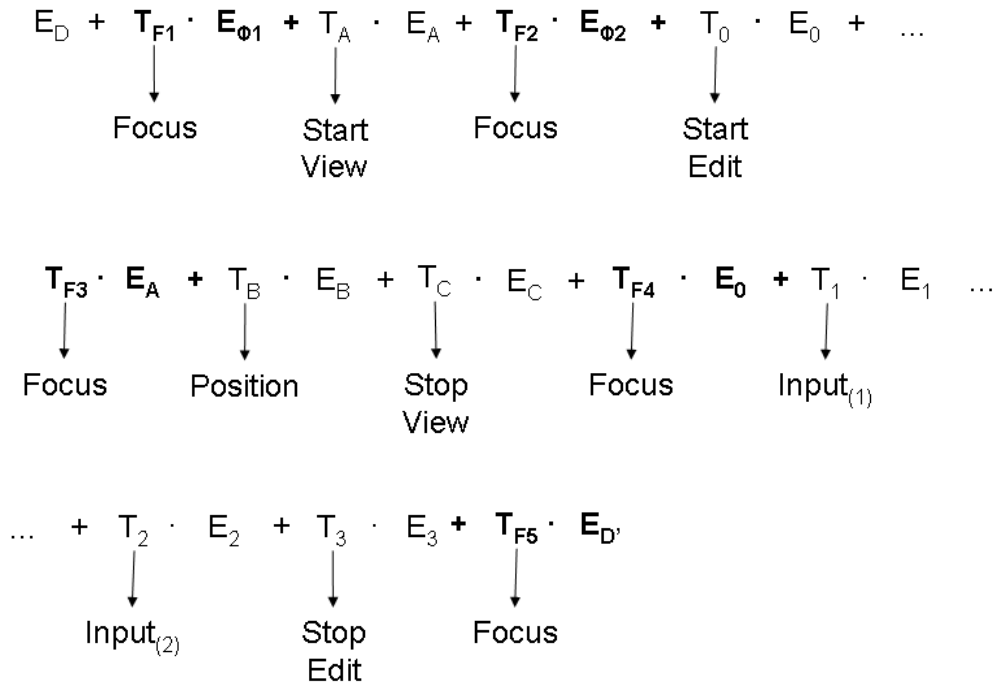


FIG. 6.9 – Trace correspondant aux actions menées conjointement, et venant s'intercaler dans la trace de la figure 6.6

Des séquences de prise de focus, de la forme $T_{Focus} \cdot E$, sont ajoutées à la place de chacune des flèches noires, introduites sur la figure 6.8, afin de représenter les allers-retours entre l'édition et la consultation. Pour les deux premières, il est chaque fois créé un état vide (E_{ϕ_1} ou E_{ϕ_1}) afin de respecter la structure d'alternance d'états et de transitions. Ces états vides n'ont pas d'utilité directe car ils sont situés avant l'opération initiale d'une action, tout comme les états

situés après les opérations finales d'une action (par exemple, E_3 et E_C). Dans des évolutions de ce modèle, ces états peuvent servir de conteneurs de connaissances supplémentaires, par exemple sur l'origine d'une action pour l'état initial.

Cet exemple de trace pour les actions simultanées illustre bien la problématique des actions indépendantes ou non entre elles. En effet, les deux actions simultanées sont indépendantes techniquement, par conséquent lors de leurs reprise de focus (T_{F3} pour la consultation, et T_{F4} pour l'édition) ce sont des copies des états lors de la perte de focus qui sont effectuées ; ainsi, on a $T_{F3} \cdot E_A$ pour la consultation, et $T_{F4} \cdot E_0$ pour l'édition. Pour le dossier, qui a été ouvert avant les actions simultanées, il est décrit initialement dans E_D ; mais après ces actions, il est dans un nouvel état $E_{D'}$, car les actions l'ont fait évoluer. Les actions d'édition ne sont pas indépendantes de l'ouverture d'un dossier. Cet aspect doit être considéré dès la conception afin de capturer des traces dont la contextualisation est valide, i.e., à jour.

6.3.2 Niveau symbolique et sémantique

Le modèle MUSETTE définit une trace d'utilisation primitive comme une séquence alternée d'états et de transitions. La grammaire formelle présentée ci-dessus respecte cette définition. De plus, elle ajoute des connaissances fonctionnelles de l'application cible (via les marqueurs d'opération) et des mécanismes de prise en compte des actions menées conjointement (via le marqueur de prise de focus).

Pour transformer une trace primitive en une trace symbolique, un processus de segmentation doit être effectué, afin d'identifier les parties de trace qui vont être reformulées symboliquement. Pour cela, nous appliquons une fenêtre "de lecture" sur la trace, que nous appelons une *section* de trace. Une section de trace englobe :

1. l'état de l'application avant que l'utilisateur n'effectue une opération. Cet état est décrit par un arbre d'observation ;

2. une transition à laquelle est associée un marqueur de fonctionnalité de l'application, et à la transition est associé un arbre d'observation correspondant aux paramètres de l'appel à cette fonctionnalité ;

3. l'état de l'application après l'appel de cette fonctionnalité. Cet état est décrit par un arbre d'observation.

Dans cette définition, nous avons appliqué l'hypothèse simplificatrice, faite dans la section précédente et valant pour la suite de ce mémoire : les séquences sont toutes de forme $T \cdot E$; il en résulte que les sections sont toutes de la forme $E \cdot T \cdot E$. Une *section* de trace primitive correspond à une séquence avec en plus l'état précédent ; ainsi, deux sections consécutives partagent l'état final de la première et l'état initial de la seconde.

Cependant, un sens partagé par l'acteur et l'agent alter ego doit être associé à chaque section extraite à l'aide de cette grammaire formelle, de telle manière qu'une reformulation symbolique proposée par l'agent alter ego fasse sens à l'acteur humain. Nous définissons dans ce paragraphe les éléments pour obtenir une représentation symbolique d'une trace : les *symboles génériques* et les *symboles de langage*. Ils seront ensuite utilisés pour exprimer la sémantique d'une élaboration

de trace, sur la base d'une expression symbolique des signatures. Le principe d'un symbole est de faire le lien bidirectionnel entre 1) un motif à reconnaître dans une section et 2) le sens humain de cette reconnaissance, défini en langage naturel par l'utilisateur.

Les marqueurs introduits dans le modèle d'application sont présents dans la trace pour représenter l'appel à des différentes fonctionnalités de l'application cible. Nous avons argumenté que les fonctionnalités sont des éléments de sens pour l'utilisateur, c'est pourquoi nous introduisons un moyen pour les représenter symboliquement en tant que tel.

Pour exprimer simplement l'appel à une fonctionnalité, nous introduisons la notion de *symbole générique* σ_{G_i}. Le label d'un symbole générique est le nom de la fonctionnalité, fixé lors de la conception. Chaque marqueur définit ainsi un symbole générique, et ces symboles seront invariants : ceci constitue le sous-ensemble statique des symboles, qui ne peuvent pas être modifiés par l'acteur et qui, en raison de leur caractère trop général, ne sont pas exploitables pour exprimer une situation d'assistance (une signature).

Un unique marqueur est présent dans chaque section ; ainsi, le symbole générique associé à ce marqueur constitue la reformulation symbolique par défaut pour la section, si l'on ne considère que son marqueur, i.e., le motif est réduit au marqueur.

La reformulation symbolique par défaut d'une trace n'est porteuse d'aucun sens lié aux activités de l'utilisateur ; c'est une représentation purement fonctionnelle de la trace qui a un intérêt principalement réflexif. Dans l'introduction de notre approche faite en 6.2, nous avons indiqué que le but de l'agent alter ego est de réutiliser les indications données par l'utilisateur sur le caractère significatif des OI présents dans une certaine section de trace, afin de caractériser une situation d'assistance. Toutes ces indications sont décrites dans l'*état de langage en cours*, via un ensemble de symboles de langage.

Précisément, un *symbole* σ_i *de langage* est la paire constituée :

- d'un motif ; ce dernier a la même structure qu'une section de trace. Il contient le marqueur de la section dont il est issu, et des patterns correspondant à la transformation des indications de l'utilisateur données dans les arbres d'observation de la section, selon les principes détaillés sur un exemple en 6.2.1 ;
- d'un label, qui est défini en langage naturel par l'acteur lors de l'ajout d'indications, afin d'exprimer le sens (humain) qu'il donne à la perception du motif de ce symbole.

Le niveau symbolique est imposé par notre transposition des mécanismes d'émergence de langage détaillée en 4.6. Il représente le médium de communication entre l'utilisateur et l'agent alter ego pour discuter des élaborations.

Dans une logique de proactivité de l'agent alter ego, la reformulation symbolique d'une trace est enrichie par les symboles qu'il perçoit, i.e., ceux dont les motifs peuvent être appariés à une section. De la sorte, une trace primitive est reformulée en une trace symbolique, qui est définie par une suite d'ensembles de symboles identifiés pour chaque section. Un de ces ensembles est constitué d'au moins un symbole générique et éventuellement d'autres symboles de langage. La figure 6.10 illustre ces notions. Lors de la présentation d'une reformulation symbolique à l'utilisateur, dans les interfaces spécifiques que nous allons présenter, ce sont les labels des symboles qui seront utilisés pour lui indiquer le sens d'une section.

En adoptant cette définition de la trace symbolique, nous y intégrons l'opération de perception de symboles, qui est la première étape du mécanisme d'élaboration d'épisode présenté en 6.6.1 ; nous attirons l'attention qu'une reformulation symbolique est ainsi fonction des symboles

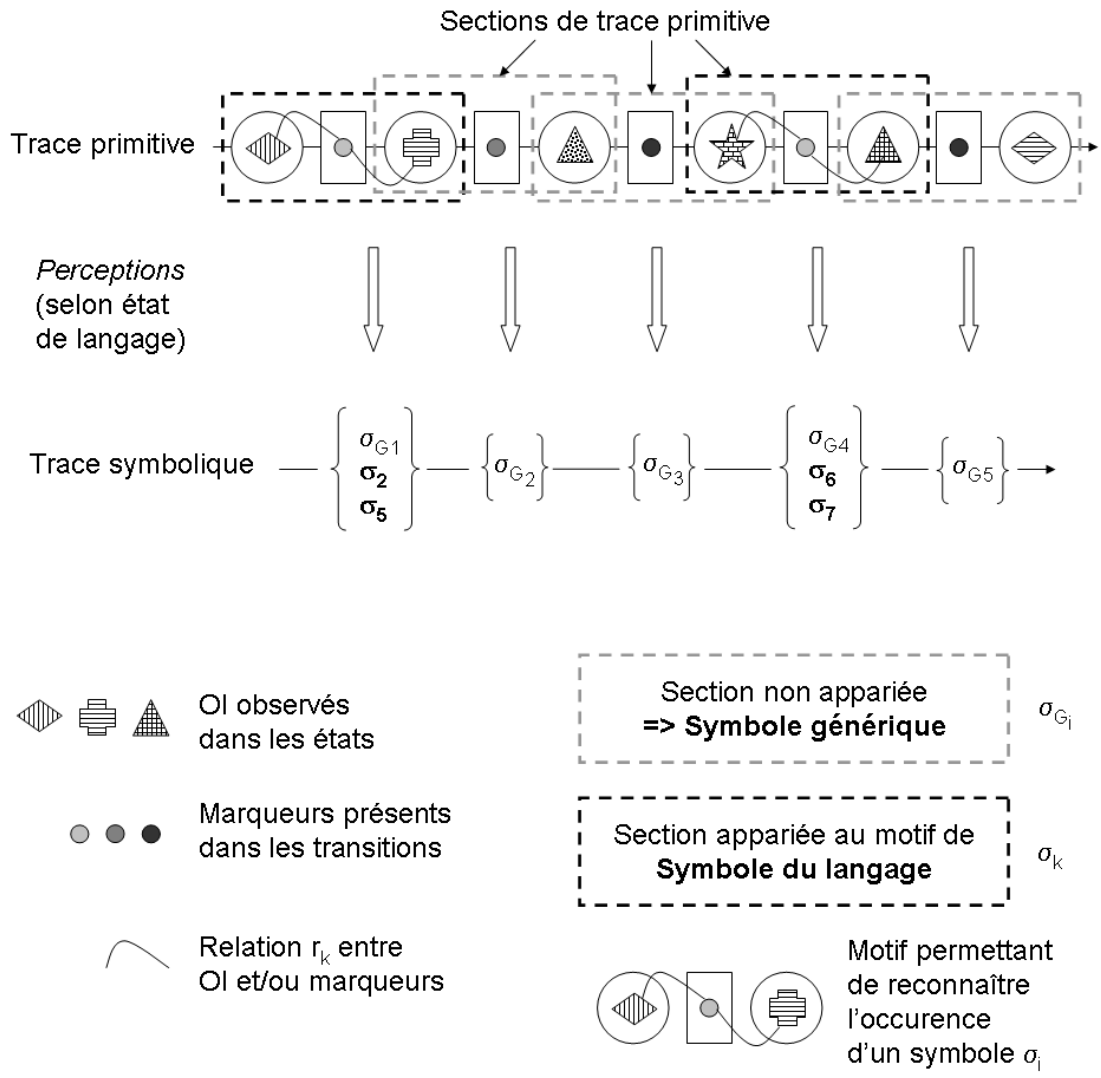


FIG. 6.10 – La trace primitive et sa représentation symbolique

définis dans l'état de langage en cours.

Notons que l'expression de la signification d'un symbole de langage par l'utilisateur à l'aide d'une phrase en langue naturelle, est une approche simplifiée de la problématique d'expression de sens. En effet, il serait intéressant, et même légitime, de pouvoir inclure des références aux objets d'intérêt, qui sont représentées par des captures (constantes ou variables) contenues dans les patterns du symbole, lors de la composition de cette phrase explicative. Nous reviendrons sur cela dans les perspectives.

6.4 Représentation des signatures de tâches

Dans notre introduction des principes d'acquisition de sens d'assistance (i.e., des signatures de tâches) à partir des indications de l'acteur humain, effectuée en début de chapitre, nous avons

indiqué le besoin d'une expression symbolique pour ces signatures de tâche. Cette expression symbolique est nécessaire à la transposition des mécanismes d'émergence de langage introduite pour gérer ces indications.

Le rôle d'une signature de tâche est de faire le lien entre un ensemble de symboles perçus et un sens d'assistance, exprimé en langage naturel par l'utilisateur, lui faisant par la même sens.

Ainsi, nous définissons une *signature (symbolique) de tâche* (ou signature) comme la paire formé :

- d'un ensemble non ordonné de symboles de langage ;
- d'un label en langage naturel, fixé par l'utilisateur.

Dans cette définition, l'ordre et les relations qui peuvent exister entre les symboles ne sont pas prises en compte ; nous traiterons du besoin de tenir compte de cela pour augmenter l'expressivité d'une signature et des possibilités qu'offre notre approche en perspectives (c.f., 10.4).

Tout comme pour les symboles, l'expression du sens d'une signature à l'aide du "phrase" en langage naturel est une première approche simplifiée pour l'expression du sens par l'utilisateur.

La reconnaissance d'une partie d'une signature dans la trace en cours représente une élaboration pour cette signature. Elle est définie par un ensemble de symboles perçus, sous-ensemble de l'ensemble des symboles définissant une signature. Une élaboration permet de construire un *épisode* cible, et plus précisément ce qui correspond à la partie problème de cet épisode cible.

L'agent utilise le label de la signature pour exprimer le sens d'une élaboration à l'utilisateur. Il disposera d'une interface pour donner à ce dernier l'accès au détail d'une élaboration, i.e., les symboles perçus sur telles sections.

L'épisode cible élaboré peut être comparé à des épisodes sources élaborés dans d'autres traces, et ce au regard des symboles communs à leur élaboration respectives ; les parties non reconnues des épisodes sources sont candidates à être solution pour l'épisode cible ; c'est le principe d'assistance par suggestion d'épisodes antérieurs similaires que nous avons adoptée. Nous reviendrons sur ces mécanismes dans la section détaillant le module de RàPET de l'agent alter ego, c.f., 6.6.

Les symboles et les signatures avec leurs définitions symboliques sont décrits dans un état de langage ; nous définirons formellement cette notion au chapitre suivant, c.f., 7.2, lors de la présentation de notre transposition des mécanismes d'émergence de langage.

Similarité de signatures :

Dans le processus de négociation de signature de l'agent alter ego, introduit en 4.6, il sera nécessaire à ce dernier de pouvoir évaluer la similarité entre deux signatures, par une valeur comprise dans $[0; 1]$.

Ainsi, nous définissons la similarité entre deux signatures Z_i et Z_j, décrites respectivement par les ensembles de symboles $Symb_{Z_i}$ et $Symb_{Z_j}$, par :

$$sim(Z_i, Z_j) = \frac{\sum_{S \in Symb_{Z_i} \cap Symb_{Z_j}} dim(S)}{\sum_{S \in Symb_{Z_i} \cup Symb_{Z_j}} dim(S)} \quad (6.6)$$

où, pour un symbole S, de forme $E \cdot T \cdot E$, avec les trois patterns associés constituant le motif : $Obs_{pattern_{Avant}}$, $Obs_{pattern_{Opération}}$ et $Obs_{pattern_{Après}}$, la dimension du symbole S, notée $dim(S)$, est défini par :

$$dim(S) = dim(Obs_{pattern_{Avant}}) + dim(Obs_{pattern_{Opération}}) + dim(Obs_{pattern_{Après}}) \quad (6.7)$$

La pondération de la mesure de similarité entre signature, c.f., équation 6.6, par les dimensions de leurs symboles est faite pour augmenter l'influence des symboles de grandes dimensions. Cela part du fait que nous considérons qu'un symbole de grande dimension est apparemment plus discriminant qu'un symbole de faible dimension, car il contient plus d'OI dans son motif, et nous pensons qu'il doit donc jouer un rôle plus significatif sur cette mesure de similarité en fonction de son caractère commun ou non.

Cependant, nous insistons sur le fait que ce principe de pondération, et plus précisément son expression, ne joue ici que le rôle d'**heuristique**, et dont il conviendrait d'en faire l'évaluation par l'expérimentation et l'ajustement dans une étude spécifique.

Exemple :

Considérons les symboles de langage, avec leur dimension (nombre d'OI capturés dans leur motif), suivants : $(\sigma_1, 3)$, $(\sigma_2, 5)$, $(\sigma_3, 2)$ et $(\sigma_4, 1)$. Ils permettent de définir les signatures Z_1 et Z_2 par : $Symb_{Z_1} = \{\sigma_1, \sigma_2\}$ et $Symb_{Z_2} = \{\sigma_2, \sigma_3, \sigma_4\}$. La similarité entre Z_1 et Z_2 est :

$$sim(Z_1, Z_2) = \frac{5}{3+5+2+1} \approx 0,45$$

6.5 Le choix d'une représentation avec RDF

Notre choix de RDF pour langage de description s'appuie sur l'analyse détaillée faite par [Champin 02].

Une évolution envisagée du World Wide Web est désignée par le terme Web Sémantique. Cette vision part du constat qu'une grande majorité des informations disponibles sur le Web sont destinées à être manipulées par des acteurs humains. Néanmoins, face à la quantité colossale d'informations disponibles, l'acteur humain peut tirer un grand avantage à s'appuyer sur un traitement automatique ou semi-automatique de l'information. Pour assurer cette tâche, les systèmes informatiques utilisés doivent disposer d'une meilleure représentation du contenu des

pages. Cette évolution a déjà commencée en remplaçant les descriptions de documents en HTML[2] par des descriptions basées sur les technologies XML[3], incluant entre autres des feuilles de style décrites en XSL[4]. Ce changement traduit la volonté de séparer la description des données (en XML) de leur représentation (décrite en XSL), ce qui n'était pas le cas en HTML.

Cette ambition est aussi applicable à la gestion des connaissances dans l'entreprise, pour faciliter l'exploitation par ses acteurs des informations stockées. Pour être efficace, une instrumentation doit être personnalisée (dépendante de l'acteur) et contextualisée (dépendante de la situation en cours), sans quoi la surcharge d'information risque de devenir plus néfaste que bénéfique. Nous retrouvons ici la vision contextuelle de la connaissance abordée en 2.1.

De manière générale, la réalisation du Web Sémantique suppose que les ressources soient appropriables par des agents logiciels hétérogènes. Il en résulte une problématique d'interopérabilité entre ces agents. Les techniques de représentation de connaissances et de raisonnement, développées en IA, semblent avoir leur place. Nous n'allons pas faire d'analyse détaillée et exhaustive des différentes approches existantes, mais présenter les principes de RDF[5] qui a l'avantage de nous permettre de décrire de façon assez directe et libre les éléments que nous manipulons.

RDF : RDF est souvent présenté comme un langage de description des *méta-données*, permettant d'enrichir un langage de description déjà utilisé, ce qui est convenable lorsqu'une ressource ne peut être modifiée.

Le principe de RDF est basé sur la notion de *propriété* : chaque ressource possède un certain nombre de propriétés ayant une certaine valeur. Chaque *triplet* (ressource, propriété, valeur) est représenté en RDF par une *déclaration*, et un ensemble de déclarations RDF peut être vu comme un graphe orienté étiqueté : chaque sommet représente une ressource (identifiée par une URI) ou un littéral (en l'absence d'URI, le sommet représente une chaîne de caractères) et chaque arc représente une déclaration. Pour une présentation plus détaillée, un tutoriel RDF est proposé par [Champin 01].
Avec RDF tout objet est identifiable par une ressource. Cela lui donne une réflexivité, car les propriétés sont des ressources. Un procédé de *réification* permet aussi de représenter un déclaration par une ressource, et donc de la manipuler comme telle. Notons également que XML et RDF doivent être vus comme complémentaires, le premier permet la définition de *syntaxes* des documents manipulés, quand au deuxième il propose d'exprimer un cadre *sémantique* minimum sans toutefois proposer une sémantique propre.

Une modélisation en RDF est une description "ontologiquement neutre", sur laquelle peu d'inférences peuvent être effectuées. Pour les permettre, il faut s'intéresser au domaine des *ontologies*. Les schémas RDF offrent quelques primitives pour décrire les schémas, basés sur des *classes* et des *propriétés* qui peuvent toutes deux être spécialisées ou généralisées. Les principes de réflexivité de RDF sont appliqués et un RDF-Schéma peut être représenté en RDF. Les RDF-Schema montrent cependant des limites dans l'expression de certaines conditions, tandis que les langages DAML+OIL et OWL permettent d'y remédier. N'exploitant pas les capacités d'inférence, nous nous baserons sur un RDF-Schema représentant l'ontologie MUSETTE et sa spécialisation à un

[2]HyperText Markup Language
[3]eXtensible Markup Language
[4]eXtensible Stylesheet Language
[5]Resource Description Framework

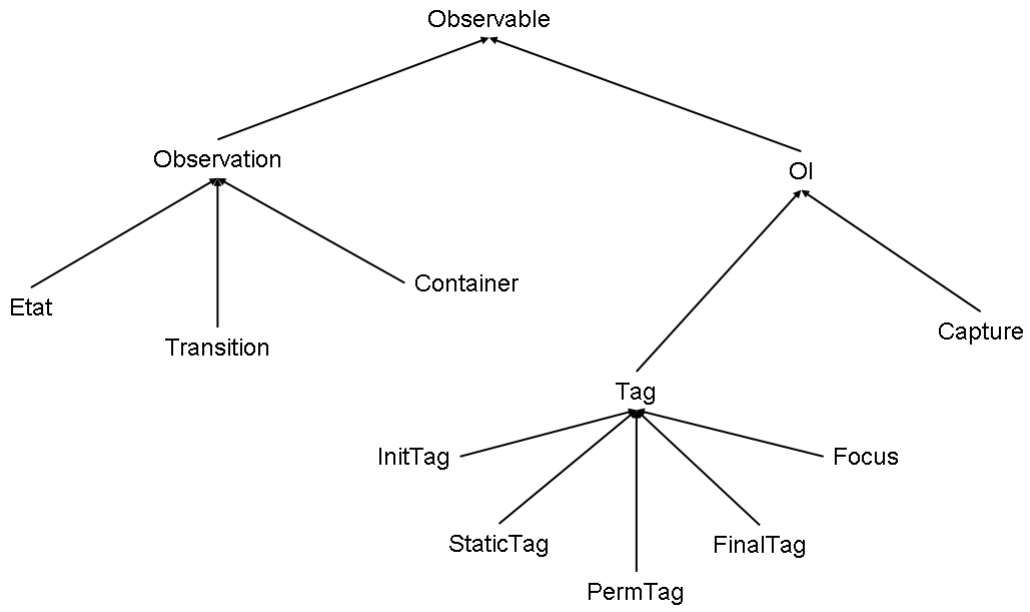


FIG. 6.11 – Ontologie de la *trace primitive* selon les principes de l'ontologie MUSETTE

environnement documentaire, afin de définir clairement les objets que nous manipulons. Toutefois, une grosse partie de la sémantique de nos descriptions de traces et de langage est portée par les procédures que nous introduisons.

Exemples : l'annexe A donne une série d'exemples de descriptions RDF pour :

- les fonctionnalités de l'application cible que nous considérons lors de la capture de trace, en A.1 ;
- une observation de trace, en A.2
- une observation de symbole, en A.3 ;
- la structure de trace, correspondant à deux actions menées conjointement, en A.4. Les observations n'y figurent pas ;
- la structure d'un symbole, en A.5. La propriété *topOnto* : $MATCHING\_ID$ associée à une capture, représente une variable ;
- les principes de description d'un état du language, en A.6.

Ces descriptions sont basées sur un RDF-Schema, regroupant l'ontologie de construction de trace primitive et l'ontologie de construction de trace symbolique.

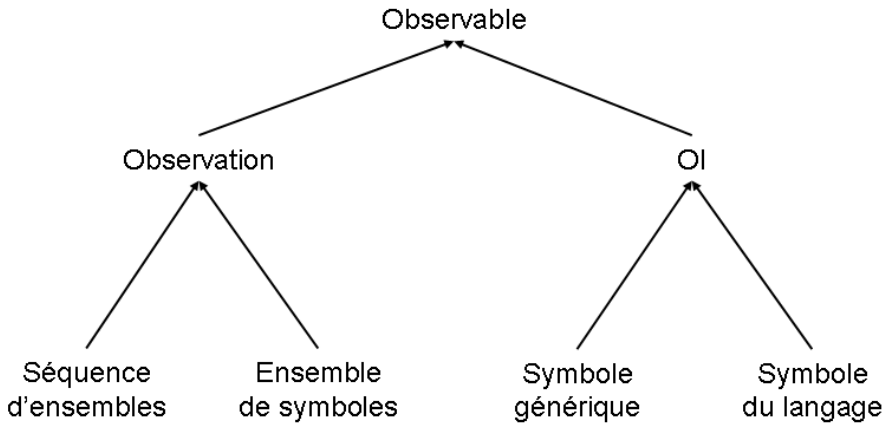


FIG. 6.12 – Ontologie de la *trace symbolique* selon les principes de l'ontologie MUSETTE

L'ontologie de trace primitive est représentée par la figure 6.11, dans laquelle les classes de l'ontologie de haut niveau MUSETTE (i.e., *OI*, et *Observation*, sous-classes de *Observable*) sont spécialisées de la façon suivante :

- pour la classe *OI*, nous définissons :
  - la classe *Capture*, dont les instances vont représenter la présence des mots-clés, c.f., modèle de domaine en 6.1, dans les champs des documents ;
  - la classe *Tag*, introduite pour unifier les différents types de marqueurs représentés par les sous-classes *InitTag*, *StaticTag*, *PermTag*, *FinalTag* et *Focus*. Les instances de ces sous-classes vont être utilisées pour marquer l'appel aux fonctionnalités de l'application cible, selon les principes définis dans notre modèle de description de la trace primitive à base de grammaire formelle, c.f., 6.3.1 ;
- pour la classe *Observation*, nous définissons :
  - la classe *Container*, dont les instances sont utilisées dans la construction de structures d'arbres d'observation ;
  - et nous rappelons les classes *Etat* et *Transition*, nécessaires pour la structure MUSETTE d'une trace, i.e., alternance d'instances d'Etat et de Transition.

L'ontologie de trace symbolique est représentée par la figure 6.12, dans laquelle les classes de l'ontologie de haut niveau MUSETTE sont spécialisées de la façon suivante :

- pour la classe *OI*, nous définissons :
  - la classe *Symbole générique*, dont une instance existe pour chaque instance des sous-

classes de *Tag*. Les symboles génériques permettent de désigner l'appel aux fonctionnalités de l'application cible ;

- la classe *Symbole du langage*, dont une instance représente la "perception" d'un symbole du langage sur une section de trace. Nous définirons précisément la notion de perception de symbole en 6.6.1 ;

- pour la classe *Observation*, nous définissons :

  - la classe *Ensemble de symboles*, dont une instance correspond à l'ensemble des symboles de langage perçus sur une section de trace auquel s'ajoute le symbole générique correspondant au marqueur présent dans la section. En cas d'absence de perception de symbole de langage, cet ensemble est composé uniquement du symbole générique ;

  - la classe *Séquence d'ensembles* de symboles, dont une instance représente la trace symbolique pour une trace primitive. Elle est construite à partir des instances d'*Ensemble de symboles* pour chaque section de la trace primitive.

6.6 Le module de RàPET

Cette section présente la mise en oeuvre des deux premières étapes du cycle de RàPET : l'*élaboration*, en 6.6.1, et la *remémoration* ou *recherche d'épisode*, en 6.6.2. Nous traiterons de la réalisation complète du cycle de RàPET en perspective (c.f., 10.5).

6.6.1 Le processus d'élaboration

Dans le RàPET, l'*élaboration* désigne le processus de construction d'un épisode cible dans la trace en cours par l'identification d'une signature. C'est le même mécanisme qui sera utilisé pour construire un épisode source dans une trace antérieure, selon les principes détaillés en 6.6.2 ; nous faisons sa présentation dans le cas de l'épisode cible.

Dans notre approche symbolique des traces et des signatures, une élaboration s'applique à la trace sous sa forme symbolique, et elle se décompose en :

1. une *perception* des symboles, définis dans l'état de langage en cours. Cette opération est sous-jacente à la reformulation symbolique de la trace.
2. une *interprétation* d'une signature de ce même état de langage, à partir des symboles perçus, i.e., ceux présents dans la trace symbolique.

La construction d'un épisode cible pour une signature est ascendante et elle est composée de deux mécanismes asynchrones :

1. une perception de symboles de langage sur la trace primitive, afin de construire la trace symbolique. Pour la trace en cours cette perception se fait au fil des acquisition de trace ;
2. une interprétation pour cette signature, revenant à reconnaître une situation d'assistance au delà d'un certain seuil d'occurrence dans la trace symbolique de symboles, qui la définissent.

La figure 6.13 illustre intuitivement les opérations effectuées par ces deux mécanismes, lors de l'élaboration d'un épisode cible pour la signature Z définie par l'ensemble de symboles $\{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6, \sigma_7\}$. Les symboles σ_6 et σ_7 ne sont pas reconnus, ils seront à la base des

suggestions de l'agent alter ego.

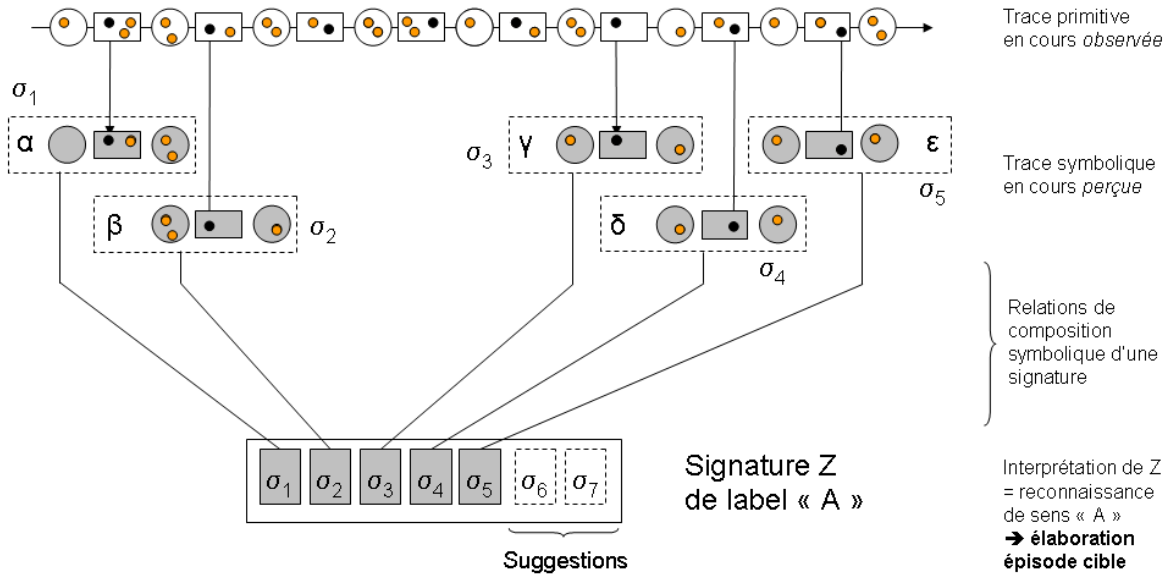


FIG. 6.13 – Mécanisme d'élaboration d'un épisode cible dans la trace en cours pour une signature

Plus précisément, les principes des ces mécanismes sont :

1. la perception est une opération de recherche itérative des symboles, dont les motifs peuvent être appariés à la section. Cette recherche est basée sur le marqueur présent dans la section : les symboles possédant le même marqueur sont candidats au test d'appariement de leur motif à la section. Une solution d'un tel appariement constitue une perception du symbole sur la section.
   L'appariement d'un motif de symbole à une section de trace n'est possible que si une solution d'appariement existe entre le pattern du motif $Obs_{pattern_{pos}}$ et l'arbre d'observation de la section $Obs_{section_{pos}}$ pour chaque position *pos* (Av, Op et Ap) dans la structure $E{\cdot}T{\cdot}E$ commune au motif de symbole et à la section. Une solution d'appariement du motif de symbole à la section est représentée par un triplet formé d'une solution d'appariement du pattern sur l'arbre pour chaque position. .
   L'algorithme d'appariement d'un pattern sur un arbre d'observation de trace est détaillé en 6.6.1.1.

2. l'interprétation d'une signature sur une trace symbolique est faite dans le but de sélectionner, parmi les perceptions de symboles, un sous-ensemble de perceptions de symboles qui la définisse. Cette sélection définit un épisode pour la signature considérée. Nous présenterons et discuterons l'algorithme utilisé dans ce travail en 6.6.1.2.

Lorsque ce mécanisme est appliqué à la construction de l'épisode cible dans la trace en cours, il doit également guider le déclenchement de l'étape de RàPET suivante, qu'est la remémoration.

En effet, étant donné qu'une signature de tâche regroupe l'expression de la partie problème et celle de la partie solution d'une situation d'assistance, ce déclenchement doit être fait avant que

tous les symboles de cette signature n'aient été perçus, afin de garder une dimension suggestive dans l'assistance. D'un autre côté, le déclenchement ne doit pas être précoce, car les suggestions de l'agent alter ego auraient peu de chance de faire sens à l'utilisateur.

Pour évaluer la pertinence de l'élaboration d'un épisode dans une trace trc pour une signature Z, nous introduisons une mesure appelée *taux d'identification*. Soient S_Z l'ensemble des symboles de Z, et $R_Z(trc)$ l'ensemble des symboles de Z reconnus à un certain instant dans la trace trc, on définit le *taux d'identification* par :

$$\tau_Z(trc) = \frac{\sum_{\sigma_i \in R_Z(trc)} \mathcal{P}(\sigma_i, Z) \cdot dim(\sigma_i)}{\sum_{\sigma_i \in S_Z} \mathcal{P}(\sigma_i, Z) \cdot dim(\sigma_i)} \tag{6.8}$$

où la fonction $\mathcal{P}(\sigma, Z)$ représente une pondération interne à l'agent alter ego. Cette fonction est basée sur des données propres à un état de langage (sur des pondérations des éléments de langage). Elle exprime la "confiance" ou l'"habitude d'utilisation avec succès" qu'a acquis l'agent alter ego à utiliser le symbole σ dans son ensemble, mais aussi plus spécifiquement dans l'interprétation de la signature Z. La fonction $\mathcal{P}(\sigma, Z)$ est croissante selon ces deux dimensions ; nous présenterons et discuterons cette fonction en 7.5.

Dans ce calcul du taux d'identification, la pondération du rapport entre le nombre de symboles reconnus à un instant et le nombre total de symbole de Z, par $\mathcal{P}(\sigma_i, Z) \cdot dim(\sigma_i)$, est faite pour tenir compte de :

- la "confiance" qu'a acquis l'agent alter ego à utiliser un symbole σ_i, selon l'explication que nous venons de donner de la fonction $\mathcal{P}(\sigma_i, Z)$. L'utilisation de cette fonction rend la mesure de taux d'identification fonction d'un état de langage ; cette mesure va évoluer selon les variations de "confiance" de l'agent alter ego inscrites dans les états de langage successivement construits ;
- l'importance d'un symbole, via $dim(\sigma_i)$, en considérant que plus sa dimension est grande, plus son rôle est important car porteur d'une capacité de discrimination importante (une dimension élevé traduit la présence de beaucoup de captures dans les patterns du motif de σ_i, c'est donc intuitivement un symbole "discriminant"). Nous reprenons ici le principe de pondération déjà discuté pour la mesure de similarité de signature, c.f., équation 6.6.

  De la même façon, nous insistons sur le fait que ce principe de pondération ne joue ici que le rôle d'**heuristique**. Il conviendrait d'en faire l'évaluation par l'expérimentation et l'ajustement de son expression, qui pourrait diverger de celui fait pour la similarité de signature.

L'étape suivante du RàPET, i.e., la recherche d'épisodes sources similaires, est déclenchée lorsque le taux d'identification d'un épisode cible dépasse un seuil, noté τ_{idSign}.

6.6.1.1 Algorithme de recherche des appariements d'un pattern à un arbre d'observation

Notre appariement d'un pattern (*source*) à un arbre d'observation (*cible*) est basé sur le respect d'une structure, via la mise en correspondance des containers, et sur le respect d'un

contenu, via l'appariement de toutes les constantes et de toutes les variables d'un container de pattern aux constantes du container d'arbre d'observation correspondant dans la structure. L'appariement n'est possible que si toutes ces conditions sont remplies.

D'autre part, nous choisissons de faire une recherche complète de tous les appariements du pattern à l'arbre d'observation. Cette approche est motivée par l'idée que l'agent alter ego ne doit pas privilégier une alternative à une autre, car cette opération d'appariement a un caractère uniquement perceptif ; la sélection d'une alternative doit plutôt s'effectuer au moment de l'interprétation, i.e., lors de l'assemblage des symboles pour identifier une signature. L'algorithme que nous présentons est une approche générale de la problématique d'appariement, pour notre modèle de pattern et d'arbre d'observation ; nous discuterons de son utilisation et de ses limites à la fin de cette section.

Pour effectuer cette opération d'appariement, nous utilisons un algorithme récursif. Pour les containers nous avons mis la condition de labels discriminants entre containers frères, cette condition est nécessaire pour construire les ensembles I et U lors du calcul de la similarité entre patterns, c.f., équation 6.5. Cela évite également des ambiguïtés qui conduisent à des solutions d'appariement multiples, et limite la complexité de notre algorithme.

Cependant, l'étiquetage des containers avec des labels distincts entraîne l'impossibilité d'exprimer que deux containers tiennent des rôles similaires (i.e., pouvant être permutés) dans l'expression d'une observation. Pour illustrer cela, considérons un container regroupant d'autres containers, que nous appellerons ses *sous-containers*, afin de représenter simplement un ensemble non ordonné. Comme les sous-containers doivent être étiquetés de façon distincte, on peut par exemple leur donner les labels *item-1*, *item-2*, ... *item-N*. L'application de notre algorithme d'appariement ne cherchera qu'à rapprocher les sous-containers de même étiquette (*item-i* entre eux) sans exploiter leur statut équivalent (*item-i* avec *item-j*). Pour notre travail, nous gardons cette hypothèse de labels distincts. Le relâchement de cette condition permettrait de traiter la limite évoquée ci-dessus. L'absence de label demande de revoir l'algorithme d'appariement décrit ci-dessous, à l'instar du traitement de l'appariement des variables que nous allons présenter, et entraînerait une combinatoire forte ; la similarité entre patterns devrait également être modifiée.

Pour l'appariement des variables, d'un container du pattern, aux constantes, d'un container d'arbre d'observation, il se peut que plusieurs alternatives d'appariement soient possibles. Sur la figure 6.14, nous donnons : 1) un exemple d'appariement pour lequel plusieurs alternatives se présentent : la variable L_B^* peut être appariée à B ou b, et la variable L_C^* peut être appariée à c ou c' ou c''. Cela implique de considérer les six alternatives d'appariement : $\{(L\_B^*, B), (L\_C^*, c)\}$, $\{(L\_B^*, B), (L\_C^*, c')\}$, $\{(L\_B^*, B), (L\_C^*, c'')\}$, $\{(L\_B^*, b)$ $, (L\_C^*, c)\}$, $\{(L\_B^*, b), (L\_C^*, c')\}$ et $\{(L\_B^*, b), (L\_C^*, c'')\}$. Dans cet exemple, les variables L_B^* et L_C^* sont indépendantes, mais il se peut qu'une variable représente une abstraction qui subsume celle d'une autre variable et leur appariement aux constantes devient alors concurrent avec des possibilités de blocage, un exemple de ce problème classique est donné en 2). Pour résoudre ce problème, qui se ramène à un problème de satisfaction de contraintes, nous utiliserons l'algorithme complet de *Simple Back Track*. Enfin, pour exemple en 3), un échec d'appariement dû à l'absence de candidat pour la variable L_C^*.

Soit un pattern source Obs_s et un arbre d'observation cible Obs_c, nous notons respectivement O_s et O_c l'ensemble des containers et des captures d'objets d'intérêt qui les composent. Les éléments de O_s et O_c sont considérés indistinctement comme des *objets* pour l'algorithme d'appariement, ce sont des ressources en RDF.

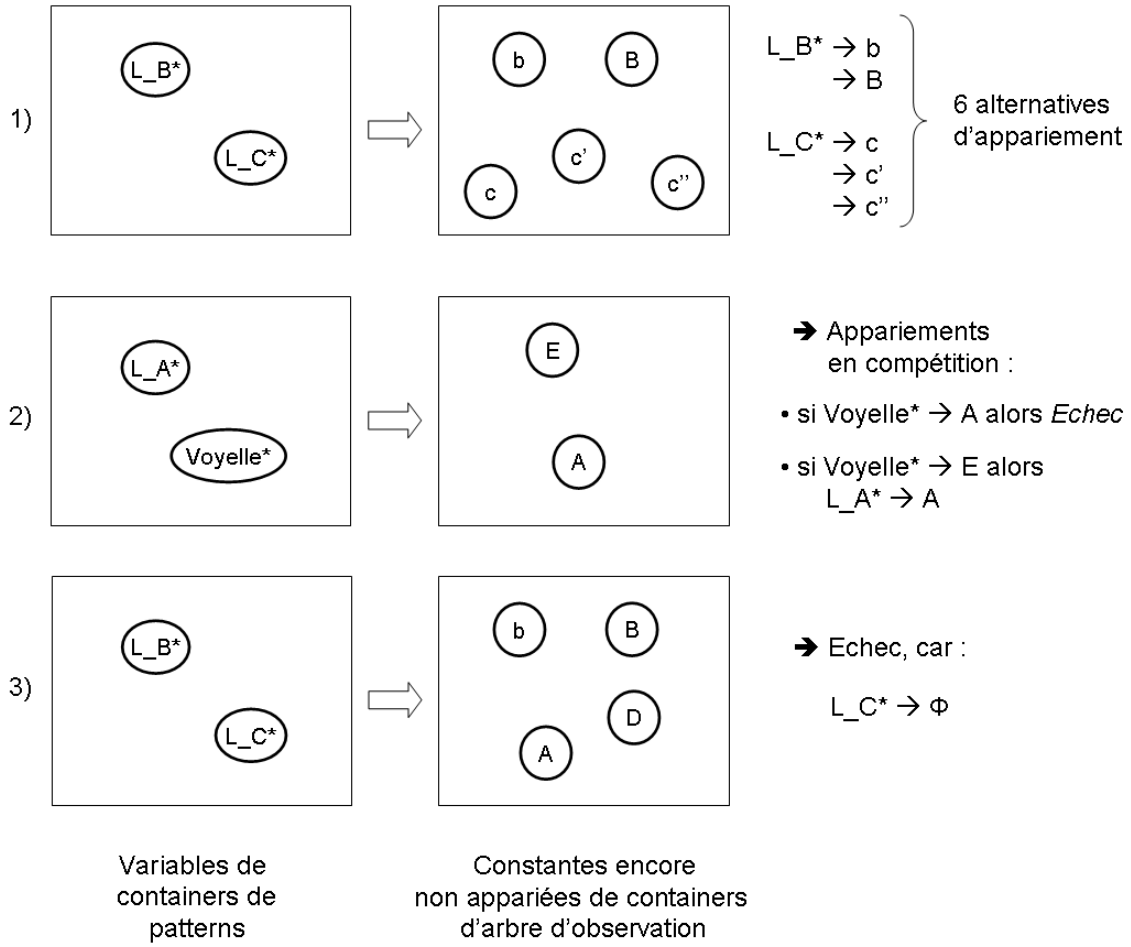


FIG. 6.14 – Deux situations (en haut et en bas) d'appariement de variables d'un container de pattern avec les variables encore non appariées d'un container d'arbre d'observation

Pour représenter, un appariement de deux objets, un appariement partiel et les alternatives d'appariement partiel qui sont générées par l'appariement des variables, nous introduisons les structures de données C, P et S suivantes :

- $C = (o_s, o_c) \subset O_s \times O_c$, un *couple* d'objets appariés ;
- $P = \{C_1, \ldots, C_n\}$, un *appariement partiel* : un ensemble de couples ;
- $S = \{P_1, \ldots, P_n\}$, une *solution partielle* : un ensemble d'appariement partiel, i.e., des alternatives d'appariement.

Nous introduisons également $S_\phi = \{\emptyset\}$, l'ensemble d'alternative vide, i.e., il représente aucune possibilité d'apparier le pattern à l'arbre d'observation, et il est considéré comme l'expression de l'échec de cet appariement.

Fonction Combine
  entrée : $A_1 = \{P_i\}$ et $A_2 = \{P_j\}$
  sortie : un ensemble A_S d'alternatives d'appariement
début
  si $A_1 = A_\phi$ OU $A_2 = A_\phi$ **alors retourne** A_ϕ
  sinon
    $A_S \leftarrow \{\emptyset\}$
    pour tout $P_i \in A_1$ **faire**
      pour tout $P_j \in A_2$ **faire**
        $A_S \leftarrow A_S \cup \{P_i \cup P_j\}$
      fin-pour
    fin-pour
  fin-si
  retourne A_S
fin

Fig. 6.15 – Algorithme de la fonction de combinaison de deux alternatives d'appariement partiel

Deux solutions partielles doivent être combinées au cours des appels récursifs à notre algorithme d'appariement, afin de former une nouvelle solution partielle correspondant à toutes les possibilités d'assemblage de leurs appariements partiels. La fonction Combine, c.f., figure 6.15, effectue cette opération de combinaison de deux solutions partielles $S_1 = \{P_i\}_{i\in[1,N_1]}$, avec des appariements partiels de taille M_1, et $S_2 = \{P_j\}_{j\in[1,N_2]}$, avec des appariements partiels de taille M_2. Elle renvoie une solution partielle composée de $N_1 \times N_2$ alternatives d'appariement partiel, de taille $M_1 + M_2$, qui est définie formellement par :

$$\textsc{Combine}(S_1, S_2) = \left\{ \{P_i \cup P_j\}_{\forall(i,j)\in[1,N_1]\times[1,N_2]} \right\}$$

La fonction Combine est telle que :

$$\textit{pour toute solution partielle } S,\ \textsc{Combine}(S, S_\phi) = \textsc{Combine}(S_\phi, S) = S_\phi$$

L'algorithme d'appariement de Obs_s avec Obs_c est donné par la fonction indirectement récursive Match détaillée sur la figure 6.16. L'appariement est lancé entre appelant Match entre les deux containers racines du pattern et de l'arbre d'observation.

La première étape de la fonction Match est l'appel de la fonction booléenne VérifierSémantique (c.f., figure 6.17) qui teste si les objets candidats peuvent s'apparier au regard de la sémantique correspondant à la vérification de l'une des conditions suivantes :

- si les objets source et cible sont des containers, ils doivent avoir le même label ;
- si les objets source et cible sont des captures (fonction booléenne EstCapture) constantes (fonction booléenne EstConstante, ils doivent représenter le même OI (schématiquement $o_s = o_c$) ;
- si l'objet source o_s est une capture (fonction booléenne EstCapture) variable (fonction booléenne EstVariable), il doit représenter un concept dont l'objet cible o_c est une instanciation (testé avec la fonction booléenne EstInstanceDuConcept)(o_c, o_s).

Fonction Match
  entrée : $o_s \in O_s$ et $o_c \in O_c$
  sortie : une solution partielle S, S_ϕ si échec
début
  si VérifierSémantique(o_s, o_c) = faux **alors retourne** S_ϕ
  sinon
    $S \leftarrow \{\{(o_s, o_c)\}\}$
    si EstCapture(o_s) **alors retourne** S
    sinon
      $S \leftarrow$ Combine$(S,$ MatchCapturesConstantes$(o_s, o_c))$
      si $S = S_\phi$ **alors retourne** S_ϕ
      $S \leftarrow$ Combine$(S,$ MatchCapturesVariables$(o_s, o_c))$
      si $S = S_\phi$ **alors retourne** S_ϕ
      $S \leftarrow$ Combine$(S,$ MatchContainers$(o_s, o_c))$
    fin-si
  fin-si
  retourne S
fin

Fig. 6.16 – Algorithme d'appariement d'un arbre d'observation de pattern sur un arbre d'observation de trace

Fonction VérifierSémantique
  entrée : deux objets $o_s \in O_s$ et $o_c \in O_c$
  sortie : vrai si la sémantique de l'appariement est vérifiée, faux sinon
début
  si EstConstante$(o_s) \wedge$ ($o_s = o_c$) **retourne** vrai
  si EstVariable$(o_s) \wedge$ (EstInstanceDuConcept(o_c, o_s) = vrai) **retourne** vrai
  si EstContainer$(o_s) \wedge$ (Label(o_s) = Label(o_c)) **retourne** vrai
  retourne faux
fin

Fig. 6.17 – Algorithme de vérification de la sémantique pour l'appariement de l'objet o_s avec l'objet o_c

```
Fonction MatchCapturesConstantes
   entrée  : deux objets o_s ∈ O_s et o_c ∈ O_c
   sortie  : une solution partielle S, S_φ si échec
début
   S ← {∅}
   C_s ← CapturesConstantes(o_s)
   C_c ← CapturesConstantes(o_c)
   pour tout c ∈ C_s
      c' ← RechercheConstanteCandidate(c, C_c)
      si c' inexistant retourne S_φ
      sinon
         S_M ← Match(c, c')
         si S_M = S_φ retourne S_φ
         sinon
            si S = {∅} alors S ← S_M
            sinon
               S ← Combine(S, S_M)
            fin-si
            C_c ← C_c − {c'}
         fin-si
      fin-si
   fin-pour
   retourne S
fin
```

Fig. 6.18 – Algorithme d'appariement des constantes

```
FONCTION MATCHCONTAINERS
   entrée  : deux objets o_s ∈ O_s et o_c ∈ O_c
   sortie  : une solution partielle S, S_φ si échec
début
   S ← {∅}
   C_s ← SOUSCONTAINERS(o_s)
   C_c ← SOUSCONTAINERS(o_c)
   pour tout c ∈ C_s
      c' ← RECHERCHESOUSCONTAINERCANDIDAT(c, C_c)
      si c' inexistant retourne S_φ
      sinon
         S_M ← MATCH(c, c')
         si S_M = S_φ retourne S_φ
         sinon
            si S = {∅} alors S ← S_M
            sinon
               S ← COMBINE(S, S_M)
            fin-si
            C_c ← C_c − {c'}
         fin-si
      fin-si
   fin-pour
   retourne S
fin
```

FIG. 6.19 – Algorithme d'appariement des sous containers de l'objet o_s avec ceux de l'objet o_c

```
FONCTION MatchCapturesVariables
  entrée : deux objets o_s ∈ O_s et o_c ∈ O_c
  sortie : une solution partielle S, S_φ si échec
début
  S ← {∅}
  C_s[ ] ← CapturesVariables(o_s)
  C_c[ ] ← CapturesConstantesRestantes(o_c)
  si taille(C_s) > taille(C_c) alors retourne S_φ fin-si

  tableau[taille(C_s)][taille(C_c)] tab
  pour i de 1 à taille(C_s) − 1 faire
    pour j de 1 à taille(C_c) − 1 faire
      tab[i][j] ← Match(C_s[i], C_c[j])
    fin-pour
  fin-pour

  Alt ← SimpleBackTrack(tab, taille(C_s), taille(C_c), {∅}, {∅})

  si Alt = {∅} alors retourne S_φ
  sinon
    Alt ← EnleverDuplicata(Alt)
    pour tout a ∈ Alt faire
      S_a ← {∅}
      pour tout p ∈ a faire
        si S_a = {∅} alors S_a ← tab[p.x][p.y]
        sinon
          S_a ← Combine(S_a, tab[p.x][p.y])
        fin-si
      fin-pour
      S ← S ∪ S_a
    fin-pour
  fin-si
  retourne S
fin
```

FIG. 6.20 – Algorithme d'appariement des variables

Fonction SimpleBackTrack
  entrée : tab, le tableau à deux dimensions construit dans MatchCapturesVariables
    N_s, nombre de sources
    N_c, nombre de cibles
    A_p, l'alternative en cours de construction, i.e, un ensemble de *Positions*
    Sol, les alternatives déjà trouvées, i.e., un ensemble d'alternatives de taille N_s
  sortie : un ensemble d'alternatives d'appariement des variables
    ensemble vide si aucune possibilité
début
  si $taille(A_p) = N_s$ **alors**
    $Sol \leftarrow Sol \cup A_p$
    retourne Sol
  sinon
    pour i **de** 1 **à** $N_s - 1$ **faire**
      pour j **de** 1 **à** $N_c - 1$ **faire**
        si $tab[i][j] \neq S_\phi$ ET EstDéjàAffecté$(i, j, A_p) = faux$ **alors**
          $A_r \leftarrow A_p \cup$ **new** Position(i, j)
          $Sol \leftarrow$ SimpleBackTrack$(tab, N_s, N_c, A_r, Sol)$
        fin-si
      fin-pour
    fin-pour
    retourne Sol
  fin-si
fin

Fig. 6.21 – Algorithme de *Simple Back Track* utilisé pour trouver les alternatives d'appariement des variables

Dans la fonction MATCH, après le test de la sémantique pour o_s et o_c, si o_s est une capture, la solution partielle $\{\{(o_s, o_c)\}\}$ est renvoyé ; cela constitue à une des deux conditions d'arrêt de la récursivité de la fonction MATCH. Sinon, o_s et o_c sont des containers de même label, qui sont appariés.

La suite de la fonction MATCH, qui se déroule uniquement lorsque o_s et o_c sont des containers, est consacrée à l'appariement successif :

1. des captures constantes du pattern aux captures constante de l'arbre d'observation, via la fonction MATCHCAPTURESCONSTANTES, c.f., figure 6.18 ;
2. des captures variables du pattern aux captures constantes non appariées de l'arbre d'observation, via la fonction MATCHCAPTURESVARIABLES, c.f., figure 6.20 ;
3. des sous-containers (i.e., les containers directement imbriqués) de o_s avec ceux de o_c, via la fonction MATCHCONTAINERS, c.f., figure 6.19.

Ces fonctions contiennent chacune un appel récursif à la fonction MATCH. Les solutions partielles intermédiaires sont combinées (appel à COMBINE), et en en cas d'impossibilité d'appariement (retour de S_ϕ par l'une d'entre elles), la fonction MATCH est interrompue en renvoyant S_ϕ.

Les fonctions MATCHCAPTURESCONSTANTES et MATCHCONTAINERS se déroulent suivant le même principe, en raison de l'absence d'alternatives. Les ensembles C_s et C_c respectivement d'objets sources candidats et d'objets cibles candidats sont construits, avec des fonctions spécifiques aux types d'objets manipulés (fonctions CAPTURESCONSTANTES et SOUSCONTAINERS). Ensuite, pour chaque objet source candidat et il recherché l'objet cible, respectivement via les fonctions RECHERCHECONSTANTECANDIDATE et RECHERCHESOUSCONTAINERCANDIDAT, qui s'appuient toutes deux sur les contraintes sémantiques (fonction VÉRIFIERSÉMANTIQUE) ; si aucun candidat n'est trouvé, l'appariement échoue dans son ensemble, i.e., renvoi de S_ϕ. Si un candidat existe, l'appel récursif à la fonction MATCH est effectué ; pour les constantes, cet appel revient à renvoyer la solution partielle $\{\{(c, c')\}\}$, pour les containers c'est une boucle de récursivité. Si l'appariement est impossible, S_ϕ est renvoyé, sinon toutes les solutions partielles pour chaque couple de candidats sont combinées (fonction COMBINE). Un candidat cible apparié est retiré de C_c via l'opération $C_c \leftarrow C_c - \{c'\}$, cela l'exclu de la recherche de candidat aux itérations suivantes, et permet de déterminer les constantes non appariées utiles à la fonction MATCHCAPTURESVARIABLES.

La fonction MATCHCAPTURESVARIABLES doit traiter les alternatives d'appariement des différentes variables de la source avec les constantes non appariées de la cible. Elle commence par rassembler, sous la forme de tableaux ($C_s[\,]$ et $C_c[\,]$), les objets sources candidats (les variables), via la fonction CAPTURESVARIABLES, et les objets cibles candidats (les constantes non appariées), via la fonction CAPTURESCONSTANTESRESTANTES. A ce niveau, un test de faisabilité de l'appariement est fait, via le test de $taille(C_s) > taille(C_c)$ qui, s'il est vérifié, conduit au renvoi de S_ϕ, car toutes les variables doivent au moins pouvoir être appariées à un candidat cible. Ensuite, une tentative d'appariement est effectuée (appel à la fonction MATCH) pour toutes les paires de candidats, et les solutions partielles obtenues sont rassemblées dans le tableau $tab[\,][\,]$.

L'ensemble des appariements possibles de toutes les variables aux constantes doit maintenant être déterminé. Cette opération est effectuée à l'aide d'un algorithme récursif de *Simple Back Track* (fonction SIMPLEBACKTRACK, c.f., figure 6.21). Pour représenter l'appariement du $i^{ième}$ candidat source au $j^{ième}$ candidat cible, nous utilisons une $Position(i, j)$. Une combinaison complète ou non est un ensemble de positions appelé *Alternative*. La solution est un ensemble

d'alternatives complètes. La contrainte à respecter pour ajouter une $Position(i, j)$ à une alternative partielle (A_p) est que l'appariement soit possible ($tab[i][j] \neq S_\phi$) et que ni la source et ni cible soient déjà dans l'alternative partielle, ceci est testé par la fonction booléenne EstDéjàAffecté. Quand une alternative contient des positions pour tous les candidats source, elle est ajoutée à la solution.

La fonction MatchCapturesVariables stocke le résultat de l'appel à SimpleBackTrack dans *Alt*, i.e., ensemble d'alternatives complètes. Si *Alt* est vide, aucune combinaison est possible et cela conduit à un échec de l'appariement. Sinon, les duplicata d'alternatives, issus des symétries du problème résolu par *Simple Back Track*, sont supprimées (fonction EnleverDuplicata). Les alternatives de *Alt* sont ensuite parcourues ; pour chacune, il est construit une solution partielle qui est la combinaison (fonction Combine) des solutions partielles contenues dans $tab[\,][\,]$, en y faisant référence à l'aide des positions contenues dans l'alternative. Les appariements partiels de la solution partielle de chaque alternative sont ajoutés à la solution partielle qui sera retournée.

Nous avons traité le problème théorique de la perception de l'agent alter ego, en considérant qu'elle pouvait être multiple, ce qui conduit de fait à des perceptions candidates en compétition dès lors que l'on doit en choisir une. Nous avons argumenté que ce choix relève de l'interprétation, ce qui nous conduit à rechercher toutes les possibilités d'appariement d'un pattern à un arbre d'observation. Nous sommes conscients que l'approche que nous proposons peut entraîner une explosion combinatoire ; cela est conditionné par le nombre de variables présentes dans les containers ; et, a fortiori, si on envisage de relâcher l'hypothèse de labels discriminants pour les containers frères, et qu'on adopte une démarche analogue aux variables pour traiter les alternatives d'appariement entre containers frères.

Dans le cas de notre démonstrateur, c.f., chapitre 9, les arbres d'observation sont de très petite dimension et leurs containers contiennent une ou deux captures ; les patterns qui vont pouvoir être créés sont tout au plus de la dimension des arbres d'observation. Cette situation est très favorable en terme de complexité, et il y a bien souvent qu'un seul appariement possible. Pour ces raisons, dans la suite, nous allons faire implicitement l'hypothèse qu'il n'existe qu'un appariement possible d'un pattern à un arbre d'observation.

Néanmoins, toutes les mesures de similarité que nous avons introduites, et celles que nous introduirons, dépendent des appariements considérés ; en cas de multiplicité des appariements, la selection des appariements utilisés pour les calculs de similarité peut se faire, dans un premier temps, dans une logique de maximisation de ces mesures, sans toutefois écarter la possibilité de réviser cette sélection. En perspective, c.f., 10.4, nous traiterons du besoin d'ajouter des contraintes inter et intra-pattern ; de telles contraintes pourraient servir à réduire la complexité de l'algorithme d'appariement et permettre la sélection des alternatives en fournissant des "graines", i.e., en imposant des couples (objet source, objet cible). La problématique de la perception de l'agent alter ego, et nous verrons en 10.4 que c'est également le cas pour l'interprétation, est un lieu privilégié pour la programmation par contraintes.

6.6.1.2 Algorithme d'interprétation de signature

Le mécanisme d'appariement, que nous venons de présenter dans la section précédente, est l'opération élémentaire pour l'appariement d'un motif de symbole à une section de trace, via son application à tous les couples (pattern, arbre d'observation) donnés par leur structure commune. La trace symbolique est le résultat d'une recherche systématique, sur chaque section, de perceptions de symboles de langage ; nous rappelons qu'un symbole générique peut toujours être associé à une section, un tel symbole fait partie de la trace symbolique.

Parmi les symboles de langage perçus dans la trace symbolique, le mécanisme d'interprétation d'une signature revient à effectuer une sélection pertinente de perceptions pour les symboles qui définissent cette signature. Il s'agit bien d'une sélection, car un symbole de langage définissant une signature peut être perçu sur plusieurs sections de trace, et donc être présent plusieurs fois dans la trace symbolique. En parlant de sélection pertinente, nous soulignons le fait que ce mécanisme a pour but de construire un épisode (cible puis source) et que de cette construction résulte l'expression détaillée d'un sens d'élaboration à l'utilisateur. Cette pertinence d'interprétation est contrainte par l'expressivité du modèle symbolique d'une signature.

Compte tenu de notre modèle symbolique de signature de tâche, i.e., un ensemble non ordonné de symboles de langage, cette sélection se résume, pour les symboles définissant une signature, à trouver les symboles qui ont été perçus ; en cas de perceptions multiples pour un symbole, une unique perception doit être choisie.

La faible expressivité de notre modèle de signature, nous a conduit à écarter dans un premier temps la problématique de pertinence, abordée ci-dessus. En effet, il est vain d'étudier cette question sur la base d'un modèle si pauvre. Nous reviendrons sur cela en perspective, c.f., 10.4, lorsque nous traiterons de l'enrichissement de cette expressivité.

Par conséquent, nous adoptons dans un premier temps un mécanisme glouton d'interprétation : en parcourant chronologiquement les sections d'une trace, la première perception d'un symbole définissant une signature est sélectionnée et cette sélection ne sera pas remise en cause. Nous faisons l'hypothèse que cette interprétation est pertinente pour l'utilisateur.

Dans la suite de ce chapitre, nous discuterons des faiblesses et incorrections de ce premier mécanisme d'interprétation lorsque nous aborderons des aspects qui relèvent de cette problématique.

6.6.2 Recherche d'épisode dans les traces

La recherche d'épisodes similaires dans les traces antérieures est l'étape suivant celle d'élaboration dans le cycle de RàPET. Nous avons introduit dans la section précédente que la recherche d'épisodes sources s'effectue à partir d'un certain seuil (τ_{idSign}) d'identification d'une signature ; cela vient de contraintes techniques car la recherche d'épisodes sources est une opération nécessitant beaucoup de ressources de calcul, mais aussi du fait qu'une recherche d'épisode à partir d'une signature faiblement identifiée a peu de chance d'intéresser l'utilisateur. Pour une signature Z identifiée au delà de ce seuil, la recherche d'épisodes est faite sur la base des symboles reconnus à ce stade de l'élaboration, ce qui est par nature fonction du temps.

Considérons un symbole σ_i, constitué d'un label en langage naturel et d'un motif de structure $E \cdot T \cdot E$ (désignée par les positions $Av \cdot Op \cdot Ap$), à laquelle sont associés respectivement les patterns $Obs_{pattern_{Av}}$, $Obs_{pattern_{Op}}$ et $Obs_{pattern_{Ap}}$. Ce symbole est identifié sur une section Sec_{a_i} de la trace en cours et sur une section Sec_{b_i} d'une trace antérieure. Tout comme le symbole, ces sections ont des structures de la forme $E \cdot T \cdot E$, et auxquelles sont respectivement associées les ensembles d'arbres d'observation $\{Obs_{Av_{a_i}}, Obs_{Op_{a_i}}, Obs_{Ap_{a_i}}\}$ et $\{Obs_{Av_{b_i}}, Obs_{Op_{b_i}}, Obs_{Ap_{b_i}}\}$.

Nous définissons la similarité $sim_{\sigma_i}(Sec_{a_i}, Sec_{b_i})$ entre les sections Sec_{a_i} et Sec_{b_i} au regard du motif du symbole σ_i comme la moyenne pondérée des similarités $sim_{Obs_{pattern_{pos}}}(Obs_{pos_{a_i}}, Obs_{pos_{b_i}})$ (c.f., équation 6.2), où *pos* représente une position dans le parcours de la structure $E \cdot T \cdot E$ du symbole ou d'une section ($pos \in \{Av, Op, Ap\}$). Une pondération p_{pos} est associée à chaque

position, afin de pouvoir nuancer l'influence respective des patterns.

$$sim_{\sigma_i}(Sec_{a_i}, Sec_{b_i}) = \frac{\sum_{pos \in \{Av,Op,Ap\}} p_{pos} \cdot sim_{Obs_{pattern_{pos}}}(Obs_{pos_{a_i}}, Obs_{pos_{b_i}})}{\sum_{pos \in \{Av,Op,Ap\}} p_{pos}} \quad (6.9)$$

Nous plaçons les pondérations p_{pos} en paramètres général de l'agent alter ego, dans une première approche. Toutefois, ces valeurs devraient faire partie de la définition d'un symbole ; nous reviendrons en perspective sur l'intégration de ces pondérations dans la définition d'un symbole, et nous aborderons les problèmes que cela soulève pour la comparaison de symboles.

Soient la trace en cours trc_a et S_a l'ensemble des symboles de la signature Z qui y ont été perçus selon le mécanisme d'élaboration présenté en 6.6.1. Soit une trace antérieure trc_b candidate, le même mécanisme d'élaboration est utilisé, en limitant la recherche d'appariements de motifs de symboles sur les sections de trc_b aux motifs des symboles de Z. On note S_b l'ensemble des symboles de Z qui peuvent être identifiés dans trc_b en la parcourant jusqu'à son terme.

Pour évaluer la similarité entre l'épisode en cours de formation dans trc_a avec un épisode reconnu dans trc_b, nous ne pouvons le faire qu'en comparant les symboles qui sont reconnus dans chacune des traces. On note $S_C = S_a \cap S_b$ l'ensemble des symboles de Z identifiés à la fois dans trc_a et dans trc_b ; ceci représente la *similarité pour la partie commune* aux interprétations de Z dans les deux traces. Afin d'augmenter la pertinence de notre mesure de similarité, nous introduisons deux facteurs de correction dans le but de tenir compte de :

1. la *couverture* (notée $couv \in [0;1]$)) de l'épisode antérieur sur l'épisode en cours, élaborés tous deux avec Z. En considérant que seulement une partie des symboles de Z peut être reconnue dans un épisode antérieur, et a fortiori dans l'épisode en cours, la couverture est faite pour privilégier les épisodes antérieurs qui ont le plus de symboles communs (S_C) avec ceux de l'épisode en cours (S_a). Intuitivement, nous estimons qu'un épisode antérieur "englobant", en terme d'ensembles de symboles perçus, l'épisode en cours est plus pertinent qu'un épisode antérieur qui n'a pas tous les symboles de l'épisode en cours ;
2. la *représentativité* (notée $repr \in [0;1]$)) de l'épisode antérieur par rapport à la signature Z. Intuitivement, nous voulons mettre en avant combien un épisode antérieur est une expérience, qui illustre la plus complètement, la situation désignée par Z.

Nous illustrerons dans l'exemple ci-dessous les raisons qui motivent l'introduction de ces deux facteurs de correction.

Sur la base de ces principes, nous définissons la similarité sémantique entre les épisodes identifiés par la signature Z, à un instant, dans les traces trc_a (la trace en cours) et trc_b (une trace antérieure) par :

$$sim_Z(trc_a, trc_b) = sim_Z(S_C, trc_a, trc_b) \cdot couv_Z(S_a, S_C) \cdot repr_Z(trc_b) \quad (6.10)$$

où la similarité au regard de la partie d'interprétation commune S_C, en incluant la similarité entre sections, c.f., équation 6.9, est définie par :

$$sim_Z(S_C, trc_a, trc_b) = \frac{\sum_{\sigma_i \in S_C} \mathcal{P}(\sigma_i, Z) \cdot dim(\sigma_i) \cdot sim_{\sigma_i}(Sec_{a_i}, Sec_{b_i})}{\sum_{\sigma_i \in S_C} \mathcal{P}(\sigma_i, Z) \cdot dim(\sigma_i)} \quad (6.11)$$

La correction de *couverture* est définie par :

$$couv_Z(S_a, S_C) = \frac{\sum_{\sigma_i \in S_C} \mathcal{P}(\sigma_i, Z) \cdot dim(\sigma_i)}{\sum_{\sigma_i \in S_a} \mathcal{P}(\sigma_i, Z) \cdot dim(\sigma_i)} \quad (6.12)$$

et, la correction de *représentativité* est définie par :

$$\begin{aligned} repr_Z(trc_b) &= \frac{\sum_{\sigma_i \in S_b} \mathcal{P}(\sigma_i, Z) \cdot dim(\sigma_i)}{\sum_{\sigma_i \in S_Z} \mathcal{P}(\sigma_i, Z) \cdot dim(\sigma_i)} \\ &= \tau_Z(trc_b) \end{aligned} \quad (6.13)$$

Notons que la présence des facteurs de correction rendent cette mesure non-symétrique par rapport aux traces. Etant toujours appelée en respectant les rôles de trc_a (i.e., trace en cours) et trc_b (i.e., trace antérieure), cela ne pose pas de problème.

Dans cette mesure de similarité entre épisodes, nous avons choisi de pondérer les trois facteurs, i.e., la similarité sur la partie commune, la couverture et la représentativité par $\mathcal{P}(\sigma_i, Z) \cdot dim(\sigma_i)$.

Le facteur $\mathcal{P}(\sigma_i, Z)$ exprime combien son utilisation a été fructueuse jusqu'ici. Cette fonction de pondération utilise des données internes de l'état de langage en cours, dont dispose l'agent alter ego, c.f., 7.5.

Pour le facteur $dim(\sigma_i)$, nous adoptons la logique utilisée pour la définition du *taux d'identification* $\tau_Z(trc)$ (c.f., équation 6.8, ou encore, la similarité de signature, c.f., 6.6) : donner d'autant plus d'importance à un symbole qu'il est de grande dimension (i.e., qu'il contient beaucoup de captures d'OI). De la même façon, nous insistons sur le fait que ce principe de pondération ne joue ici que le rôle d'**heuristique**. Il conviendrait d'en faire l'évaluation par l'expérimentation et l'ajustement de son expression, qui pourrait diverger de celui fait pour les deux premières mesures, tout comme les ajustements qui seraient obtenus pour les trois facteurs donnés ci-dessus.

Cette réutilisation de mêmes principes de pondération du taux d'identification pour ceux de la représentativité, nous conduisent (dans un premier temps) à une expression de la représentativité de l'épisode antérieur pour Z identique à celle du taux d'identification de Z dans trc_b, i.e., $\tau_Z(trc_b)$.

Pour respecter la terminologie du RàPET, un épisode reconnu dans la trace en cours joue le rôle de partie problème cible ; mais, un épisode retrouvé dans une trace antérieure désigne dans son ensemble une partie problème source et une partie solution source, qui est candidate à la résolution du problème cible. La partie problème source a été jugée similaire à la partie problème cible, via $sim(S_C, trc_a, trc_b)$ dans l'équation 6.10, et la partie solution source candidate est représentée par les symboles restants, qui ont été reconnus dans la trace antérieure.

L'acquisition de la trace en cours fait partie du premier comportement de l'agent alter ego (c.f., figure 5.1), ainsi que les processus d'élaboration et de recherche d'épisodes, qui forment le module de RàPET.

Le deuxième comportement (c.f., figure 5.2) est déclenché lorsque l'acteur demande des suggestions à l'agent alter ego. Celui-ci lui soumet tout d'abord les interprétations qu'il a effectuées ;

l'utilisateur en choisi une ; les épisodes retrouvés dans des traces antérieures pour la signature associée à l'interprétation choisie sont proposés à l'acteur, ordonnés par similarité décroissante avec l'épisode reconnu dans la trace en cours. Dans notre première approche, nous laissons à l'utilisateur la tâche d'appropriation des suggestions de solution et celle d'adaptation de ces suggestions pour obtenir une solution cible. Nous traiterons en perspectives, c.f., 10.5, de l'achèvement du cycle de RàPET par l'agent alter ego.

Exemple :

Dans un premier temps, nous allons détailler le calcul de similarité entre deux sections de traces dans lesquelles un symbole σ_i a été perçu, on a donc $\sigma_i \in S_C$. Nous illustrons sur la figure 6.22 les objets impliqués dans ce calcul de similarité : deux sections de traces, Sec_{a_i} et Sec_{b_i}, qui ont chacune été appariées au motif d'un symbole σ_i ; les sections et le motif de symbole ont structure commune de la forme $E \cdot T \cdot E$, nous parlons de trois positions $pos \in \{Av, Op, Ap\}$ pour désigner les éléments de cette structure. Nous rappelons qu'un appariement d'un motif de symbole sur une section de trace n'est possible que si une solution d'appariement existe entre le pattern du symbole $Obs_{pattern_{pos}}$ et l'arbre d'observation de la section $Obs_{pos_{section}}$ pour chaque position pos (Av, Op et Ap) ; une solution d'appariement de symbole est représenté par un triplet formé d'une solution pour chaque position. La similarité entre deux sections appariées au motif de σ_i est une moyenne pondérée des similarités de leurs arbres d'observation appariés au pattern de σ_i pour toutes les positions.

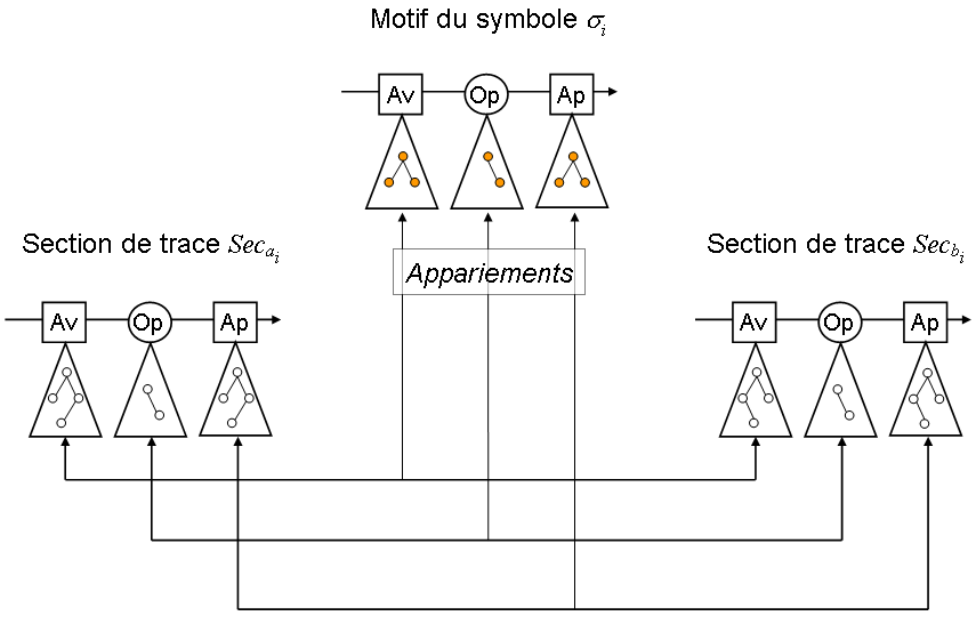


FIG. 6.22 – Exemple de calcul de similarité entre une section (de la trace en cours) et une section (d'une trace antérieure) appariées au motif d'un symbole σ_i

En considérant que :
- $sim_{Obs_{pattern_{Av}}}(Obs_{Av_{a_i}}, Obs_{Av_{b_i}}) = 0.7$ (s_{Av}) avec un poids $p_{Av} = 2$,
- $sim_{Obs_{pattern_{Op}}}(Obs_{Op_{a_i}}, Obs_{Op_{b_i}}) = 0.9$ (s_{Op}) avec un poids $p_{Op} = 3$,
- $sim_{Obs_{pattern_{Ap}}}(Obs_{Ap_{a_i}}, Obs_{Ap_{b_i}}) = 0.5$ (s_{Ap}) avec un poids $p_{Ap} = 2$,

on a :

$$sim_{\sigma_i}(Sec_{a_i}, Sec_{b_i}) = \frac{p_{Av} \cdot s_{Av} + p_{Op} \cdot s_{Op} + p_{Ap} \cdot s_A}{p_{Av} + p_{Op} + p_{Ap}} = \frac{2 \cdot 0.7 + 3 \cdot 0.9 + 2 \cdot 0.5}{2 + 3 + 2} \approx 0,73$$

Nous allons maintenant détailler le calcul de la similarité entre épisodes. Afin de mettre en évidence ses propriétés, nous appuyons notre présentation sur trois exemples décrits par la figure 6.23.

Soit une signature Z définie avec les symboles σ_1, σ_2, σ_3, σ_4 et σ_5, on note S_Z cet ensemble de symboles ; ils ont les dimensions $dim(\sigma_1) = 4$, $dim(\sigma_2) = 3$, $dim(\sigma_3) = 5$, $dim(\sigma_4) = 3$ et $dim(\sigma_5) = 6$. Les valeurs de pondération interne renvoyées par la fonction $\mathcal{P}$ sont : $\mathcal{P}(\sigma_1, Z) = 3$, $\mathcal{P}(\sigma_2, Z) = 5$, $\mathcal{P}(\sigma_3, Z) = 6$, $\mathcal{P}(\sigma_4, Z) = 4$ et $\mathcal{P}(\sigma_5, Z) = 3$.

Sur la figure 6.23, nous décrivons schématiquement les élaborations effectuées pour Z sur la trace en cours trc_a et sur trois traces antérieures trc_{b1}, trc_{b3} et trc_{b3} ; l'interprétation de la trace en cours est répétée à coté de celle de chaque trace antérieure. Les sections de trace où un symbole a été perçu sont grisées et encadrées en pointillés ; le lien entre deux cadres représente la perception du même symbole dans les deux sections. On a $S_a = \{\sigma_1, \sigma_2, \sigma_3\}$, et :

- pour trc_{b1} : $S_b = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5\}$ donc $S_C = \{\sigma_1, \sigma_2, \sigma_3\}$. On note $sim_{a1} = 0.7$ la similarité entre les deux sections où σ_1 a été perçu, $sim_{b1} = 0.8$ pour σ_2 et $sim_{c1} = 0.5$ pour σ_3 ;

- pour trc_{b2} : $S_b = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ donc $S_C = \{\sigma_1, \sigma_2, \sigma_3\}$. On note $sim_{a2} = 0.8$ la similarité entre les deux sections où σ_1 a été perçu, $sim_{b2} = 0.9$ pour σ_2 et $sim_{c2} = 0.5$ pour σ_3 ;

- pour trc_{b3} : $S_b = \{\sigma_1, \sigma_2, \sigma_4, \sigma_5\}$ donc $S_C = \{\sigma_1, \sigma_2\}$. On note $sim_{a3} = 0.9$ la similarité entre les deux sections où σ_1 a été perçu, $sim_{b3} = 0.8$ pour σ_2, mais σ_3 n'a pas été reconnu.

Sur ces exemples, nous souhaitons que l'épisode de trc_{b1} soit proposé en premier, donc le plus similaire à l'épisode en cours, car il "couvre" (en terme d'ensemble de symboles perçus) l'épisode en cours et car il "représente" le plus complètement la situation Z, i.e., tous les symboles de Z peuvent être perçus dans trc_{b1}. Après devrait venir l'épisode de trc_{b2}, car il couvre l'épisode courant mais il représente un peu moins Z, σ_5 n'étant pas présent. En dernier viendrait l'épisode de trc_{b3} car il ne couvre pas l'épisode courant, même s'il est représentatif de Z à un niveau comparable à l'épisode de trc_{b2}. En résumé, notre objectif est d'avoir :

$$sim_Z(trc_a, trc_{b1}) > sim_Z(trc_a, trc_{b2}) > sim_Z(trc_a, trc_{b3})$$

Afin d'obtenir l'ordre de suggestion, correspondant à un classement par similarité d'épisodes décroissante : **1)** trc_{b1}**, 2)** trc_{b2}**, 3)** trc_{b3}.

- Lors du calcul de la similarité pour la partie commune, on a :

  - pour trc_{b1}, comme $S_C = \{\sigma_1, \sigma_2, \sigma_3\}$:

$$sim_Z(S_C, trc_a, trc_{b1}) = \frac{3 \cdot 4 \cdot 0.7 + 5 \cdot 3 \cdot 0.8 + 6 \cdot 5 \cdot 0.5}{3 \cdot 4 + 5 \cdot 3 + 6 \cdot 5} \approx 0.62$$

  - pour trc_{b2}, comme $S_C = \{\sigma_1, \sigma_2, \sigma_3\}$:

$$sim_Z(S_C, trc_a, trc_{b2}) = \frac{3 \cdot 4 \cdot 0.8 + 5 \cdot 3 \cdot 0.9 + 6 \cdot 5 \cdot 0.5}{3 \cdot 4 + 5 \cdot 3 + 6 \cdot 5} \approx 0.67$$

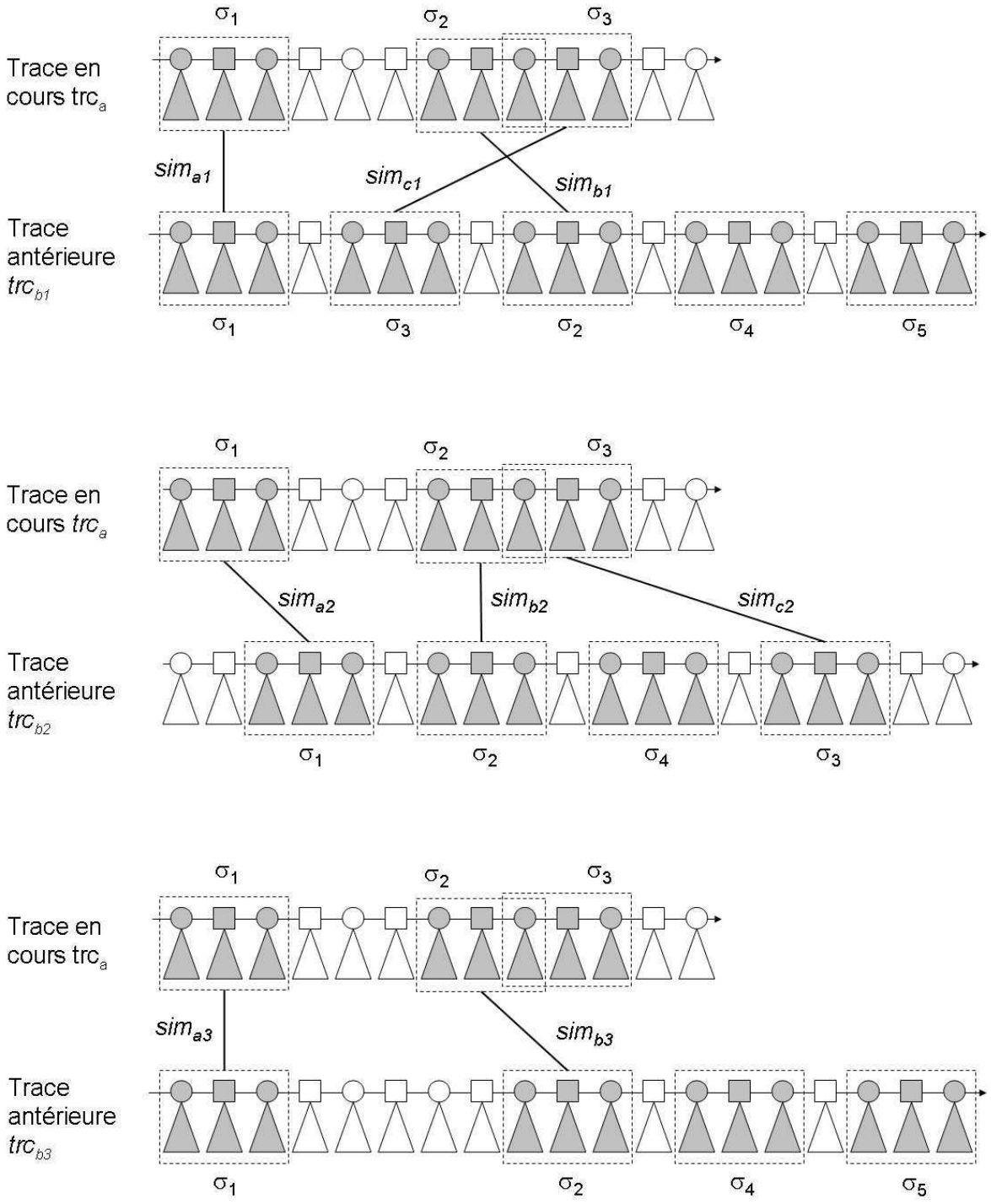


FIG. 6.23 – Exemples de calcul de similarité entre la trace en cours et une trace antérieure au regard de l'interprétation avec une signature Z, définie avec les symboles σ_1, σ_2, σ_3, σ_4 et σ_5

– pour trc_{b3}, comme $S_C = \{\sigma_1, \sigma_2, \}$:

$$sim_Z(S_C, trc_a, trc_{b3}) = \frac{3 \cdot 4 \cdot 0.9 + 5 \cdot 3 \cdot 0.8}{3 \cdot 4 + 5 \cdot 3} \approx 0.84$$

Si notre mesure de similarité se limitait à la similarité pour la partie commune, on aurait l'ordre de suggestion : **1)** trc_{b3}**, 2)** trc_{b2}**, 3)** trc_{b1}.

- Le facteur de correction de *couverture* nous donne :

  – pour trc_{b1}, comme $S_C = S_a$:

  $$couv_Z(S_C, S_a) = 1$$

  – pour trc_{b2}, comme $S_C = S_a$:

  $$couv_Z(S_C, S_a) = 1$$

  – pour trc_{b3}, comme $S_C \neq S_a$:

  $$couv_Z(S_C, S_a) = \frac{3 \cdot 4 + 5 \cdot 3}{3 \cdot 4 + 5 \cdot 3 + 6 \cdot 5} \approx 0.47$$

Si notre mesure de similarité était le produit de la similarité pour la partie commune et de la couverture, sans la représentativité, on aurait les valeurs de similarité avec l'épisode en cours suivantes :

– pour trc_{b1} : $0.62 \cdot 1 = 0.62$;

– pour trc_{b2} : $0.67 \cdot 1 = 0.67$;

– pour trc_{b3} : $0.84 \cdot 0.47 \approx 0.4$.

Cela impliquerait l'ordre de suggestion : **1)** trc_{b2}**, 2)** trc_{b1}**, 3)** trc_{b3}.

- Le facteur de correction de *représentativité* nous donne :

  – pour trc_{b1}, comme $S_b = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5\} = S_Z$:

  $$repr_Z(trc_{b1}) = 1$$

  – pour trc_{b2}, comme $S_b = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$:

  $$repr_Z(trc_{b2}) = \frac{3 \cdot 4 + 5 \cdot 3 + 6 \cdot 5 + 4 \cdot 3}{3 \cdot 4 + 5 \cdot 3 + 6 \cdot 5 + 4 \cdot 3 + 3 \cdot 6} \approx 0.79$$

  – pour trc_{b3}, comme $S_b = \{\sigma_1, \sigma_2, \sigma_4, \sigma_5\}$:

  $$repr_Z(trc_{b3}) = \frac{3 \cdot 4 + 5 \cdot 3 + 4 \cdot 3 + 3 \cdot 6}{3 \cdot 4 + 5 \cdot 3 + 6 \cdot 5 + 4 \cdot 3 + 3 \cdot 6} \approx 0.66$$

Notre mesure de similarité, produit de la similarité pour la partie commune, de la couverture et de la représentativité, nous donne les valeurs de similarité avec l'épisode en cours suivantes :

- pour trc_{b1}, $sim_Z(trc_a, trc_{b1}) = 0.62 \cdot 1 \cdot 1 = 0.62$;
- pour trc_{b2}, $sim_Z(trc_a, trc_{b2}) = 0.67 \cdot 1 \cdot 0.79 \approx 0.53$;
- pour trc_{b3}, $sim_Z(trc_a, trc_{b3}) = 0.84 \cdot 0.47 \cdot 0.66 \approx 0.26$.

Cela implique l'ordre de suggestion que nous recherchons : **1)** trc_{b1}**, 2)** trc_{b2}**, 3)** trc_{b3}.

6.7 Récapitulatif

Dans ce chapitre, nous avons présenté l'ensemble des modèles et des mécanismes qui nous permettent de doter notre agent alter ego d'un module de RàPET.

Nous sommes partis du modèle MUSETTE, qui nous a servi de guide pour construire les traces, et dans lequel nous donnons un rôle central aux signatures de tâches. Un modèle de domaine, c.f., 6.1, a été introduit ; il donne à l'utilisateur le moyen d'exprimer, au moment où il désigne un élément, à quel degré d'abstraction cet élément doit être considéré pour caractériser sa tâche. A partir de cela, nous avons présenté en 6.2.1 notre modèle pour les observations de trace ; de ce modèle découle notre modèle de pattern (i.e., une observation de symbole), utilisé pour décrire des indications données par l'utilisateur sur des éléments d'une observation de trace. Nous avons détaillé en 6.3 notre modèle pour construire une trace primitive, en vue de sa reformulation symbolique, par l'ajout de marqueurs définis dans un modèle des fonctionnalités de l'application cible ; le modèle formel de la trace primitive permet de "découper" la trace en sections, qui sont de forme $E \cdot T \cdot E$ autour d'un marqueur contenu dans la transition ; une section est l'entité de reformulation symbolique d'une trace. Dans sa reformulation symbolique, l'agent alter ego cherche à réutiliser les éléments symboliques d'élaboration qu'il a acquis de l'utilisateur : pour chaque section de trace, isolée par un marqueur, l'agent alter ego cherche les symboles de langage qui peuvent être perçus, i.e., ceux dont les motifs peuvent être appariés à la section. Dans notre application du RàPET, les signatures de tâche représentent les situations d'assistance. Nous avons besoin dans notre approche d'une expression symbolique des signatures de tâches ; nous la définissons, en 6.4, sous la forme d'un ensemble non ordonné de symboles de langage, auquel est associé son sens défini en langage naturel par l'utilisateur. Les symboles et les signatures (définies symboliquement) utilisés par l'agent alter ego à un moment donné sont tous décrits dans un même modèle, appelé un *état de langage*. Les deux premières étapes de RàPET sont réalisées à partir d'un unique état de langage, appelé l'*état de langage en cours*.

Nous avons détaillé, en 6.6.1, le mécanisme d'élaboration de notre moteur de RàPET qui a pour but d'identifier des situations d'assistance candidates dans la trace en cours ; cela passe par la construction d'un épisode cible pour différentes signatures susceptibles d'être reconnues dès lors qu'un de leur symbole est perçu. La construction d'un épisode cible pour une signature se fait en identifiant les section de la trace en cours où des symboles, définissant cette signature, ont été perçus. Une signature devient candidate (pour l'étape suivante de RàPET : la recherche d'épisodes source) lorsque son taux d'identification dépasse le seuil arbitraire τ_{idSign}.

Pour une signature Z candidate à la recherche d'épisodes, le même mécanisme d'élaboration est utilisé pour construire un épisode source dans une trace antérieure. En 6.6.2, nous avons présenté une mesure de similarité entre épisodes source et cible. Les épisodes sources construits par l'agent alter ego sont proposés à l'utilisateur, en les classant par similarité décroissante avec

l'épisode cible.

Les signatures de tâches sont au cœur des mécanismes permettant d'effectuer des rapprochements entre des épisodes reconnus dans les traces. En effet, les différentes mesures introduites dans ce chapitre dépendent toutes de données relevant des signatures de tâche. Nous avons introduit un niveau symbolique dans l'expression des traces, qui est à la base de l'expression des signatures.

Afin que ces rapprochements aient un sens pour l'utilisateur, nous allons présenter dans le chapitre suivant les mécanismes que nous proposons pour acquérir, en situation, ces signatures de tâches, à partir des indications données par l'utilisateur. Les principes d'émergence de langage (de forme lexicale) vont constituer notre paradigme pour gérer cette acquisition des signatures de tâche.

Chapitre 7

Co-construire le sens par négociation pour réutiliser l'expérience tracée

Sommaire

Nous venons de présenter les principes de notre système proactif d'assistance (l'*agent alter ego*), facilitant le rapprochement entre la situation en cours et des situations antérieures similaires, selon les principes du RàPET. Toutes ces opérations se faisant en utilisant des connaissances d'élaboration qui ont déjà été acquises de l'utilisateur.

La description des situations est faite sur la base de traces d'utilisation, les traces primitives ; cette description est utilisée pour la situation en cours d'acquisition, via la trace en cours, et les situations déjà rencontrées, via les traces antérieures.

Les principes d'acquisition et d'évolution des éléments d'élaboration, que nous allons présenter, nous demandent de disposer d'une représentation symbolique. Apparemment contraignante, cette approche symbolique joue le rôle de médium de communication entre l'acteur humain et son agent alter ego ; pour l'acteur humain, elle contribue à la réflexivité de la trace et constitue un support à son activité méta-cognitive.

Cependant, en reprenant la problématique introduite en 4.5, une hypothèse forte de notre approche d'assistance par RàPET est que l'agent alter ego dispose de descriptions significatives pour l'acteur humain des situations demandant de l'assistance, et nous considérons qu'il est impossible d'en effectuer une description a priori qui soit complète et durable. Nous allons présenter dans ce chapitre les mécanismes que nous proposons pour acquérir, en situation, ces signatures de tâches, à partir des indications données par l'utilisateur. Les principes que nous proposons ont fait l'objet d'une publication, c.f., [Stuber 05], alors qu'ils n'étaient pas aussi précisément définis qu'ils vont l'être ici. Néanmoins, l'idée première persiste ; elle est de considérer les principes d'émergence de langage (de forme lexicale) comme à la fois comme paradigme et comme modèle

théorique, permettant l'acquisition des signatures de tâche et des éléments symboliques qui les composent.

Nous partons de l'hypothèse qu'aucune signature n'est disponible au départ ; cette hypothèse est faite dans la logique théorique de notre transposition des principes d'émergence de langage. Toutefois, dans une utilisation pratique, il conviendrait de la relâcher, en partant des éléments de langage qui relèvent du consensus, sans que cela ne vienne remettre en cause les principes que nous allons présenter ; ceci principalement afin d'éviter à un utilisateur novice d'avoir à effectuer une longue initialisation de l'état de langage de son assistant.

Dans cette perspective théorique, nous étudions les moyens pour l'agent alter ego d'acquérir des éléments d'élaboration à partir des indications de l'utilisateur, dans le but de faire évoluer son état de langage. Ces indications peuvent être données explicitement par l'utilisateur, mais nous allons chercher également à tirer parti d'indications implicites aux actions de l'utilisateur.

En premier lieu parmi les indications explicites, nous avons les sens d'assistance. En effet, il n'est pas envisageable de demander à l'agent alter ego une totale autonomie dans l'identification des sens d'assistance ; nous traiterons en perspective, c.f., 10.3, de la possibilité d'une relative proactivité de l'agent ego pour cette tâche. De plus, nous avons souligné dans notre présentation des *agents interface*, c.f., 4.2, que l'utilisateur exigeait de garder le contrôle de son agent assistant afin de lui accorder sa confiance et que les agents interface ne sont pas appropriés pour intervenir dans des situations critiques.

Pour ces raisons, l'approche que nous présentons ici se fait en laissant à l'utilisateur la tâche d'énoncer, en langage naturel, les sens d'assistance qui l'intéressent ; de ce point de vue, il s'agit pour l'agent alter ego d'un apprentissage supervisé par l'utilisateur. Chaque sens d'assistance est représenté par une signature, dont le label exprime sa signification en langage naturel et est fixé par l'utilisateur. La tâche de l'agent alter ego est de trouver une définition de ce sens via un ensemble de symboles ; cette même définition lui permet d'"exprimer" le sens d'une élaboration de la situation courante avec une signature et par là même la logique de rapprochement avec des situations antérieures.

L'idée principale de cette partie du travail est de profiter des moments où l'utilisateur interagit directement avec son agent alter ego, i.e., lors des suggestions de rapprochement, pour acquérir de façon opportuniste de nouveaux éléments symboliques, à la base de l'expression des signatures. La section suivante présente les motivations qui nous guident à transposer les mécanismes d'émergence de langage lexical afin de gérer les symboles acquis et de les confronter à la réutilisation dans le cadre de l'expression des signatures.

7.1 Jeux de langage pour la négociation de sens

Nous avons argumenté en 4.5 que la condition fondamentale du RàPET est d'avoir, ou plutôt d'acquérir car la définir à l'avance est infaisable, une description significative d'une signature ayant du sens pour l'utilisateur et utile à l'agent alter ego, afin d'effectuer en situation des rapprochements d'épisodes pertinents. Nous avons effectué le rapprochement de cet objectif avec la problématique de construction d'un système de communication de forme lexicale, i.e., formé de mots et sens, entre agents situés en interaction, et ce sans qu'aucun élément de langage soit prédéfini.

Dans notre cas, nous ne pouvons parler d'aucun élément prédéfini, car nous considérons que les symboles génériques ne font pas partie du langage ; ils ne peuvent pas être utilisés dans une

signature, ils ont uniquement un rôle d'expression symbolique par défaut pour le marqueur présent dans une section. Cela peut être vu comme un mode de communication *non verbal* pour l'agent alter ego, qui lui permet de désigner une opération.

Notre transposition des mécanismes d'émergence de langage se fait en considérant l'acteur humain et son agent alter ego comme un système multi-agent hybride, composé de deux agents. Ces deux agents sont immergés dans la tâche en cours, qui est représentée par la trace en cours. Nous rappelons que cette trace est construite par l'observation d'un environnement idéalement ouvert, i.e., une application cible permettant de manipuler des entités documentaires décrites avec des mots-clés spécifiques au domaine. Les deux agents ont des objectifs distincts : 1) l'acteur humain doit réaliser sa tâche avec succès et 2) l'agent alter ego doit interpréter la situation courante afin de proposer des rapprochements avec des situations antérieures. Ces objectifs impliquent un intérêt à la collaboration entre ces agents, mais ils entraînent surtout des actions distinctes pour les deux agents : 1) la réalisation d'opération sur l'application cible pour l'acteur humain et 2) pour l'agent alter ego, l'élaboration d'épisodes cibles pour la situation en cours, au regard de signatures candidates, et la recherche d'épisodes antérieurs.

Il n'est bien entendu pas envisageable de penser atteindre la complexité humaine dans l'expression d'un sens, a fortiori sur la base de notre modèle de signature qui n'est qu'un ensemble de symboles ; il n'est non plus pas envisageable de demander à l'acteur humain de traiter directement son propre modèle de langage. Notre transposition des jeux de langage se place au niveau de l'agent alter ego, afin de lui permettre de gérer l'utilisation des symboles dans l'"expression" d'une signature.

Dans cette approche, nous plaçons l'utilisateur au centre du système et nous comptons sur l'appropriation qu'il se fait du système pour "donner du sens" à l'identification d'une signature dans le contexte de la situation courante. Les limites de son appropriation, celles des modèles de signature et de symbole et l'ouverture des situations, qui peuvent être rencontrées, agissent comme des contraintes sur le langage de l'agent alter ego. Dans notre approche, nous avons d'un côté un agent humain et son langage propre, que nous ne cherchons pas à modéliser, et un agent alter ego avec un langage de forme lexicale, dont les sens (les signature) sont supervisés par l'acteur humain. Nous cherchons à acquérir, dans le langage de l'agent alter ego, des "expressions symboliques" de sens humains afin que l'agent *alter ego* reflète le mieux les logiques d'action de l'agent humain. Cette acquisition passe par la construction successive d'états de langage, à partir de l'état de langage initial L_0 vide.

Pour effectuer cette acquisition de sens, nous assimilons la trace en cours, sous sa forme symbolique, à un "discours" de l'agent humain décrivant la situation dans laquelle il se trouve. En cherchant à construire des épisodes cibles, l'agent alter ego réalise une sorte d'"interprétation" de ce discours symbolique afin d'*interpréter* des signatures, selon le nom donné à la deuxième phase de notre mécanisme d'élaboration.

Le principe, qui guide notre transposition des mécanismes d'émergence de langage, est de considérer schématiquement comme :

- **l'initiation d'une tentative de communication** entre l'agent alter ego et l'acteur humain : toute consultation par l'agent humain des suggestions d'épisodes antérieurs de l'agent alter ego, obtenus après l'élaboration d'un épisode cible pour une signature ;

- **la cloture de cette tentative de communication**, le moment ou l'acteur humain souhaite soit :

- mettre fin à cette consultation en émettant un avis sur l'élaboration. L'agent alter ego construit un nouvel état de langage en fonction de cela ;
- que l'agent alter ego mette à jour ses propositions en tenant compte de ses réactions. Dans ce cas, nous allons introduire ci-dessous un mécanisme de négociation permettant à l'agent alter ego de collecter des informations supplémentaires afin de construire un nouvel état de langage. Cela est nécessaire pour intégrer les réactions et de refléter le succès ou l'échec de l'utilisation des éléments de l'état de langage en cours, qui ont été impliqués dans cette tentative (de communication, avec pour sujet un sens d'assistance). Ce mécanisme de négociation permet également à l'agent alter ego de s'assurer de la signature qui est visée par l'utilisateur. C'est elle qui sera considérée pour la mise à jour des épisodes sources élaborés avec le nouvel état de langage. Nous voyons cette mise à jour comme l'initiation d'une nouvelle tentative de communication au sujet de la signature négociée.

Nous voulons que l'agent alter ego se serve des interactions avec l'utilisateur pour améliorer ses descriptions de signature et de symbole. Néanmoins, les réactions de l'utilisateur sont par nature ambiguës pour l'agent alter ego, car :

- l'agent alter ego ignore si une intervention de l'utilisateur sur un symbole ou une signature doit être considérée comme une création d'un nouvel élément de langage ou bien son évolution ;

- toute modification d'un motif de symbole peut conduire à un motif identique à celui d'un autre symbole, et il en est de même pour la définition d'une signature en terme d'ensemble de symboles. Nous considérons chacune de ces deux situations comme ambiguës pour l'agent alter ego, car nous estimons qu'il sait qu'il doit parler de deux choses distinctes mais qu'il ne dispose d'aucun élément de distinction. Seul l'acteur humain peut apporter des éléments distinctifs complémentaires, ou bien indiquer à l'agent alter ego qu'il s'agit de la même chose.

Pour résoudre ces ambiguïtés, l'agent alter ego doit disposer d'un mécanisme interactif permettant de négocier avec l'utilisateur le sens des différents éléments des symboles créés ou modifiés, puis de manière très directe de la définition symbolique de la signature à considérer pour la tentative suivante.

De manière générale, ce processus de négociation consiste à convenir, en situation, d'un moyen partagé pour exprimer une situation d'assistance (la sémantique d'une signature, exprimée en langage naturel par l'utilisateur) à l'aide d'un ensemble de signes (les symboles, avec un sens propre à chacun exprimé par leur label en langage naturel par l'utilisateur).

Avant de détailler notre transposition, nous allons rappeler les principes d'émergence d'un langage de forme lexicale, i.e., des sens et des mots pour parler de ces sens, détaillés en 4.6. Nous faisons cela pour définir la notion d'état de langage, et pour définir clairement le rôle que nous donnons aux éléments d'élaboration (symboles et signatures), introduits précédemment. Nous allons introduire différentes pondérations, qui vont nous permettre de définir la fonction $\mathcal{P}(.,.)$ utilisée en 6.6.1, lors de la définition du taux d'identification, ainsi qu'en 6.6.2 lors de la définition de la similarité entre épisodes. Ces pondérations sont au centre des mécanismes d'émergence de langage ; elles sont utilisées dans les itérations de jeux de langage pour effectuer des renforcements ou des inhibitions des différents éléments de langage, intervenant dans une

tentative de communication ; elles sont au centre des mécanismes de pression sélective.

7.2 Modèle d'état de langage

Nous avons présenté en 4.6.2 une synthèse des travaux récents sur la modélisation des origines des systèmes de communication, effectuée par [Steels 00]. Les travaux présentés sont principalement appliqués au domaine de la robotique ; [Steels 99] est un ouvrage entier consacré à des travaux sur la construction de mots et de sens dans ce domaine.

Selon [Steels 00], il apparaît que l'émergence de langage est basée sur un accord, obtenu entre deux agents, portant sur :

- un *répertoire de symboles*, utilisé comme "medium de communication" entre les agents. Ce medium est le résultat de l'utilisation avec succès de capacités sensori-motrices des agents avec leurs contraintes propres dans le but de "construire" un médium de communication, le cas de l'émergence de voyelles [De Boer 97] constitue un exemple caractéristique de cette opération ;

- un *répertoire de sens*, pour chaque agent ; ce répertoire représente les abstractions de la réalité faites par un agent ;

- un *répertoire de paires symbole-sens*, pour chaque agent, afin d'être en mesure d'interagir avec succès. Lorsqu'un agent veut interagir avec l'autre agent à propos d'un sens, il cherche dans son répertoire le symbole le plus approprié. Une tentative de communication est initiée par le premier agent, lorsqu'il "exprime" le sens qu'il vise pour l'interaction en transmettant ce symbole à l'autre agent. Ce deuxième agent fait l'opération inverse en cherchant, parmi ses sens, le sens qu'il a convenu de désigner par ce symbole ; puis, il agit en fonction du sens identifié. Si l'action du deuxième agent est compatible avec le sens visé par le premier agent, la tentative de communication est un succès, sinon c'est un échec.

Les mécanismes interactifs élémentaires permettant cet accord sont les jeux de langage, qui sont respectivement le *jeu d'imitation*, le *jeu de discrimination* et le *jeu de nommage*. Disposer d'un tel accord permet aux agents d'interagir avec succès en "parlant" d'une signature de sens à l'aide d'un symbole.

Cet accord, qui est en réalité un ensemble de trois accords interdépendants, est obtenu en appliquant 1) le *principe de renforcement* et d'*inhibition* aux éléments manipulés au cours des itérations des différents jeux. 2) L'*auto-organisation* des répertoires de symboles et de paires symboles-sens est due à la boucle de rétro-action positive entre l'utilisation et le succès d'une itération de jeu associé. 3) Le *sélectionisme* est globalement porté par les limites des capacités sensori-motrices des agents, elle est associée à une diversification due à la variabilité des situations rencontrées, ouvertes par principe. Pour le jeu de nommage, une quatrième propriété, appelée *couplage structurel*, est issue de la co-évolution des répertoires de symboles et de sens ; une bonne part du caractère adaptatif d'un langage émergent vient de cette propriété.

Dans notre transposition, le label en langage naturel, défini par l'utilisateur, d'une signature expliquée de tâche, au sens du RàPET, représente un sens d'un état de langage. Tout naturel-

lement, les éléments symboliques d'élaboration (les symboles, chacun avec un label en langage naturel et un motif) sont représentés par les symboles d'un état de langage. Les relations de composition, permettant de définir symboliquement une signature par un ensemble de symboles, sont représentées par des relations symbole-sens dans un état de langage. Pour enregistrer les renforcements et les inhibitions, une pondération doit être associée à chaque symbole, à chaque signature et à chaque relation symbole-signature. Ces répertoires avec leurs éléments pondérés vont évoluer selon les résultats des itérations successives dans notre transposition des jeux de langage, et la pondération va conditionner la survie d'un élément de répertoire.

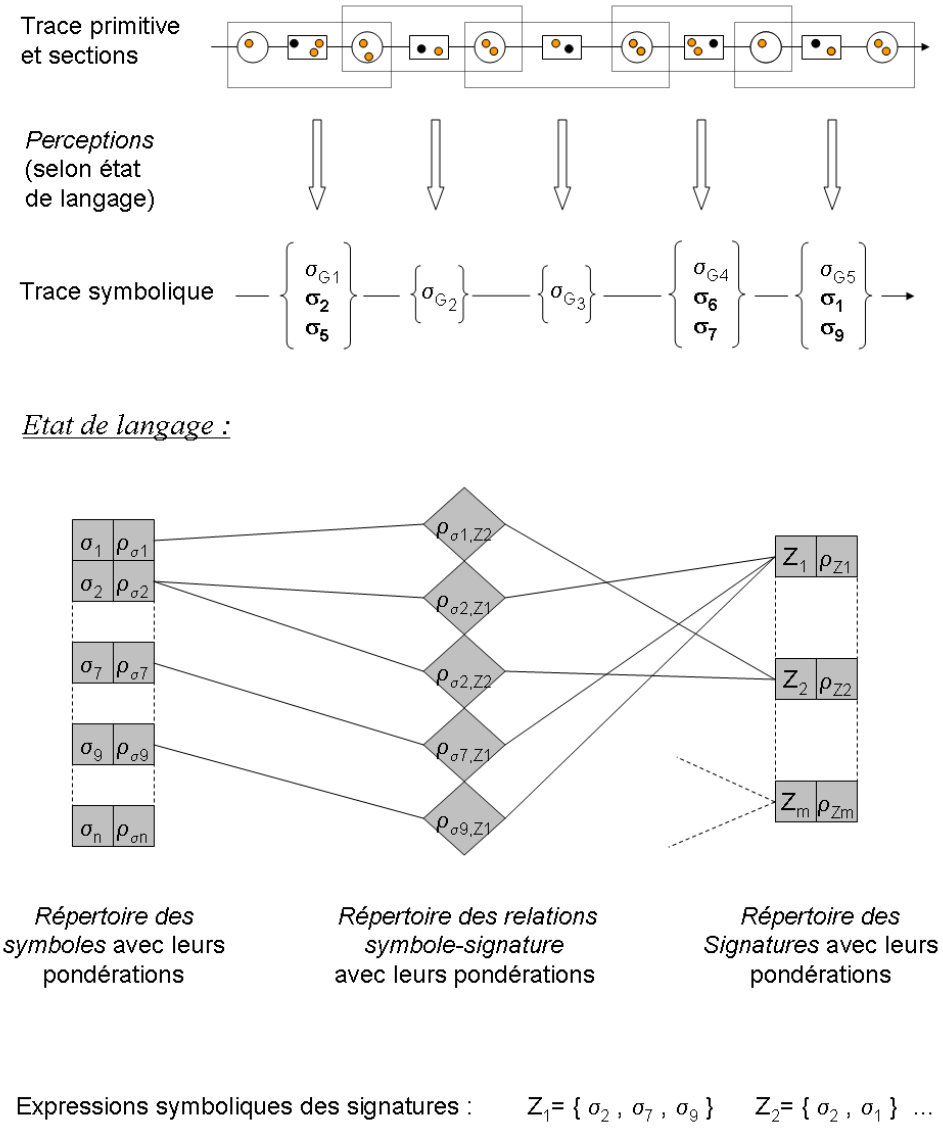


FIG. 7.1 – Les éléments d'un état de langage et leurs pondérations respectives

La partie inférieure de la figure 7.1 illustre les informations constituant un *état de langage* : un répertoire de symboles avec leurs pondérations, un répertoire de signatures avec leurs pondérations et un répertoires de relation symbole-signature avec leurs pondérations. La partie supérieure de la figure 7.1 rappelle schématiquement la structure d'une trace primitive et sa reformulation

en trace symbolique à partir de l'état de langage donné en dessous.

Nous attirons l'attention sur le fait que notre transposition utilise un ensemble de relations symbole-signature pour "faire le lien" entre une signature et son expression symbolique ; en cela, nous nous écartons du modèle d'état de langage de [Steels 00], où un agent cherche à convenir d'une unique relation (un *lien*) pour une signature. Notre approche ne va pas en contradiction avec cela, car nous considérons que c'est l'ensemble de nos relations symbole-signature pour une signature qui forme le *lien* entre une "expression" symbolique et une signature ; les relations symbole-signature d'un tel ensemble ne seront pas en compétition. Cette variante vient du besoin légitime, dans notre cas, d'exprimer une signature au moins à l'aide d'un ensemble de symboles ; nous discuterons de cela plus en détail en 7.5.

7.3 Structuration de l'assistance

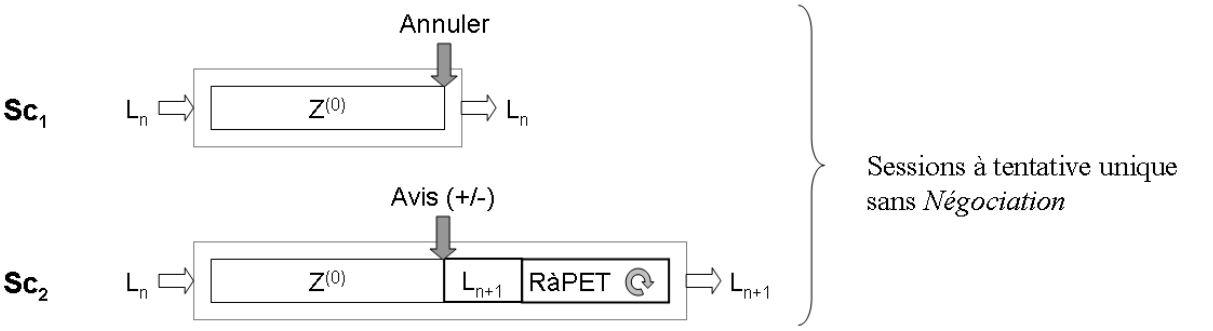
Pour structurer les interactions entre l'utilisateur et l'agent alter ego, nous introduisons la notion de *session d'assistance*. Une session d'assistance débute lorsque l'utilisateur consulte les rapprochements trouvés pour une signature, via une interface spécifique, et se termine lorsque cette interface est close.

Selon notre approche à base de langage émergent, introduite en 7.1, une session d'assistance initie une "tentative de communication" et se termine par la cloture d'une "tentative de communication", la même ou une autre ; cela dépend des réactions de l'utilisateur. Le terme de "tentative de communication" a été utilisé pour bien expliciter notre transposition, dans la suite nous parlerons plutôt de *tentative d'assistance*, car cela est l'objectif de notre agent alter ego. La mise à disposition des rapprochements pour une signature $Z^{(0)}$, dans un état de langage L_n, est le tronc commun de la première *tentative d'assistance* d'une session d'assistance, la bifurcation s'effectue lorsque l'utilisateur met fin à cette première tentative, soit :

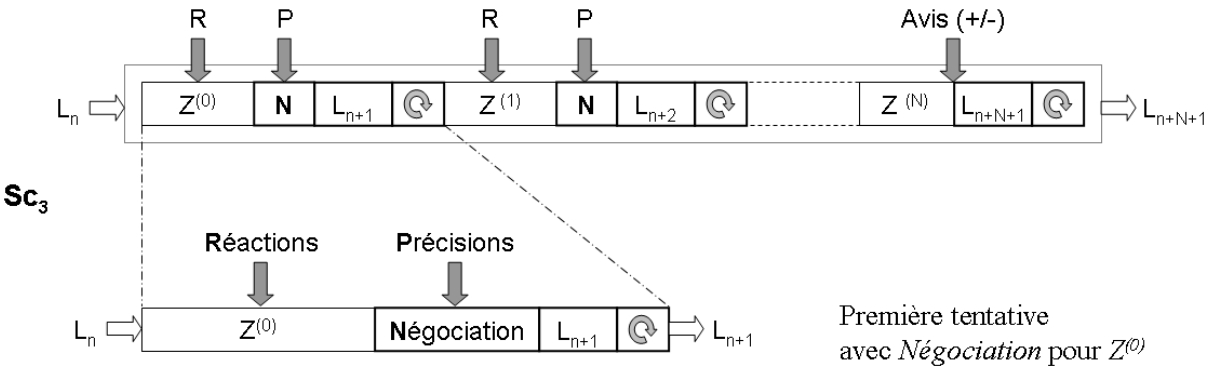
1. en cloturant l'interface de suggestion d'expérience, sans donner d'avis sur l'élaboration effectuée ;
2. en cloturant l'interface de suggestion et en donnant un avis favorable ou défavorable sur l'élaboration effectuée ;
3. en demandant à l'agent alter ego de mettre à jour les suggestions en tenant compte de ses réactions. Cette mise à jour fait appel au processus de négociation, afin de lever les ambiguïtés issues des actions de l'utilisateur et de leur influence sur le langage (motifs de symboles identiques, définitions symboliques de signatures identiques). Cette négociation permet de construire un état de langage L_{n+1} et de redéfinir la signature $Z^{(1)}$ visée par l'utilisateur pour la tentative d'assistance suivante. Cette nouvelle tentative est initiée pour afficher l'élaboration de la trace en cours effectuée avec $Z^{(1)}$ de L_{n+1} et les épisodes retrouvés. A la fin de cette deuxième tentative et pour toutes les tentatives suivantes, nous aurons les alternatives 2) et 3), respectivement de suite ou de fin de la session d'assistance, que nous venons d'énoncer.

La figure 7.2 illustre les trois scenari de session d'assistance, que nous considérons dans notre transposition :

- **scénario $\mathbf{Sc_1}$ sans négociation :** la session est composée d'une seule tentative sur la base de l'état de langage L_n, et l'acteur n'émet pas d'avis sur la tentative. Aucune modification



• Session à tentatives multiples avec *Négociation*:



• Alternatives d'enchaînement des tentatives dans Sc_3 :

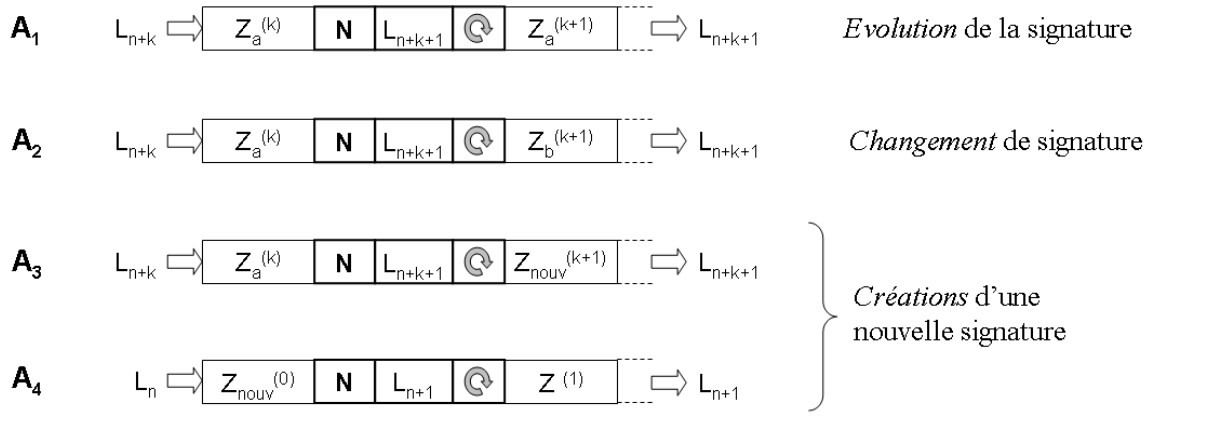


FIG. 7.2 – Scenari de sessions d'assistance et alternatives d'enchaînement de tentatives d'assistance pour les sessions à tentatives multiples

de l'état de langage n'est effectuée, c'est donc l'état L_n qui continuera d'être utilisé par l'agent alter ego. Cette situation est introduite pour laisser à l'utilisateur le moyen consulter le détail d'une élaboration et les épisodes rapprochés sans influencer le langage ;

- **scénario Sc_2 sans négociation :** la session est composée d'une seule tentative sur la base de l'état de langage L_n, et l'acteur donne un avis favorable ou défavorable à cette tentative. Cela conduit à la modification du langage, qui est inscrite dans un nouvel état de langage L_{n+1}. L'appel au module de RàPET, représenté par le logo  , traduit la prise en compte de L_{n+1} dans la suite des élaborations, qui sont effectuées en tâche de fond par l'agent alter ego. L'acteur continue sa tâche sur l'application cible et l'agent alter ego complète ses élaborations d'épisodes cibles pour la situation courante avec les signatures de L_{n+1} candidates ;

- **scénario Sc_3 avec négociation :** la session est composée d'une suite de $N+1$ tentatives. Cette session débute avec l'état de langage L_n et se termine avec l'état de langage L_{n+N+1}. Les N premières tentatives (sauf la dernière) sont avec négociation car elles correspondent à des demandes de mise à jour des propositions d'épisodes par l'utilisateur ; un nouvel état de langage est construit à chaque tentative. La session se termine sur une tentative sans négociation lorsque l'utilisateur ferme l'interface de suggestion d'expérience ; l'agent alter ego lui demande son avis, favorable ou défavorable, sur la dernière élaboration. Cette dernière tentative sera traitée comme le scénario Sc_2, i.e., tentative unique avec avis de l'utilisateur, et conduit à la construction du dernier état de langage L_{n+N+1}.

Le scénario Sc_1 est sans intérêt pour l'agent alter ego, car il n'apporte aucune information, mais il est nécessaire de considérer ce cas d'utilisation du point de vue de l'utilisateur. Le scénario Sc_2 apporte à l'agent alter ego, via l'avis de l'utilisateur, une estimation globale de la qualité de l'expression symbolique de la signature, mais aucun élément d'élaboration (symboles ou signature) n'est modifié. De cela, nous ne pouvons qu'en déduire le renforcement ou l'inhibition des tous les éléments de langage impliqués.

Le scénario Sc_3 est le plus riche pour l'agent alter ego ; par définition, il a lieu lorsque l'utilisateur a réagi avec les descriptions des symboles et qu'il souhaite la mise à jour des propositions de rapprochement par l'agent alter ego, en tenant compte de ses réactions. Pour réaliser une mise à jour, l'agent alter ego doit lever les ambiguïtés issues des réactions de l'utilisateur ; pour cela, l'agent alter ego dispose d'un mécanisme de négociation. Les réactions de l'utilisateur et les précisions apportées par l'utilisateur durant la négociation sont des sources privilégiés d'information pour permettre à l'agent alter ego de faire évoluer son langage.

Nous numérotons par k de 0 à N les tentatives d'une session à $N+1$ tentatives, selon le scénario Sc_3. Les tentatives $k \in [\![0; N-1]\!]$ sont avec négociation et la dernière ($k = N$) est sans négociation. La tentative k avec négociation ($k \in [\![0; N-1]\!]$) porte sur la signature $Z^{(k)}$ et elle est composée de :

- une proposition d'élaboration d'un épisode cible pour la trace en cours, au regard de la définition de $Z^{(k)}$ dans L_{n+k}, avec les épisodes antérieurs retrouvés pour cette élaboration. L'utilisateur peut réagir avec les différents symboles de la trace symbolique pour $Z^{(k)}$. Les réactions peuvent donc porter soit :

  1. sur des sections où seuls les symboles génériques, associés aux marqueurs des sections,

sont présents, et ce par défaut ;

2. sur des sections où des symboles du langage ont été perçus.

Selon cette distinction, une *réaction* est représentée par :

1. un triplet $(section, label, motif)$, où *section* désigne la section sur laquelle porte la réaction, *label* est le sens en langage naturel que donne l'utilisateur au symbole, qu'il veut en principe créer, et $motif$ est la retranscription, en motif de symbole, des indications qu'il a donné sur cette section. Les indications de l'utilisateur peuvent être de considérer un *objet d'intérêt* de la section comme important soit 1) par sa présence en tant que telle, il sera donc représenté par une capture constante dans un des arbres d'observation composant le motif, soit 2) par le fait qu'il est représentatif pour une abstraction à un certain degré, selon le modèle de domaine, c.f., 6.1 ; il sera représenté par une capture variable, pointant sur le concept définissant l'abstraction, dans un des arbres d'observation composant le motif ;
2. un quadruplet $(section, symbole, label, motif)$, où *section* désigne la section sur laquelle porte la réaction, *label* est le sens en langage naturel (éventuellement nouveau) que donne l'utilisateur à son intervention sur la définition du symbole, et $motif$ est la retranscription, en motif de symbole, des indications qu'il a donné sur cette section où le symbole a été perçu ; ce motif part du motif d'origine du symbole et intègre les indications de l'utilisateur. Cette retranscription est faite de la même manière que dans le cas précédent.

Dans notre description algorithmique qui va suivre, nous représenterons ces deux types de *réactions* par une même structure. Toutes les ***R**éactions* de l'utilisateur sur les éléments de la trace symbolique en cours ou sur ceux d'une trace symbolique antérieure sont enregistrées ;

– la ***N**égociation*, menée par l'agent alter ego, de toutes les réactions enregistrées, par rapport à l'état de langage L_{n+k}. Nous avons déjà abordé les ambiguïtés, pour l'agent alter ego, qu'entraînent ces réactions ; selon les types de réactions, introduits ci-dessus, l'agent alter ego a besoin de savoir si :

1. une réaction sur une section, où seulement un symbole générique est "reconnu", n'a pas conduit à un motif identique à celui d'un symbole existant déjà dans L_{n+k}. Si tel est le cas, cela constitue, du point de vue de l'agent alter ego, une *ambiguïté de symboles*. Il demande à l'utilisateur convenir d'un label commun et le symbole retenu sera le symbole déjà présent dans L_{n+k} avec ce nouveau label. Si tel n'est pas le cas, un nouveau symbole sera créé, défini avec le label donné par l'utilisateur et le motif décrivant la réaction ;
2. une réaction sur une section où un symbole de langage est perçu, doit être considérée soit 1) comme une évolution de ce symbole, i.e., l'affinement de sa description, soit 2) comme une création d'un nouveau symbole avec un sens propre. L'utilisateur doit préciser cela. En complément de cela, la réaction peut conduire, dans les deux cas, à une *ambiguïté de symboles*, i.e., le motif décrivant la réaction est identique à celui d'un symbole de L_{n+k}. Dans le cas 1) d'évolution de symbole, si le nouveau motif du symbole n'est pas ambigu, un nouveau symbole sera créé, avec le label du symbole d'origine et le motif de la réaction ; s'il y a ambiguïté, le symbole de L_{n+k} sera retenu, après convention d'un label commun. Dans le cas 2), cela est traité de la même manière qu'une création de symbole pour une réaction sur une section appariée uniquement à un symbole générique, en incluant le traitement de l'ambiguïté de symboles.

Les réactions de création et d'évolution de symbole sont traitées de la même manière : un nouveau symbole de langage est créé. Nous faisons cela parce que nous considérons qu'une réaction menant à l'évolution d'un symbole doit rester spécifique à la signature cible de la négociation ; en effet, ce symbole peut être utilisé dans l'expression d'autres signatures, si nous remplaçons le motif d'origine dans L_{n+k} par le motif de la réaction afin de redéfinir le symbole dans l'état de langage suivant, cette modification affecterait toutes les signatures utilisant ce symbole. La distinction entre création et évolution se situera au niveau de la gestion de leurs pondérations, tel que nous le présenterons en 7.5.

La deuxième phase de la ***Négociation*** consiste à lever l'ambiguïté, pour l'agent alter ego, sur le "rôle" des *réactions* par rapport à la signature, soit 1) il s'agit d'une évolution de l'expression symbolique de la signature négociée, soit 2) il s'agit de la création d'une nouvelle signature. En supplément de cela, l'agent doit veiller à ce qu'il n'y ait pas d'ambiguïté de signature, i.e., la définition de la signature négociée, en terme d'ensemble de symbole, ne doit pas être identique à celle d'une signature de L_{n+k}. Cette négociation de signature permet de déterminer la signature visée pour la prochaine tentative d'assistance, i.e., $Z^{(k+1)}$, ainsi que son expression symbolique ; $Z^{(k+1)}$ peut être la même signature (évolution de signature), une nouvelle signature (création de signature) ou une autre signature de L_{n+k}, en cas d'ambiguïté de signature.

Nous désignons par ***Précisions*** l'ensemble des informations apportées par l'utilisateur durant la négociation ;

- la construction d'un nouvel état de langage L_{n+k+1}, à partir de L_{n+k}, en tenant compte des *réactions* et des *précisions* données par l'utilisateur ;

- l'appel au module de RàPET, représenté par  , pour relancer l'élaboration d'un épisode cible avec $Z^{(k+1)}$, de L_{n+k+1}, et effectuer le rapprochement avec des épisodes antérieurs. Il s'en suit la tentative $k+1$ portant sur la signature $Z^{(k+1)}$ et proposant les épisodes retrouvés.

Le processus de négociation de la tentative k, qui porte sur une signature $Z^{(k)}$ dans L_{n+k}, entraîne la définition de la prochaine signature $Z^{(k+1)}$, décrite dans un nouvel état de langage L_{n+k+1}. La signature $Z^{(k+1)}$ sera visée dans la tentative $k+1$ suivante. Cette négociation peut conduire soit 1) à l'évolution d'une même signature, soit 2) au passage d'une signature à une autre, soit 3) à la création d'une nouvelle signature. Pour permettre à l'utilisateur de créer directement une nouvelle signature, nous ajoutons le cas 4) d'une session d'assistance, dont la première tentative porte sur une nouvelle signature. Cette situation relève du scénario Sc_3, car l'utilisateur doit faire appel à une mise à jour pour que ses réactions, décrivant la nouvelle signature, soient prises en compte.

Notre transposition va exploiter ces informations. Pour cela, nous introduisons quatres alternatives, décrites sur la figure 7.2, qui vont pouvoir survenir dans l'enchaînement des tentatives d'une session relevant du scénario Sc_3 :

- **alternative $\mathbf{A_1}$** : la signature $Z_a^{(k)}$, définie dans L_{n+k}, qui est la cible de la tentative k est voit son expression modifiée, mais reste la cible, $Z_a^{(k+1)}$, de la tentative suivante $(k+1)$; sa nouvelle expression est donnée dans L_{n+k}. Cette alternative correspond à la modification

d'une signature ;

- **alternative $\mathbf{A_2}$** : la signature $Z_a^{(k)}$, cible de la tentative k, est remplacée par une autre signature Z_b (de L_{n+k}), dont une nouvelle expression peut être acquise durant la négociation. Cette nouvelle expression s'applique à la définition de Z_b dans L_{n+k+1}. La signature Z_b devient la cible ($Z_b^{(k+1)}$) de la tentative $k+1$ suivante. Cette situation est rendue possible par le processus de négociation détaillé en 7.4. Cette alternative correspond au changement de signatures ;

- **alternative $\mathbf{A_3}$** : en réagissant sur une signature Z_a (définie dans L_{n+k}), cible de la tentative k (on note donc $Z_a^{(k)}$), l'utilisateur peut souhaiter créer un nouveau sens d'assistance. Lors de la négociation, l'agent alter ego crée une nouvelle signature Z_{nouv} avec l'expression symbolique négociée, qui sera l'objet de la tentative $k+1$ suivante, on note cela $Z_{nouv}^{(k+1)}$;

- **alternative $\mathbf{A_4}$** : en sachant dès le début d'une session qu'un sens d'assistance n'est pas connu de l'agent alter ego, l'utilisateur doit pouvoir commencer une session en définissant une nouvelle signature, on note $Z_{nouv}^{(0)}$. Après négociation, les autres alternatives peuvent s'enchaîner.

Les trois premières alternatives peuvent survenir dans un ordre et un nombre quelconque. La quatrième est un cas particulier d'initiation de session d'assistance.

Intuitivement, dans notre transposition des jeux de langage, l'agent alter ego va appliquer ses mécanismes de renforcement spécifiques à chacune de ces alternatives. Il va se servir des informations collecter pour statuer du succès ou de l'échec de l'utilisation d'un élément de langage, dans l'élaboration d'une situation d'assistance.

Exemple

Nous venons d'introduire un vocabulaire nouveau et des notations pour structurer l'assistance. Cet exemple est destiné à récapituler ce vocabulaire sur une session d'assistance à tentatives d'assistance multiples avec négociation, décrite sur la figure 7.3.

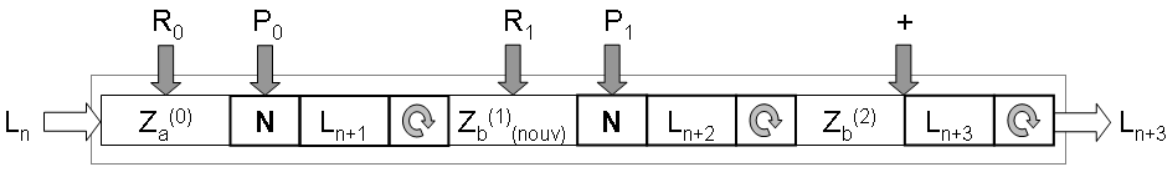


FIG. 7.3 – Exemple de structuration d'une session d'assistance.

Cette session (d'assistance) débute avec l'état de langage L_n. Elle est composée de trois tentatives (d'assistance) : les deux premières sont avec négociation et la dernière est sans négociation. Cette session se termine en fournissant l'état de langage L_{n+3} à l'agent alter ego, pour la suite de ses élaborations.

La première tentative porte sur la signature Z_a de L_n, on note cela $Z_a^{(0)}$. L'agent alter ego enregistre les réactions R_0 de l'utilisateur, et lorsque ce dernier demande la mise à jour des

suggestions d'expérience, l'agent alter ego lui demande d'apporter des précisions P_0. Parmi ces précisions, l'agent alter ego sait qu'il s'agit de l'expression d'un nouveau sens d'assistance par l'utilisateur, il va donc créer une nouvelle signature, Z_b, avec cette expression dans le nouvel état de langage L_{n+1}. Nous sommes en présence de l'alternative d'enchaînement de tentatives A_3, i.e., création de signature en cours de session.

La deuxième tentative porte sur la signature Z_b, on note cela $Z_b^{(1)}$ en insistant bien sur le fait que c'est une nouvelle signature dans L_{n+1}. L'agent alter ego enregistre les réactions R_1 de l'utilisateur, et lorsque ce dernier demande la mise à jour des suggestions d'expérience, l'agent alter ego lui demande d'apporter des précisions P_1. Parmi ces précisions, l'agent alter ego sait qu'il s'agit d'une évolution de l'expression de la signature Z_b ; avec ces réactions et ces précisions, l'agent alter ego va construire un nouvel état de langage L_{n+2} avec cette nouvelle expression de Z_b. Nous sommes en présence de l'alternative d'enchaînement de tentatives A_1, i.e., évolution de signature.

La troisième tentative est sans négociation, elle porte sur Z_b définie dans L_{n+2} ; on note cela $Z_b^{(2)}$. L'utilisateur peut avoir réagi, mais il ne souhaite pas que ses réactions soient prises en compte ; l'utilisateur veut fermer l'interface de suggestion d'expérience. L'agent alter ego lui demande son avis sur la dernière élaboration, celui-ci est favorable. L'agent alter ego construit un nouvel état de langage L_{n+3} avec cet avis. L'utilisateur continue sa tâche sur l'application cible et l'agent alter ego cherche à élaborer des épisodes pour les signatures définies dans L_{n+3}.

7.4 Mécanisme de mise à jour par négociation

Dans les sections précédentes, nous avons déjà largement introduit le besoin et les principes de la négociation du sens des réactions de l'utilisateur pour l'agent alter ego. Afin de décrire complètement le mécanisme de négociation, dans le cadre d'une mise à jour des propositions qui tienne compte des réactions, nous allons présenter les comportements d'assistance et de négociation de l'agent alter ego, du point de vue de la perception qu'en a l'utilisateur. Cette présentation nous permet d'énoncer les hypothèses d'appropriation, sur lesquelles se base notre transposition des jeux de langage.

Considérons la situation générale où l'utilisateur effectue une série d'opérations sur l'application cible, sans qu'il ne nécessite d'assistance. L'agent alter ego construit, en tâche de fond, une trace primitive pour représenter l'observation de ces interactions, i.e., la trace primitive en cours. Lorsque l'utilisateur nécessite une assistance, il fait alors appel à son agent alter ego, qui lui propose les interprétations de ses opérations, faites selon le processus ascendant présenté en 6.6.1, sur la base de l'état de langage L_n en cours.

L'acteur les consulte et il initie une session d'assistance lorsqu'il choisit une interprétation. Une première tentative d'assistance est initiée ; c'est le tronc commun de toute session d'assistance : à ce niveau, les trois scénari introduits dans notre structuration de l'assistance sont encore possibles. Notons que nous ne tenons pas compte, dans notre transposition, d'une simple consultation des interprétations par l'utilisateur lorsqu'elle est suivie de nouvelles opérations sur l'application cible, toutefois cette opération est d'un intérêt certain pour ce dernier.
La figure 7.4 illustre les éléments de structuration dont dispose l'agent alter ego à ce stade, i.e., une interprétation de la trace en cours pour la signature $Z_a^{(0)}$ sélectionnée et des propositions d'épisodes antérieurs. L'utilisateur a accès, via une interface spécifique, à ces éléments, qui lui sont présentés sous forme symbolique.

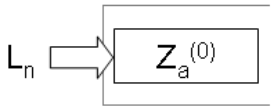


FIG. 7.4 – Première tentative d'assistance : tronc commun aux trois scénari.

C'est dans la suggestion d'épisodes antérieurs pour une interprétation de la situation courante que se situe le but final de l'agent alter ego : il est de donner à l'acteur des indications pour pouvoir poursuivre sa tâche. Ceci est principalement composé :

- d'indications sur les étapes manquantes à son travail en regardant les symboles qui n'ont pas encore été reconnus dans son cas, mais qui le sont dans les épisodes rapprochés ;
- de la désignation de situations réelles, via les épisodes antérieurs rapprochés, qui laisse la place à la facilitation des échanges humains au sein de la communauté, mais qui ne peuvent pas être directement perçus par l'agent alter ego.

Pour permettre cela, le principe de l'interface spécifique de suggestion d'épisodes, que nous proposons, est de permettre à l'acteur d'accéder à l'ensemble des sections des traces manipulées, i.e., en cours ou antérieures, ainsi qu'au détail de chacune de ces sections.

Néanmoins, cette situation idéale d'assistance n'est, par principe, pas toujours possible, en premier lieu dû à l'origine souvent implicite des connaissances. Pour palier à cela, l'acteur peut souhaiter donner à son agent alter ego des informations complémentaires, nous les voyons d'une manière générale comme la désignation, en situation, de ce qui est à prendre en considération pour affiner l'interprétation. Pour permettre cela, nous donnons à l'acteur le moyen d'exprimer, dans une description détaillée de section de trace, ce qui est significatif pour lui : à cet endroit, dans une certaine perspective et dans le contexte de sa tâche en cours. Des éléments d'interface doivent être introduits, dont les modalités d'acquisition sont spécifiques au modèle du domaine d'application. Dans notre cas, il s'agit de pouvoir exprimer la présence d'un mot clé, à un endroit, comme significative en soi ou comme instance d'un concept d'abstraction. L'ensemble de ces indications supplémentaires données par l'utilisateur face aux suggestions d'une tentative est appelé *réactions*.

La bifurcation entre les scenari d'assistance se situe lorsque l'utilisateur souhaite soit :

- mettre fin à la session d'assistance, en fermant l'interface de suggestion d'épisodes pour $Z_a^{(0)}$;
- que l'agent tienne compte de ses réactions et mette à jour ses propositions d'élaboration et d'épisodes antérieurs.

En fonction de cela, la distinction peut être faite entre les trois scénari d'assistance :

- si l'utilisateur souhaitait simplement connaître le détail de l'interprétation de la trace en cours et les épisodes rapprochés, sans influer sur l'interprétation de l'agent alter ego, nous considérons qu'il annule la première tentative, c.f., scénario Sc_1. Ceci n'influe pas l'état de langage L_n, qui continue à être utilisé ;

– si l'utilisateur désire arrêter la consultation de l'expérience rapprochée et exprimer un avis favorable ou défavorable sur l'interprétation effectuée, nous nous situons dans le cas du scenario Sc_2. Un nouvel état de langage L_{n+1} est construit et devient l'état en cours pour de nouvelles élaborations en tâche de fond. La figure 7.5 illustre les éléments de structure de session manipulés par l'agent alter ego dans ces deux premières situations ;

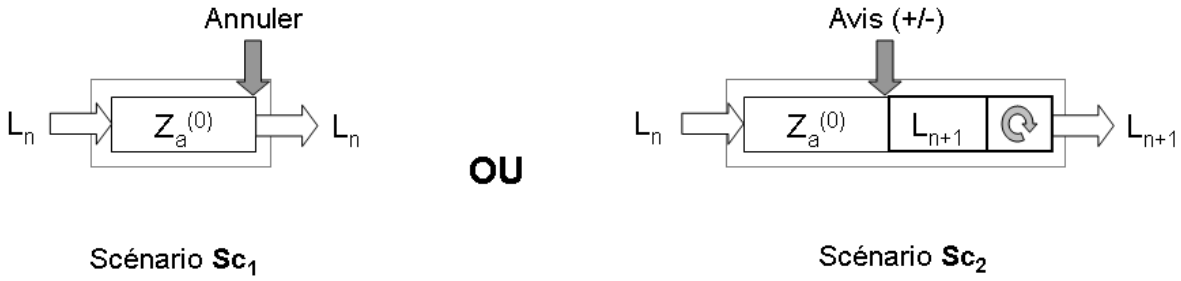


FIG. 7.5 – Fin de session à tentative unique sans négociation : scénari Sc_1 et Sc_2.

– si l'utilisateur veut que l'agent alter ego mette à jour ses propositions en tenant compte de ses réactions, nous sommes dans le scenario Sc_3. Chaque demande de mise à jour entraîne un appel au mécanisme de négociation.

Dans le cas du scénario Sc_3, après une série de réactions vis à vis des éléments symboliques, l'acteur demande à son agent alter ego de mettre à jour ses suggestions en tenant compte de ses indications.

Pour cela, l'agent alter ego doit tout d'abord construire un motif représentant les indications portées sur une section. Cette transcription passe par la construction de trois patterns reportant les indications données dans les trois observations d'une section (de forme $E \cdot T \cdot E$) ; l'assemblage de ces trois patterns constitue un motif (de forme $E \cdot T \cdot E$). La construction d'un pattern réplique la structure d'arbre de containers de l'observation associée. Les captures d'OI désignées par l'utilisateur comme significatives à un endroit dans la structure de l'observation sont représenté par des captures d'OI au même endroit dans le pattern ; ces captures sont : 1) soit constantes, si l'utilisateur estime l'OI significatif en soi 2) soit variables, si l'utilisateur estime que l'OI est significatif en tant qu'instance d'un concept généralisant, selon le modèle de domaine.

Lors de la spécification d'éléments significatifs dans une description de section, l'acteur peut être amené :

1. soit à manipuler une section de trace (en cours ou antérieure) sur laquelle un symbole a été apparié. Il en résulte une ambiguïté pour l'agent alter ego, car il peut s'agir de l'*évolution* du symbole, i.e., l'affinement de son sens, ou de la *création* d'un nouveau symbole, i.e., l'expression d'un sens inexistant.
2. soit à manipuler une section de trace sur laquelle aucun symbole n'a été reconnu. Dans ce cas, il ne peut s'agir a priori que d'une *création* d'un nouveau symbole.

Dans les deux cas, les possibilités d'interaction qui sont données à l'acteur peuvent conduire à des motifs déjà utilisés dans l'état de langage en cours pour exprimer un symbole ; cela est source d'*ambiguïté de symbole* pour l'agent alter ego.

Lors de la négociation d'un élément symbolique, appelée *négociation de symbole*, l'agent alter ego propose les symboles existants qui ont des motifs similaires au motif négocié. Dans tous les

cas, l'utilisateur peut choisir de plutôt utiliser un de ces symboles à la place du motif négocié ; nous traiterons cela comme un *remplacement* de symbole. Si le motif traité est ambigu, i.e., identique au motif d'un symbole existant, l'utilisateur a le choix de modifier le motif pour lever l'ambiguïté, ou de convenir d'utiliser le symbole existant ; il n'a pas la possibilité de créer un symbole ambigu, il est donc contraint au remplacement par le symbole existant.

L'appropriation que nous comptons de l'utilisateur lors de la négociation de symbole est de pouvoir statuer correctement du rôle que tiennent ses indications sur la section, i.e., de décider correctement s'il s'agit de la création d'un nouveau sens symbolique, de l'évolution d'un sens existant ou du remplacement par un sens existant.

L'étape suivante dans le mécanisme de mise à jour consiste à convenir d'une nouvelle définition symbolique pour la situation d'assistance visée, on l'appelle *négociation de signature*. Pour cela, l'utilisateur a les moyen d'ajouter ou d'enlever des symboles dans la définition. En interaction avec les modifications de la définition symbolique par l'utilisateur, l'agent alter ego cherche des signatures qui ont des définitions similaires voire identiques. En effet, ces possibilités d'interactions peuvent conduire à une expression symbolique déjà existante pour une autre signature, cela constitue une *ambiguïté de signature* pour l'agent alter ego. Notre modèle d'état de langage ne permet pas d'avoir deux définitions symboliques de signature identiques, sans quoi ces signatures sont *fusionnées*, i.e., elles ne forment plus qu'une seule signature avec comme pondérations de signature et de relations symbole-signature les pondérations maximales de part et d'autre.

Dans la même logique que pour la négociation de symbole, la négociation de signature laisse à l'utilisateur la possibilité de :

- faire évoluer un sens d'assistance existant. Cela est considéré comme la modification d'une signature existante dans l'état en cours. La tentative qui va suivre portera sur cette même signature, nous sommes dans l'alternative d'enchaînement de tentatives A_1 ;
- créer un nouveau sens d'assistance, qui sera traité par l'ajout d'une nouvelle signature dans l'état de langage suivant. Cette signature deviendra l'objet de la prochaine tentative, c.f., alternative A_3 ;
- remplacer la signature visée par une signature similaire proposée. La tentative qui va suivre portera sur cette autre signature, nous sommes dans l'alternative d'enchaînement de tentatives A_2. Si la définition symbolique donnée par l'utilisateur est identique à celle d'une autre signature, l'agent alter ego traite cela comme la convergence de deux sens d'assistance et effectue la fusion des signatures associées. Si l'utilisateur ne souhaite pas cela il doit modifier sa définition symbolique afin de lever l'ambiguïté.

Après la négociation de signature, l'agent alter ego construit un nouvel état de langage. La section 7.5 détaille les principes de notre transposition et donne l'algorithme de construction d'un nouvel état de langage à partir des réactions et précisions de l'utilisateur. Un appel au module de RàPET pour une élaboration de la situation courante avec la nouvelle signature cible, définie dans le nouvel état de langage, et le rapprochement avec des épisodes antérieurs. Le résultat est soumis à l'utilisateur via l'interface de proposition d'expérience.

La figure 7.6 résume les principes de ce mécanisme de mise à jour par négociation ; les étapes encadrées sont faites en interaction avec l'acteur, à l'inverse de celles encerclées qui sont effectuées seulement par l'agent alter ego. Des interfaces spécifiques sont nécessaires pour permettre la négociation avec l'utilisateur d'un symbole et d'une signature.

Le mécanisme de mise à jour des propositions par négociation est dirigée par l'agent alter ego. Ce dernier s'attache successivement à :

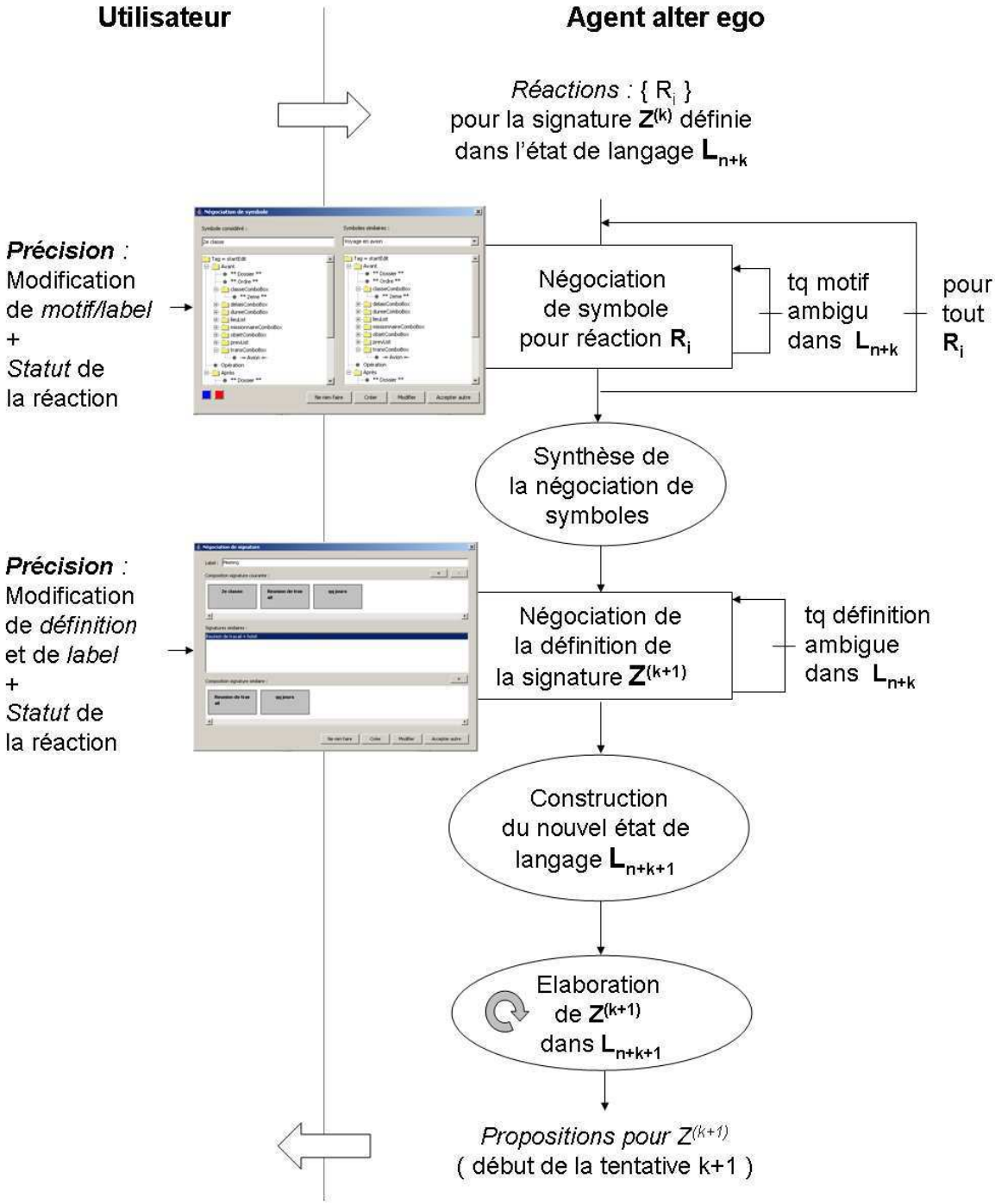


FIG. 7.6 – Mécanisme de négociation appliqué entre deux *tentatives d'assistance*

1. lever les ambiguïtés issues des réactions de l'utilisateur dans les descriptions des sections de traces. A chaque négociation de symbole l'utilisateur statue du sens de son intervention sur une section, en la considérant soit comme :
   - la *création* d'un symbole, qui sera ajouté à l'état suivant ;
   - l'*évolution* du symbole de l'état de langage en cours apparié à la section ;
   - le *remplacement* par un symbole existant ;

2. synthétiser la négociation de symbole afin de construire une nouvelle définition symbolique de la signature, en tenant compte de l'évolution, la création, le remplacement de symboles vis à vis de la définition précédente de la signature ;

3. convenir d'une expression symbolique du sens d'assistance visé ; lors de la négociation de signature. L'utilisateur statue du sens de son intervention sur une section, en la considérant soit comme :
   - l'*évolution* du sens d'assistance déjà considéré ;
   - la *création* d'un nouveau sens d'assistance, qui sera ajouté à l'état suivant ;
   - le *remplacement* du sens d'assistance visé par un autre sens, en traitant le cas d'égalité de définition symbolique par une fusion des signatures correspondantes ;

4. transcrire les informations issues des négociation en itérations de jeux de langage, afin d'obtenir un nouvel état de langage. Les principes de cette transcription sont détaillés en 7.5 ;

5. relancer une nouvelle interprétation, sur la base de ce nouvel état de langage, en appelant le module de RàPET.

La proposition à l'utilisateur de la nouvelle interprétation de la situation courante et les épisodes antérieurs rapprochés, avec la nouvelle signature cible, initie une nouvelle *tentative d'assistance*. L'utilisateur peut continuer à interagir avec les propositions faites par l'agent alter ego, cela se traduit par une succession de tentatives d'assistance lors des demandes de mise à jour : à chaque fois un nouvel état de langage est généré. A tout moment (en dehors de la mise à jour), l'utilisateur peut mettre fin à la session d'assistance ; l'agent alter ego lui demande son avis sur la dernière interprétation et il termine la session selon les principes du scénario Sc_2, en construisant un nouvel état de langage. La figure 7.7 rappelle les éléments de structuration qui interviennent *a minima* dans une session à tentatives multiples ; en effet, nous pouvons avoir $N = 1$, i.e., une session à deux tentatives.

Pour être complet sur l'utilisation, nous devons donner à l'utilisateur le moyen d'initier une session d'assistance en créant directement un nouveau sens d'assistance. Pour que ces réactions soient prises en compte, il doit nécessairement faire appel à une mise à jour. Pour traiter ce cas particulier de session à tentatives multiples (scénario Sc_3), nous avons introduit l'alternative d'enchaînement A_4.

Algorithme de négociation

Nous introduisons la structure de donnée appelée *réaction* R, pour représenter l'ensemble des indications données par l'utilisateur dans la description d'une *section*, éventuellement appariée à

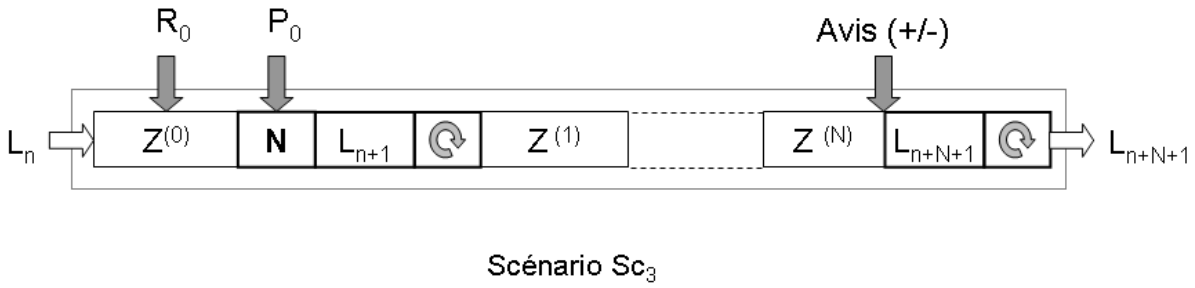


FIG. 7.7 – Session à tentatives multiples avec négociation : scénario Sc_3.

un *symbole* (*null* sinon) ; le *motif* est la transcription de ces indications, et *label* est l'étiquette en langage naturel convenue par l'utilisateur.

$$R = (section, symbole, motif, label)$$

Le résultat de la négociation de symbole pour la réaction R_i est représenté par la structure de donnée P_i, appelée *précision*. Elle est composée de la réaction, du statut (*Creation*, *Evolution*, *Remplacement* de symbole) que l'utilisateur à donné à son *action* et du *symbole* résultant défini avec $R_i.motif$ et $R_i.label$, qui ont éventuellement été modifiés durant la négociation.

$$P_i = (reaction\ R_i, action, symbole)$$

Le résultat de la négociation de signature est représenté par la structure de donnée $P_{Z^{(k)}}$, appelée *précision* pour la signature $Z^{(k)}$. Elle est composée de la nouvelle *définition* symbolique négociée, du statut (*Creation*, *Evolution*, *Remplacement* de signature) que l'utilisateur à donné à son *action*, et de la signature $Z^{(k+1)}$ résultante.

$$P_{Z^{(k)}} = (definition, action, Z^{(k+1)})$$

L'algorithme de mise à jour des propositions est donné par la fonction NÉGOCIATION, qui est décrite sur la figure 7.8. Cette fonction suit les étapes décrites schématiquement sur la figure 7.6. L'ensemble des *Précisions* de symbole $\{P_i\}$ est construit en appelant la fonction NEGOCIATIONSYMBOLEGUI pour chaque réaction R_i. La fonction SYNTHÈSE, décrite sur la figure 7.9, construit une définition symbolique intermédiaire DefInterm à partit des précisions, qui est la première définition considérée dans la négociation de signature. Cette négociation est effectuée en appelant la fonction NEGOCIATIONSIGNATUREGUI, qui renvoie la précision de signature $P_{Z^{(k)}}$. Un nouvel état de langage L_{n+k+1} est construit à l'aide de la fonction CONSTRUCTIONETAT-LANGAGE, définie dans la section suivante. L'appel au module de RàPET pour une nouvelle interprétation de $Z^{(k+1)}$ est effectué par la méthode RELANCERELABORATION. Le résultat de l'élaboration et de la recherche d'épisode est soumis à l'utilisateur via l'interface de suggestion d'expérience.

La fonction SYNTHÈSE, c.f., figure 7.9, a pour principe d'inclure tous les symboles issus des négociations de symbole dans la définition symbolique intermédiaire DefInterm renvoyée. S'il s'agit d'une signature existante, les symboles présents dans la définition initiale et non modifiés sont ajoutés ; dans le cas particulier d'une nouvelle signature dès la première tentative, la définition intermédiaire est limitée aux symboles négociés.

Fonction Négociation
  entrée : réactions $\{R_i\}$, état de langage L_{n+k}, signature $Z^{(k)}$
  sortie : une élaboration de la situation courante avec $Z^{(k+1)}$ de L_{n+k+1} et les épisodes rapprochés
début
  $Précisions \leftarrow \emptyset$
  pour toute réaction R_i
    $Précisions \leftarrow Précisions \cup$ NegociationSymboleGUI(R_i, L_{n+k})
  fin-pour
  DefInterm $\leftarrow$ Synthèse$(Précisions, Z^{(k)}, L_{n+k})$
  $P_{Z^{(k)}} \leftarrow$ NégociationSignatureGUI(DefInterm, $Z^{(k)}, L_{n+k}$)
  $L_{n+k+1} \leftarrow$ ConstructionEtatLangage$(Précisions, P_{Z^{(k)}}, L_{n+k})$
  RelancerElaboration$(Z^{(k+1)}, L_{n+k+1})$
  retourne rafraîchissement élaboration/recherche pour $Z^{(k+1)} \in L_{n+k+1}$
fin

Fig. 7.8 – Algorithme de rafraîchissement des propositions par négociation

Fonction Synthèse
  entrée : précisions $\{P_i\}$, état de langage L_{n+k}, signature $Z^{(k)}$
  sortie : un ensemble de symboles, définition intermédiaire de $Z^{(k+1)}$
début
  si $k = 0$ ET $Z^{(0)} = nouv.sign.$ **alors**
    DefInterm $\leftarrow \emptyset$
    pour toute précision P_i
      DefInterm $\leftarrow$ DefInterm $\cup$ $P_i.symbole$
    fin-pour
    retourne DefInterm
  sinon
    DefInterm $\leftarrow$ Symboles$(Z^{(k)}, L_{n+k})$
    pour toute précision P_i
      DefInterm $\leftarrow$ DefInterm $\cup$ $P_i.symbole$
      si $P_i.action = Evolution$ OU $P_i.action = Remplacement$ **alors**
        DefInterm $\leftarrow$ DefInterm $-P_i.reaction.symbole$
      fin-si
    fin-pour
    retourne DefInterm
  fin-si
fin

Fig. 7.9 – Algorithme de synthèse de la négociation des symboles pour construire une définition symbolique intermédiaire

7.5 Transposition des jeux de langage

Pour expliquer les principes de notre transposition des travaux de [Steels 00], nous allons présenter les jeux de langage que nous considérons en les discutant selon les principes fondamentaux d'émergence de langage que sont : l'*apprentissage par renforcement*, l'*auto-organisation*, le *sélectionisme* et le *couplage structurel*.

Tout d'abord, parmi les trois mécanismes permettant l'émergence d'un langage lexical selon la synthèse de [Steels 00], le **jeu de discrimination** est effectué pour obtenir une conceptualisation de l'environnement. Il est réalisé individuellement par chaque agent, qui sont des robots dans leur domaine d'application. Dans notre cas, nous allons dans un premier temps faire l'hypothèse simplificatrice selon laquelle les signatures sont définies uniquement par l'utilisateur, et qu'elles sont partagées avec son agent alter ego. Cette hypothèse simplificatrice enlève toute proactivité à l'agent alter ego pour la tâche d'identification. Nous faisons cette simplification, qui augmente grandement les possibilités d'émergence d'un langage, car nous considérons dans un premier temps que seulement l'utilisateur est en mesure de faire cette discrimination. Pour ces raisons, aucun jeu de discrimination n'a été introduit dans notre première approche, et nous traiterons en perspectives des possibilités et de l'intérêt de laisser une proactivité à l'agent alter ego dans cette tâche de discrimination.

Le principe d'*apprentissage par renforcement* est tout de même présent en traitant les signatures selon qu'elles sont maintenues ou non par l'acteur. Le principe de sélectionisme est constitué de l'élimination des signatures avec des pondérations faibles, mais aussi des limites d'expressivité évidentes que constitue le fait de ne pouvoir exprimer une signature que par un ensemble non ordonné de symboles. Durant la définition de la signature, l'acteur peut obtenir une signature déjà existante : les signatures sont fusionnées après que l'acteur ait choisi un label commun. De nouvelles signatures peuvent être créées, leur pondération, ainsi que celles de leurs relations sont fixées à des valeurs initiales arbitraires.

Une différence fondamentale avec les jeux de langage présentés par [Steels 00], est que, dans notre cas, nous ne pouvons pas exactement parler de paire symbole-signature mais plutôt d'une relation de cardinalité $1..n$, puisqu'une signature est exprimée par un ensemble de symboles. Nous insistons sur le fait qu'il ne s'agit pas d'une tentative de discrimination ; dans notre première approche, nous proposons plutôt de voir que l'identification d'un certain nombre de ces symboles (au delà du seuil τ_{idSign} introduit en 6.6.1) permet à l'agent alter ego de désigner un sens (la sémantique d'une signature) à l'aide de symboles reconnus, et de faire une action (proposer des rapprochements avec des épisodes antérieures). La pertinence de cette élaboration est exprimée de façon directe ou non par l'utilisateur, en émettant un avis (scenario Sc_2) ou en demandant la mise à jour par négociation (scenario Sc_2) ; dans les deux cas cela permet à l'agent alter ego de statuer du succès/échec d'un acte de communication (une tentative d'assistance).

Dans la suite nous allons nous focaliser sur le scenario Sc_3 ; les traitements pour le scénario Sc_2, constituent le cas simplifié de ce que nous allons détailler et conduit renforcement/inhibition de la signature, des symboles et des relation symbole-signature selon l'avis favorable/défavorable de l'utilisateur. Pour les jeux d'imitation et de nommage, nous adoptons l'approche suivante pour juger du succès ou de l'échec de la $k^{ième}$ tentative d'assistance, portant sur $Z^{(k)}$ et dont la négociation est connue par $P_{Z^{(k)}}$:

1. une itération de **jeu d'imitation** est initiée pour chaque symbole de $Z^{(k)}$ et chaque sym-

bole de $Z^{(k+1)}$;

2. une itération de **jeu de nommage** est initiée pour chaque relation symbole-signature de $Z^{(k)}$ et chaque relation symbole-signature de $Z^{(k+1)}$.

De manière générale, nous évaluons par un succès toute itération portant sur un élément de langage qui est "accepté" par l'utilisateur, et par un échec toute itération portant sur un élément de langage qui est "rejeté" par l'utilisateur. Le succès ou l'échec d'une itération entraîne directement le renforcement, respectivement l'inhibition, de l'élément de langage impliqué ; un renforcement est donc synonyme de succès d'une itération, et une inhibition (ou une suppression directe) d'échec d'une itération.

Notre construction d'un nouvel état de langage vient répercuter le succès ou l'échec de ces itérations de jeux ; nous la détaillons suivant les principes fondamentaux d'émergence de langage qu'elle doit mettre en œuvre, en y intégrant les traitements sur les signatures :

1. Les stratégies d'*apprentissage par renforcement* sont les suivantes, selon les alternatives d'enchaînement de tentatives :

   - pour l'**alternative $\mathbf{A_1}$** : la signature cible de l'assistance à la tentative k l'est toujours à la tentative $k+1$ ($Z_a^{(k)} = Z_a^{(k+1)}$ ET $Z_a^{(k)} \in L_{n+k}$). La signature Z_a est renforcée. Globalement, nous évaluons par un succès les itérations portant sur les éléments (symboles et relations) acceptées et par un échec celles portant sur les éléments rejetés.

     La nouvelle définition symbolique $P_{Z^{(k)}}.definition$ guide cette opération de renforcement. Parmi les symboles présents dans la nouvelle définition symbolique, nous avons soit :
     - des actions de création de symboles ; ces nouveaux symboles sont ajoutés au nouvel état de langage avec une relation symbole-signature, tous eux avec des pondérations initiales arbitraires ;
     - des actions d'évolution de symboles ; un nouveau symbole est créé à chaque fois et une relation symbole-signature, qui sont initialisés en "héritant" des valeurs de pondération de leurs éléments d'origine dans L_{n+k}, puis renforcés ;
     - des actions de remplacement de symboles ; à chaque fois, le symbole nouvellement choisi est renforcé et une relation avec la signature est créée, sa pondération "hérite" de la pondération de la relation d'origine.
     - soit des symboles non issus des réactions, donc déjà présents dans L_{n+k}. Ces symboles sont renforcés, et leurs relations sont 1) soit renforcées si présentes dans L_{n+k} (i.e., le symbole fait déjà partie de l'expression de $Z_a^{(k)}$), 2) soit créées avec une pondération initiale arbitraire si inexistante dans L_{n+k} (i.e., nouvellement ajoutés durant la négociation de signature).

     Enfin, les symboles de $Z_a^{(k)}$, qui ne sont plus dans la définition symbolique de $Z_a^{(k+1)}$, sont inhibés et leurs relations sont supprimées. La suppression des relations rejetées, au lieu de leur inhibition, est nécessaire pour permettre à l'utilisateur d'imposer une définition à l'agent alter ego. Cette approche est extrême, nous discuterons en 9.4.2 de son évolution, lors de notre premier retour d'expérience ;

Nous avons vu que lors de la spécialisation d'un symbole par l'utilisateur, l'agent alter ego effectue en réalité la création d'un nouveau symbole avec cette nouvelle définition et il attribut à ce nouveau symbole la pondération du symbole d'origine après incrémentation. Cette opération de "duplication" est nécessaire car un symbole peut être partagé par plusieurs signatures et l'évolution de son sens pour une signature n'est a priori pas valable pour les autre ; une évolution de symbole est traitée de façon identique à la création d'un nouveau symbole, à la différente qu'un symbole issu d'une évolution "hérite" du renforcement déjà acquis par son symbole d'origine.
L'introduction de ce mécanisme d'*héritage* des pondérations est faite intuitivement pour qu'une évolution de symbole bénéficie de l'apprentissage par renforcement déjà acquis pour son symbole d'origine, de même pour la relation. Nous avons également utilisé ce mécanisme d'héritage lors du remplacement de symbole ; nous faisons l'hypothèse que l'opération de remplacement joue le rôle d'un affinement de sens, tout comme l'évolution de symbole.

- pour l'**alternative $\mathbf{A_2}$** : la signature cible de l'assistance à la tentative k est remplacée, à la tentative $k+1$, par une autre signature déjà existante ($Z_a^{(k)} \neq Z_b^{(k+1)}$ ET $Z_b^{(k+1)} \in L_{n+k}$). La signature Z_b est renforcée et Z_a est inhibée. Globalement, nous évaluons par un succès les itérations portant sur les éléments (symboles et relations) acceptées dans $Z_b^{(k+1)}$, et par un échec celles portant sur les éléments de $Z_a^{(k)}$ qui ne sont pas communs avec $Z_b^{(k+1)}$.

  La nouvelle définition symbolique $P_{Z^{(k)}}.definition$ de $Z_b^{(k+1)}$ guide cette opération de renforcement. Parmi les symboles présents dans cette nouvelle définition symbolique, nous avons ceux issus des réactions négociées, qui correspondent soit :
  - à des actions de création de symboles ; ces nouveaux symboles sont ajoutés au nouvel état de langage avec une relation symbole-signature, tous deux avec des pondérations initiales arbitraires ;
  - à des actions d'évolutions de symboles ; un nouveau symbole est créé et une relation symbole-signature, qui sont initialisés en "héritant" des valeurs de pondération de leurs éléments d'origine dans L_{n+k} pour $Z_a^{(k)}$, puis renforcés ;
  - à des remplacements de symboles ; le symbole nouvellement choisi est renforcé et une relation avec la signature est créée, sa pondération "hérite" de la pondération de la relation d'origine dans L_{n+k} pour $Z_a^{(k)}$.

  Pour les symboles de $P_{Z^{(k)}}.definition$ non issus des réactions, donc déjà présents dans L_{n+k} : ils sont renforcés, et leurs relations avec Z_b sont 1) soit renforcées si présentes dans L_{n+k} (i.e., le symbole fait déjà partie de l'expression de Z_b), 2) soit créées avec une pondération initiale arbitraire si inexistante dans L_{n+k} (i.e., nouvellement ajoutés durant la négociation de signature).
  Puis, les relations de Z_a avec ses symboles sont inhibées, ainsi que tous les symboles de la définition de Z_a qui ne sont pas présents dans $P_{Z^{(k)}}.definition$.

  Enfin, dans cette alternative, nous avons à traiter le cas d'ambiguïté de signature, i.e., la définition symbolique négociée est identique à celle d'une signature de L_{n+k}. Nous rappelons que cela n'est pas permis dans notre modèle de langage et les signatures sont fusionnées, i.e., elles ne forment plus qu'une seule signature dont la pondération, ainsi que celle de ses relations avec ses symboles, héritent du maximum de pondération des

deux signatures fusionnées ou des relations respectives. La signature résultante devient cible de la tentative suivante.

- pour l'**alternative $\mathbf{A_3}$** : la signature cible de l'assistance à la tentative k est remplacée, à la tentative $k+1$, par une nouvelle signature ($Z_a^{(k)} \neq Z_b^{(k+1)}$ ET $Z_b^{(k+1)} \ni L_{n+k}$). La signature Z_b est créée avec une pondération initiale arbitraire et Z_a est inhibée. Globalement, nous évaluons par un succès les itérations portant sur les éléments (symboles et relations) acceptées dans $Z_b^{(k+1)}$, et par un échec celles portant sur les éléments de $Z_a^{(k)}$ qui ne sont pas communs avec $Z_b^{(k+1)}$. Le traitement est proche de celui de l'alternative A_2, en ne gardant que l'héritage de pondération pour l'évolution d'un symbole, et sans fusion.

  La définition symbolique $P_{Z^{(k)}}.definition$ de la nouvelle signature $Z_b^{(k+1)}$ guide cette opération de renforcement. Parmi les symboles présents dans cette définition symbolique, nous avons ceux issus des réactions négociées :
  - soit des actions de création de symboles, ces nouveaux symboles sont ajoutés au nouvel état de langage avec une relation symbole-signature, tous deux avec des pondérations initiales arbitraires ;
  - soit des évolutions de symboles, un nouveau symbole est créé en "héritant" de la pondération du symbole d'origine dans L_{n+k} ; une relation symbole-signature est créée avec une pondération initiale arbitraire ;
  - soit des remplacements de symboles, le symbole nouvellement choisi est renforcé et une relation avec la signature est créée avec une pondération initiale arbitraire.

  Pour les symboles de $P_{Z^{(k)}}.definition$ non issus des réactions, donc déjà présents dans L_{n+k} : ils sont renforcés, et leurs relations avec $Z_b^{(k+1)}$ sont créées avec une pondération initiale arbitraire.
  Enfin, les relations de Z_a avec ses symboles sont inhibées, ainsi que tous les symboles de la définition de Z_a qui ne sont pas présents dans $P_{Z^{(k)}}.definition$.

- pour l'**alternative $\mathbf{A_4}$** : l'utilisateur a commencé la session d'assistance en créant une nouvelle signature, qui est ajoutée va être ajoutée à l'état de langage suivant, car nous la supposons dans un premier temps non ambiguë après négociation. La signature Z_b est créée avec une pondération initiale arbitraire. Globalement, nous évaluons par un succès les itérations portant sur les éléments (symboles et relations) acceptées dans $Z_b^{(k+1)}$. Le traitement est proche de celui de l'alternative A_3, en excluant tous les traitements sur la signature rejetée, par définition inexistante à la première tentative.

  La définition symbolique $P_{Z^{(k)}}.definition$ de la nouvelle signature $Z_b^{(k+1)}$ guide cette opération de renforcement. Parmi les symboles présents dans cette définition symbolique, nous avons ceux issus des réactions négociées :
  - soit des actions de création de symboles, ces nouveaux symboles sont ajoutés au nouvel état de langage avec une relation symbole-signature, tous deux avec des pondérations initiales arbitraires ;
  - soit des évolutions de symboles, un nouveau symbole est créé en "héritant" de la pondération du symbole d'origine dans L_{n+k} ; une relation symbole-signature est créée avec une pondération initiale arbitraire ;
  - soit des remplacements de symboles, le symbole nouvellement choisi est renforcé et

une relation avec la signature est créée avec une pondération initiale arbitraire.
Pour les symboles de $P_{Z^{(k)}}.definition$ non issus des réactions, donc déjà présents dans L_{n+k} : ils sont renforcés, et leurs relations avec $Z_b^{(k+1)}$ sont créées avec une pondération initiale arbitraire.

Pour cette alternative, en cas d'ambiguïté de signature lors de la première négociation, nous revenons au cas de l'alternative A_2, avec un traitement particulier excluant toutes les opérations portant $Z_a^{(k)}$ et ses relations, car inexistants par définition dans L_{n+k}.

Notons que les jeux d'imitation et de nommage sont très semblables dans la mesure où les symboles et les relations voient leurs pondérations renforcées ou d'inhibées dans les mêmes conditions, aux mécanismes d'héritage près. Leur couplage se situe à ce niveau. Cependant, les symboles peuvent être partagés entre plusieurs signatures, ce qui laisse un espace pour l'évolution "indépendante" des répertoires de symboles et de relations.

La pondération des symboles nouvellement créés est fixée à une valeur initiale arbitraire, ce qui influe sur la durée minimale durant laquelle un symbole est mémorisé par l'agent alter ego.

Nous appliquons des renforcements ou des inhibitions avec des valeurs constantes pour les symboles, la signature et les relations symbole-signature. Pour ces dernières, il serait envisageable de nuancer ces variations en fonction du caractère discriminant d'un symbole, selon des principes analogues à la mesure TFIDF (c.f., 10.1.1 pour une définition) ; ainsi, un symbole peu présent dans les définitions de signatures serait plus renforcé/inhibé qu'un symbole fréquemment présent.

2. Le *sélectionisme* s'exprime :
   (a) pour le jeu d'imitation, par la contrainte sur le poids des symboles, qui ont tendance à survivre dans le répertoire lorsque leur poids est élevé. Les variations viennent des actions de l'utilisateur qui sont inconnues de l'assistant, et où des objets d'intérêts ont été ajoutés au motif d'un symbole ;
   (b) pour le jeu de nommage, par les limites de l'agent alter ego : si une situation ne peut pas être correctement reconnue, les signatures et les symboles introduits pour capturer sa description ont moins de chance de survivre.

Outre ces limites de perception et de modalité d'expression d'un sens au niveau symbolique, tout comme celles évoquées pour une signature, une **pression sélective** est appliquée en faisant successivement, lors de la création d'un nouvel état de langage :

(a) la suppression des relations avec des pondérations nulles, pour le jeu de nommage ;
(b) la suppression des signatures à pondération nulle, si elles ne sont reliées à aucun symbole ;
(c) la suppression des symboles à pondération nulle, pour le jeu d'imitation, si ils ne sont reliés à aucune signature.

Tel quel, ce mécanisme de pression sélective peut former des ambiguïtés de signature ; pour éviter cela, nous permettons la suppression des relations d'une signature (donc sa définition symbolique) que si elle ne forme pas d'ambiguïté de signature. En ajoutant cette restriction, il est possible d'avoir des symboles et des relations dont les pondérations sont nulles,

nous parlerons de symboles et de relations *zombis*. Ce cas doit être pris en compte dans nos mécanismes de RàPET, nous reviendrons sur cela ci-dessous lors de la présentation de la fonction $\mathcal{P}(.,.)$.

Dans le cas particulier où un symbole est utilisé par une seule signature, il serait possible d'effectuer la suppression du symbole d'origine après son évolution, afin de limiter l'explosion de la taille du répertoire de symboles ; cette situation représente le cas où la "duplication" lors de l'évolution de symbole n'est pas nécessaire. Toutefois, cette stratégie est extrême, elle peut être remplacée par un mécanisme d'oubli des symboles inutilisés (par l'inhibition de leur pondération à chaque création d'un état de langage) afin de favoriser le maintient d'une population de symboles qui peuvent s'avérer utiles par la suite.

3. L'*auto-organisation* est la résultante des boucles de rétroaction positive placées :
   (a) pour le jeu d'imitation, entre l'exactitude de la reconnaissance d'un symbole (ie., un appariement porteur de sens sur une section de trace) et son utilité pour l'acteur. Ce dernier donne explicitement son avis sur l'utilité de la dernière définition de signature, donc des symboles qu'elle utilise, en fin de session d'assistance ; l'utilité est exprimée implicitement entre deux tentatives consécutives d'assistance, via l'écart entre les deux définitions symboliques d'une signature ;
   (b) pour le jeu de nommage, entre l'utilisation d'un symbole pour désigner une signature et le succès (exprimé par l'utilisateur selon la logique détaillée ci-dessus) de cette signature dans une tentative d'assistance. Quand un symbole est accepté ou refusé, la relation symbole-signature associée est renforcée ou inhibée.

Selon les principes de [Steels 00], la retroaction positive influe :
- pour les symboles, en l'occurrence des voyelles, sur la réutilisation d'une voyelle d'autant plus qu'elle a été utilisée avec succès, i.e., correctement émise, perçue, rémise et reperçue ;
- pour les relations symboles-sens, l'utilisation d'un symbole pour exprimer un sens est effectuée en choisissant la relation dont la pondération est la plus forte. Le même mécanisme est appliqué en sens inverse par l'autre agent afin d'identifier le sens visé dans la tentative de communication à partir du symbole perçu. Le succès réside dans l'"alignement" des mapping symbole-sens respectifs ; un "alignement" est d'autant plus renforcé et donc réutilisé qu'il permet une communication avec succès.

La rétroaction positive entre succès et réutilisation se fait, dans notre cas :
- pour les symboles, quelle que soit la signature : en considérant que la reconnaissance d'un symbole (sa perception) et son acceptation par l'utilisateur dans l'expression d'une signature doivent influer sur la "confiance" de l'agent alter ego lorsqu'il va réutiliser ce symbole ;
- pour les relations symbole-signature : la sémantique des multiples relations pour une signature, i.e., dans notre cas, l'expression d'un sens par un ensemble de symbole, est différente de celle de [Steels 00] qui place ces relations en compétition. Nous considérons que l'utilisation d'un symbole (durant l'interprétation d'une signature) et son acceptation par l'utilisateur dans l'expression de la signature doivent influer sur la "confiance" de l'agent alter ego lorsqu'il va réutiliser ce symbole dans une prochaine interprétation de la même signature ;

Nous appliquons cela aux mécanismes d'identification de signature, via le calcul du taux d'identification (équation 6.8) et de rapprochement d'épisode sources, via la mesure de similarité entre épisodes (équation 6.10), en introduisant la fonction $\mathcal{P}(.,.)$; elle est basée sur les pondérations de symbole et de relation, qui constituent des mesures de "confiance". Ceci est fait pour 1) influer sur le déclenchement d'une recherche d'épisode, et 2) influer sur la mesure de similarité, afin de proposer en premier à l'utilisateur les épisodes contenant des symboles "de confiance" dont l'utilisation pour exprimer une signature est elle même faite "avec confiance".

Une première forme pour la fonction $\mathcal{P}(\sigma, Z)$, qui doit être croissante selon la pondération ρ_σ et la pondération de la relation entre σ et Z, soit $\rho_{\sigma,Z}$, est :

$$\mathcal{P}(\sigma, Z) = \rho_\sigma \cdot \rho_{\sigma,Z} \tag{7.1}$$

Pour un symbole σ *zombi*, que nous avons introduit dans le cadre de la pression sélective, il conviendrait de remplacer ρ_σ $(= 0)$ par une valeur arbitraire $\rho_{symboleZombi} > 0$, afin que $\mathcal{P}$ ne soit pas nulle ; et en faisant de même, pour la même raison, avec une relation zombie.

Compte tenu que les pondérations ne sont pas bornées, elles peuvent atteindre des valeurs importantes. Les valeurs de $\mathcal{P}$ présenteraient alors de grandes variations entre des symboles admis depuis longtemps (de forte pondération) et ceux nouvellement créés (de pondération faible) ; il en est de même pour leur utilisation dans l'expression d'une signature. Afin d'atténuer ces écarts, il peut être intéressant d'appliquer une transformation logarithmique à $\mathcal{P}$, dont une forme est donnée ci-dessous. Les traitements d'exception pour les symboles et les relations zombies, que nous venons de présenter, doivent également être appliqués.

$$\mathcal{P}(\sigma, Z) = \ln(1 + \rho_\sigma) + \ln(1 + \rho_{\sigma,Z})$$

4. Pour le jeu de nommage, le répertoire des relations symbole-signature supporte ce *couplage structurel* induit par les itérations de jeux de langage, qui génèrent des symboles et des signatures renforcés ou non selon leurs succès et qui mènent à la co-évolution entre le dictionnaire de symboles et le dictionnaire de signatures. C'est sur ces trois capacités d'adaptation que nous comptons pour capturer l'expression du sens par l'utilisateur.

Enfin, à la cloture d'une session d'assistance, il est nécessaire à l'agent alter ego de disposer de l'avis de l'utilisateur pour compléter les itérations de jeux initiées lors de la dernière tentative d'assistance. Selon l'acceptation ou le rejet de cette tentative par l'utilisateur, la signature considérée, les symboles de cette signature et les relations associées sont renforcés ou inhibés. Ce traitement est commun à la dernière tentative d'une session à tentatives multiples (scénario Sc_3) et à la tentative unique du scénario Sc_2.

Algorithme de construction d'un nouvel état de langage

Dans cette section, nous allons donner les algorithmes qui assurent la construction d'un nouvel état de langage ; elle fait suite à la négociation, détaillée sur la figure 7.8, et correspond à l'appel de la fonction ConstructionEtatLangage.

La figure 7.10 donne l'algorithme de la fonction ConstructionEtatLangage. Cette fonction teste les différentes alternatives d'enchaînement de tentatives que nous avons introduites dans notre structuration de l'assistance, et appelle la fonction interne Alternative_X pour effectuer le traitement spécifique à l'alternative X. Après la construction d'un état de langage L_{n+k+1}, selon les différentes tentatives, le mécanisme de pression sélective est appliqué à cet nouvel état de langage, via l'appel à la fonction AppliquerPressionSelective.

Fonction ConstructionEtatLangage
  entrée : précisions $\{P_i\}$, précision de signature $P_{Z^{(k)}}$, état de langage L_{n+k}
  sortie : nouvel état de langage L_{n+k+1}
début
  si $(Z^{(k)} \in L_{n+k})$ ET $Z^{(k)} = Z^{(k+1)}$ **alors**
    $L_{n+k+1} \leftarrow$ Alternative_1$(\{P_i\}, P_{Z^{(k)}}, L_{n+k})$
  fin-si
  si $(Z^{(k+1)} \in L_{n+k})$ ET $(Z^{(k)} \neq Z^{(k+1)})$ **alors**
    $L_{n+k+1} \leftarrow$ Alternative_2$(\{P_i\}, P_{Z^{(k)}}, L_{n+k})$
  fin-si
  si $(Z^{(k+1)} \ni L_{n+k})$ ET $(Z^{(k)} \neq Z^{(k+1)})$ **alors**
    $L_{n+k+1} \leftarrow$ Alternative_3$(\{P_i\}, P_{Z^{(k)}}, L_{n+k})$
  fin-si
  si $k = 0$ ET $Z^{(0)} = nouv.sign.$ tel que $Z^{(1)} \ni L_n$ **alors**
    $L_{n+1} \leftarrow$ Alternative_4$(\{P_i\}, P_{Z^{(0)}}, L_n)$
  sinon
    $L_{n+1} \leftarrow$ Alternative_2$(\{P_i\}, P_{Z^{(0)}=null}, L_n)$
  fin-si
  L_{n+k+1}.AppliquerPressionSelective()
  retourne L_{n+k+1}
fin

Fig. 7.10 – Algorithme chapeau de construction d'un nouvel état de langage

Le traitement des différentes alternatives est basé sur le même principe, comme l'utilisateur a pu le constater dans le détail des mécanismes de renforcement de notre transposition. Nous allons expliquer en détail la fonction Alternative_1, et présenterons les traitements des autres alternatives en mettant en avant leurs différences avec la première.

La fonction Alternative_1 est donnée sur la figure 7.11. Les opérations que nous allons détailler doivent être vues comme la construction d'un nouveau "modèle" d'état de langage, selon les principes de description en RDF qui sont présentés en A.6.

La première opération, via l'appel à la fonction DupliquerLangageSaufSignature, consiste à créer le nouvel état de langage L_{n+k+1} en reportant, avec leurs pondérations, 1) tous les symboles de L_{n+k}, 2) toutes les signatures exceptée celle en paramètre et 3) toutes les relations

Fonction Alternative_1
entrée : précisions $\{P_i\}$, précision de signature $P_{Z^{(k)}}$, état de langage L_{n+k}
sortie : nouvel état de langage L_{n+k+1}
début
$L_{n+k+1} \leftarrow L_{n+k}$.DupliquerLangageSaufSignature($Z^{(k)}$)
L_{n+k+1}.AjouterSignature($Z^{(k)}$, L_{n+k}.Pondération($Z^{(k)}$))
L_{n+k+1}.IncrémenterSignature($Z^{(k)}$)
$ParcoursDef \leftarrow P_{Z^{(k)}}.definition$
pour toute précision P_i **tel que** $P_i.symbole \in ParcoursDef$ **faire**
si $P_i.action = Création$ **alors**
L_{n+k+1}.AjouterSymbole($P_i.symbole$, POND_INI_SYMB)
L_{n+k+1}.AjouterRelation($Z^{(k)}$, $P_i.symbole$, POND_INI_REL)
fin-si
si $P_i.action = Evolution$ **alors**
L_{n+k+1}.AjouterSymbole($P_i.symbole$, L_{n+k}.Pondération($P_i.reaction.symbole$))
L_{n+k+1}.IncrémenterSymbole($P_i.symbole$)
L_{n+k+1}.AjouterRelation($Z^{(k)}$, $P_i.symbole$, L_{n+k}.Pondération($Z^{(k)}$, $P_i.reaction.symbole$))
L_{n+k+1}.IncrémenterRelation($Z^{(k)}$, $P_i.symbole$)
fin-si
si $P_i.action = Remplacement$ **alors**
L_{n+k+1}.IncrémenterSymbole($P_i.symbole$)
L_{n+k+1}.AjouterRelation($Z^{(k)}$, $P_i.symbole$, L_{n+k}.Pondération($Z^{(k)}$, $P_i.reaction.symbole$))
L_{n+k+1}.IncrémenterRelation($Z^{(k)}$, $P_i.symbole$)
fin-si
$ParcoursDef \leftarrow ParcoursDef - P_i.symbole$
fin-pour
pour tout $symbole \in ParcoursDef$ **faire**
L_{n+k+1}.IncrémenterSymbole($symbole$)
si L_{n+k}.ExisteRelation($Z^{(k)}$, $symbole$) **alors**
L_{n+k+1}.AjouterRelation($Z^{(k)}$, $symbole$, L_{n+k}.Pondération($Z^{(k)}$, $symbole$))
L_{n+k+1}.IncrémenterRelation($Z^{(k)}$, $symbole$)
sinon
L_{n+k+1}.AjouterRelation($Z^{(k)}$, $symbole$, POND_INI_REL)
fin-si
fin-pour
L_{n+k+1}.DécrémenterSymboles(L_{n+k}.DéfinitionSymbolique($Z^{(k)}$) $- P_{Z^{(k)}}.definition$)
retourne L_{n+k+1}
fin

Fig. 7.11 – Algorithme de construction d'un nouvel état de langage pour l'alternative *A1*

Fonction Alternative_2
entrée : précisions $\{P_i\}$, précision de signature $P_{Z^{(k)}}$, état de langage L_{n+k}
sortie : nouvel état de langage L_{n+k+1}
début
$L_{n+k+1} \leftarrow L_{n+k}$.DupliquerLangageSaufSignature($Z^{(k+1)}$)
L_{n+k+1}.AjouterSignature($Z^{(k+1)}$, L_{n+k}.Pondération($Z^{(k+1)}$))
L_{n+k+1}.DécrémenterSignature($Z^{(k)}$)
L_{n+k+1}.IncrémenterSignature($Z^{(k+1)}$)
$ParcoursDef \leftarrow P_{Z^{(k)}}.definition$
pour toute précision P_i **tel que** $P_i.symbole \in ParcoursDef$ **faire**
si $P_i.action = Création$ **alors**
L_{n+k+1}.AjouterSymbole($P_i.symbole$, POND_INI_SYMB)
L_{n+k+1}.AjouterRelation($Z^{(k+1)}, P_i.symbole$, POND_INI_REL)
fin-si
si $P_i.action = Evolution$ **alors**
L_{n+k+1}.AjouterSymbole($P_i.symbole, L_{n+k}$.Pondération($P_i.reaction.symbole$))
L_{n+k+1}.IncrémenterSymbole($P_i.symbole$)
L_{n+k+1}.AjouterRelation($Z^{(k+1)}, P_i.symbole$,
L_{n+k}.Pondération($Z^{(k)}, P_i.reaction.symbole$))
L_{n+k+1}.IncrémenterRelation($Z^{(k+1)}, P_i.symbole$)
fin-si
si $P_i.action = Remplacement$ **alors**
L_{n+k+1}.IncrémenterSymbole($P_i.symbole$)
L_{n+k+1}.AjouterRelation($Z^{(k+1)}, P_i.symbole$,
L_{n+k}.Pondération($Z^{(k)}, P_i.reaction.symbole$))
L_{n+k+1}.IncrémenterRelation($Z^{(k+1)}, P_i.symbole$)
fin-si
$ParcoursDef \leftarrow ParcoursDef - P_i.symbole$
fin-pour
pour tout $symbole \in ParcoursDef$ **faire**
L_{n+k+1}.IncrémenterSymbole($symbole$)
si L_{n+k}.ExisteRelation($Z^{(k+1)}, symbole$) **alors**
L_{n+k+1}.AjouterRelation($Z^{(k+1)}, symbole$,
L_{n+k}.Pondération($Z^{(k+1)}, symbole$))
L_{n+k+1}.IncrémenterRelation($Z^{(k+1)}, symbole$)
sinon
L_{n+k+1}.AjouterRelation($Z^{(k+1)}, symbole$, POND_INI_REL)
fin-si
fin-pour

(suite...)

```
(...suite)

pour tout symbole ∈ L_{n+k}.DéfinitionSymbolique(Z^(k)) faire
    L_{n+k+1}.DécrémenterRelation(Z^(k), symbole)
    si symbole ∋ P_{Z^(k)}.definition alors
        L_{n+k+1}.DécrémenterSymbole(symbole)
    fin-si
fin-pour
si P_{Z^(k)}.definition = L_{n+k}.DéfinitionSymbolique(Z^(k+1)) alors
    L_{n+k+1}.FusionnerSignatures(Z^(k), Z^(k+1))
fin-si
retourne L_{n+k+1}
fin
```

FIG. 7.12 – Algorithme de construction d'un nouvel état de langage pour l'alternative *A2*

des signatures retenues. La signature $Z^{(k)}$, qui était le sujet de la tentative k et qui est retenue pour la tentative $k+1$, est ajoutée avec sa pondération, puis renforcée. La nouvelle définition symbolique de $Z^{(k)}$, i.e., $P_{Z^{(k)}}.definition$, guide la suite de la construction de L_{n+k+1}. Selon les réactions retenues dans cette définition, qui ont fait l'objet de précisions de la part de l'utilisateur, il est distingué les cas de *création*, de *évolution* et du *remplacement* de symbole. Les symboles résultants sont ajoutés à L_{n+k+1}, ainsi que leurs relations, la distinction se fait dans les pondérations qui 1) en cas de création, sont initialisées à une valeur arbitraire, 2) en cas d'évolution, héritent des valeurs pour le symbole d'origine, et 3) en cas de remplacement, le symbole retenu est incrémenté et sa relation hérite de la pondération de la relation pour le symbole d'origine puis est incrémentée.

Ensuite, les symboles de la nouvelle définition, qui n'ont pas fait l'objet de réactions, sont incrémentés ; leur relation est incrémentée, si déjà existante, ou bien créée et initialisée à une valeur arbitraire. Enfin, l'ensemble des symboles qui étaient dans la définition de $Z^{(k)}$ dans L_{n+k} et qui ne sont plus dans la nouvelle définition sont inhibés ; leurs relations avec $Z^{(k)}$ sont supprimées.

La fonction ALTERNATIVE_2, donnée sur la figure 7.12, adopte des principes similaires. Un nouvel état de langage L_{n+k+1} est créé en dupliquant L_{n+k} excepté la signature cible de la tentative suivante. La signature $Z^{(k+1)}$, qui est retenue pour la tentative $k+1$, est ajoutée avec sa pondération, puis renforcée. La signature $Z^{(k)}$ est inhibée. La nouvelle définition symbolique de $Z^{(k+1)}$, i.e., $P_{Z^{(k)}}.definition$, guide la suite de la construction de L_{n+k+1}. Le traitement des réactions présentes dans cette définition suit les mêmes principes que dans la fonction ALTERNATIVE_1, la différence est dans le fait que les pondérations de relation héritent des pondérations pour les relations avec $Z^{(k)}$; cette approche est discutable, nous le faisons dans le sens d'un "transfert de confiance".

Ensuite, le même traitement est appliqué aux symboles de la nouvelle définition, qui n'ont pas fait l'objet de réactions. Les relations de $Z^{(k)}$, selon sa définition dans L_{n+k}, sont inhibées (échec de nommage) ; les symboles qui ne sont pas dans la nouvelle définition de $Z^{(k+1)}$ sont inhibés. Enfin, en cas d'ambiguïté entre les signatures $Z^{(k)}$ et $Z^{(k+1)}$, celles-ci sont fusonnées ; la signature résultante sera le sujet de la tentative $k+1$.

Fonction Alternative_3
entrée : précisions $\{P_i\}$, précision de signature $P_{Z^{(k)}}$, état de langage L_{n+k}
sortie : nouvel état de langage L_{n+k+1}
début
$L_{n+k+1} \leftarrow L_{n+k}$.DupliquerLangage()
L_{n+k+1}.AjouterSignature($Z^{(k+1)}$, POND_INI_SIGN)
L_{n+k+1}.DécrémenterSignature($Z^{(k)}$)
$ParcoursDef \leftarrow P_{Z^{(k)}}.definition$
pour toute précision P_i **tel que** $P_i.symbole \in ParcoursDef$ **faire**
si $P_i.action = Création$ **alors**
L_{n+k+1}.AjouterSymbole($P_i.symbole$, POND_INI_SYMB)
L_{n+k+1}.AjouterRelation($Z^{(k+1)}, P_i.symbole$, POND_INI_REL)
fin-si
si $P_i.action = Evolution$ **alors**
L_{n+k+1}.AjouterSymbole($P_i.symbole, L_{n+k}$.Pondération($P_i.reaction.symbole$))
L_{n+k+1}.IncrémenterSymbole($P_i.symbole$)
L_{n+k+1}.AjouterRelation($Z^{(k+1)}, P_i.symbole$, POND_INI_REL)
fin-si
si $P_i.action = Remplacement$ **alors**
L_{n+k+1}.IncrémenterSymbole($P_i.symbole$)
L_{n+k+1}.AjouterRelation($Z^{(k+1)}, P_i.symbole$, POND_INI_REL)
fin-si
$ParcoursDef \leftarrow ParcoursDef - P_i.symbole$
fin-pour
pour tout $symbole \in ParcoursDef$ **faire**
L_{n+k+1}.IncrémenterSymbole($symbole$)
L_{n+k+1}.AjouterRelation($Z^{(k+1)}, symbole$, POND_INI_REL)
fin-pour
pour tout $symbole \in (L_{n+k}$.DéfinitionSymbolique($Z^{(k)}$) **faire**
L_{n+k+1}.DécrémenterRelation($Z^{(k)}, symbole$)
si $symbole \ni P_{Z^{(k)}}.definition$ **alors**
L_{n+k+1}.DécrémenterSymbole($symbole$)
fin-si
fin-pour
retourne L_{n+k+1}
fin

Fig. 7.13 – Algorithme de construction d'un nouvel état de langage pour l'alternative *A3*

Fonction Alternative_4
  entrée : précisions $\{P_i\}$, précision de signature $P_{Z^{(0)}}$, état de langage L_n
  sortie : nouvel état de langage L_{n+1}
début
  $L_{n+1} \leftarrow L_n$.DupliquerLangage()
  L_{n+1}.AjouterSignature($Z^{(1)}$, POND_INI_SIGN)
  $ParcoursDef \leftarrow P_{Z^{(0)}}.definition$
  pour toute précision P_i **tel que** $P_i.symbole \in ParcoursDef$ **faire**
    si $P_i.action = Création$ **alors**
      L_{n+1}.AjouterSymbole($P_i.symbole$, POND_INI_SYMB)
      L_{n+1}.AjouterRelation($Z^{(1)}, P_i.symbole$, POND_INI_REL)
    fin-si
    si $P_i.action = Evolution$ **alors**
      L_{n+1}.AjouterSymbole($P_i.symbole, L_n$.Pondération($P_i.reaction.symbole$))
      L_{n+1}.IncrémenterSymbole($P_i.symbole$)
      L_{n+1}.AjouterRelation($Z^{(1)}, P_i.symbole$, POND_INI_REL)
    fin-si
    si $P_i.action = Remplacement$ **alors**
      L_{n+1}.IncrémenterSymbole($P_i.symbole$)
      L_{n+1}.AjouterRelation($Z^{(1)}, P_i.symbole$, POND_INI_REL)
    fin-si
    $ParcoursDef \leftarrow ParcoursDef - P_i.symbole$
  fin-pour
  pour tout $symbole \in ParcoursDef$ **faire**
    L_{n+1}.IncrémenterSymbole($symbole$)
    L_{n+1}.AjouterRelation($Z^{(1)}, symbole$, POND_INI_REL)
  fin-pour
  retourne L_{n+1}
fin

Fig. 7.14 – Algorithme de construction d'un nouvel état de langage pour l'alternative *A4*

Dans la fonction ConstructionEtatLangage (figure 7.10), l'appel à Alternative_2, pour $k = 0$ dans le cas inverse conduisant à l'appel de Alternative_2, nous plaçons en paramètre $Z(0)$ qui est inexistante dans L_{n+k}, soit égale à *null*. Dans ce cas, nous considérons que tous les traitements impliquant $Z(0)$ ne sont pas effectués.

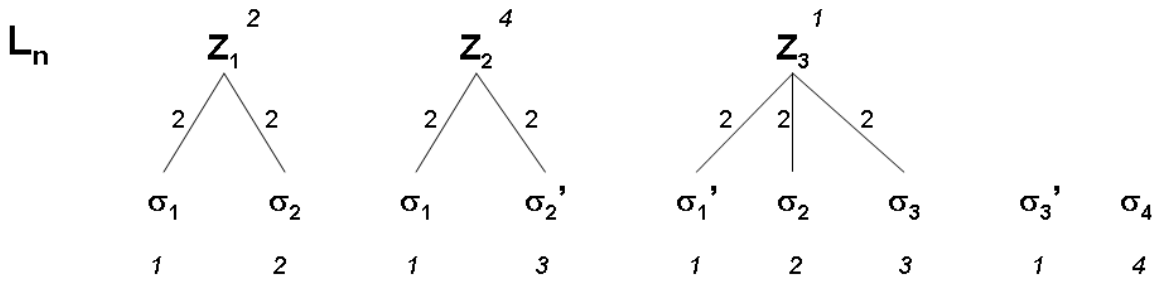
La fonction Alternative_3, donnée sur la figure 7.13, adopte des principes similaires. Toutefois, le nouvel état de langage L_{n+k+1} est issu de la duplication intégrale de L_{n+k}. La nouvelle signature $Z^{(k+1)}$ est créée et ajoutée avec une pondération arbitraire ; $Z^{(k)}$ est inhibée. La définition symbolique de $Z^{(k+1)}$, i.e., $P_{Z^{(k)}}.definition$, guide la suite de la construction de L_{n+k+1}. Le traitement des réactions présentes dans cette définition suit les mêmes principes que dans les fonctions Alternative_1 et Alternative_2, mais un seul héritage est présent, lors de l'évolution d'un symbole ; toutes les autres pondérations sont fixées à des valeurs arbitraires. La fin du traitement est la même que dans la fonction Alternative_1, excepté dans la première boucle où nous n'avons par principe pas à traiter l'existence d'une définition préalable.

La fonction Alternative_4, donnée sur la figure 7.14, adopte des principes très proches de ceux de la fonctions Alternative_3, excepté les traitements qui portent sur $Z^{(k)}$ vu que dans ce cas cette signature est inexistante dans L_{n+k}. Ainsi, la dernière boucle de la fonction Alternative_3, portant sur la définition de $Z^{(k)}$ dans L_{n+k}, n'est pas présente dans la fonction Alternative_4.

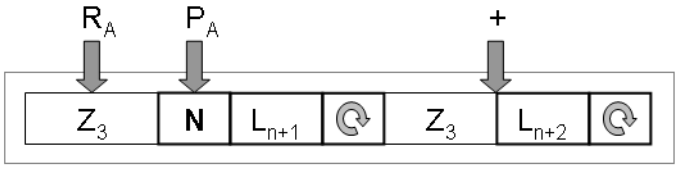
7.6 Exemple d'évolution d'un langage

Nous allons illustrer les mécanismes, que nous avons décrits dans la section précédente ; nous n'aborderons pas toutes les alternatives car cela correspondrait à une situation défavorable à l'émergence d'un langage, dont nous voulons également illustrer la dynamique. Nous partons d'un état de langage L_n, qui est défini sur la figure ci-dessous. Cet état de langage est composé de trois signatures (Z_1, Z_2 et Z_3), qui sont exprimées avec des symboles (σ_1, σ'_1, σ_2, σ'_2 et σ_3) ; à cela s'ajoute des symboles qui sont inutilisés (σ'_3 et σ_4). Les traits entre les signatures et les symboles matérialisent la présence des relations symbole-signature, qui définissent une expression symbolique d'une signature. Les signatures partagent plusieurs symboles (σ_1, σ_2 et σ_3), qui apparaissent plusieurs fois pour des raisons de lisibilité.

Les chiffres représentent les valeurs de pondération des différents éléments :
- pour une signature Z_i : au-dessus à droite, ρ_{Z_i} ;
- pour un symbole σ_j : en-dessous, ρ_{σ_j} ;
- pour une relation entre Z_i et σ_j, à coté du trait : ρ_{Z_i,σ_j}.



L'utilisateur a effectué une série d'opérations avec l'application cible, et il consulte son agent alter ego. Ce dernier lui propose une interprétation sur la base de Z_3. L'utilisateur la choisit, ce qui initie une session d'assistance dont la première tentative porte sur Z_3. La figure ci-dessous schématise la première session composée de deux tentatives : la première avec négociation et la seconde avec un avis favorable de l'utilisateur.



Nous rappelons qu'un renforcement d'un élément de langage est synonyme de succès d'une itération pour cet élément ; inversement, l'inhibition (ou une suppression directe) est synonyme de l'échec de cette itération.

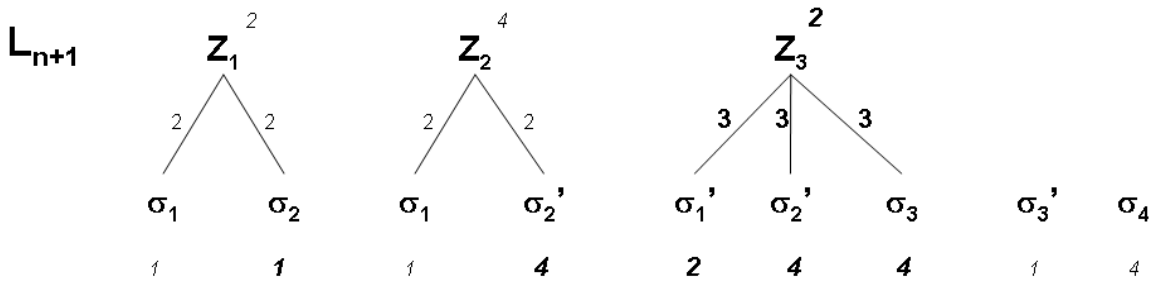
Lors de la première tentative, l'utilisateur apporte les informations suivantes :

- $\mathbf{R_A}$: il réagit sur la section où σ_2 est apparié, et modifie ce symbole ;
- $\mathbf{P_A}$: la négociation de σ_2 permet à l'agent de considérer cette modification comme une évolution de ce symbole. L'agent indique que cette évolution a abouti à un symbole existant déjà (σ'_2) ; l'utilisateur accepte cela et σ'_2 prend la place de σ_2.

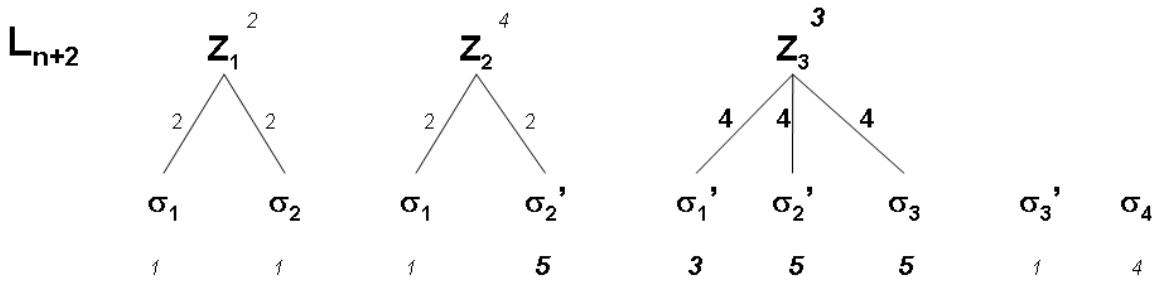
L'application de nos jeux de langage et la prise en compte du remplacement fait que :

- ρ_{Z_3} est renforcé ;
- $\rho_{\sigma'_1}$, $\rho_{\sigma'_2}$ et ρ_{σ_3} sont renforcés. ρ_{σ_2} est inhibé ;
- ρ_{Z_3,σ'_1} et ρ_{Z_3,σ_3} sont renforcés. La relation $Z_3 - \sigma'_2$ remplace la relation $Z_3 - \sigma_2$; ρ_{Z_3,σ'_2} hérite de ρ_{Z_3,σ_2} et est renforcé.

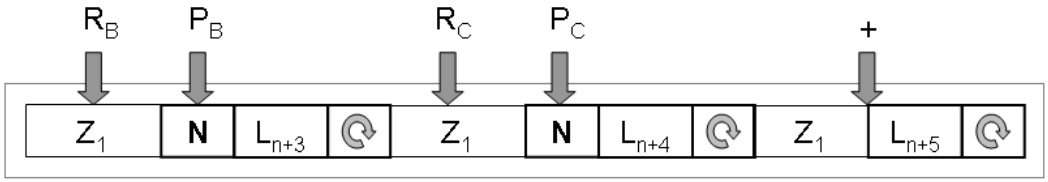
La figure ci-dessous définit le nouvel état de langage obtenu (L_{n+1}).



Dans la deuxième tentative, l'utilisateur met fin à la session d'assistance en émettant un avis favorable sur l'élaboration. Cela conduit au succès de toutes les itérations de jeux de langage, i.e., le renforcement de la signature, de ses symboles et de ses relations. Un nouvel état (L_{n+2}), défini ci-dessous, est créé. Ce traitement correspond à celui effectué dans le cas de la tentative unique du scénario Sc_2.



Par la suite, l'utilisateur lance une autre session d'assistance en choisissant Z_1 comme objet d'assistance lors des trois tentatives qui la composent (c.f., alternative A_1). La figure ci-dessous illustre cette session.



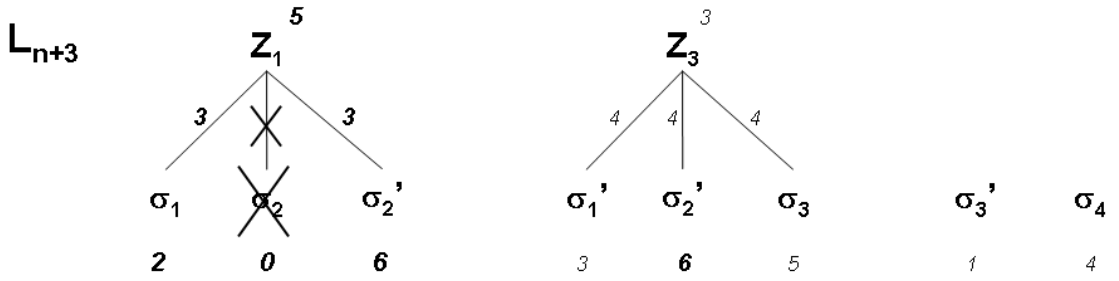
Dans la première tentative, l'utilisateur apporte des informations similaires à R_A et P_A :

- **R_B** : il réagit sur la section où σ_2 est apparié, et modifie ce symbole ;
- **P_B** : la négociation de σ_2 permet à l'agent de considérer cette modification comme une évolution de ce symbole. L'agent indique que cette évolution a abouti à un symbole existant déjà (σ'_2) ; l'utilisateur accepte ce point de vue et σ'_2 prend la place de σ_2. Cette nouvelle définition conduit à une égalité entre Z_1 et Z_2 ; l'utilisateur définit un label commun pour ces deux signatures et elles sont fusionnées.

L'application de nos jeux de langage, en tenant compte du remplacement de symbole et de la fusion de signature, fait que :

- ρ_{Z_1} hérite de ρ_{Z_2}, car plus grand, et est renforcé ;
- ρ_{σ_1} et $\rho_{\sigma'_2}$ sont renforcés. ρ_{σ_2} est inhibé ;
- ρ_{Z_1,σ_1} est renforcé. La relation $Z_1 - \sigma'_2$ remplace la relation $Z_1 - \sigma_2$; ρ_{Z_1,σ'_2} hérite de ρ_{Z_1,σ_2} et est renforcé.

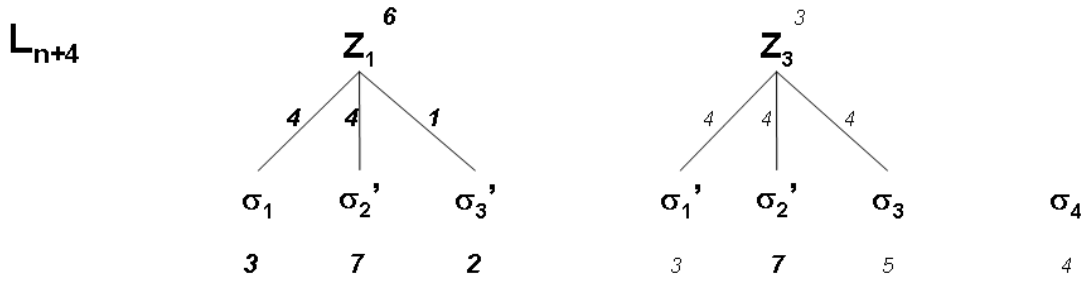
La pression sélective conduit à la suppression de σ_2, car de pondération nulle. La figure ci-dessous décrit le nouvel état de langage obtenu (L_{n+3}).



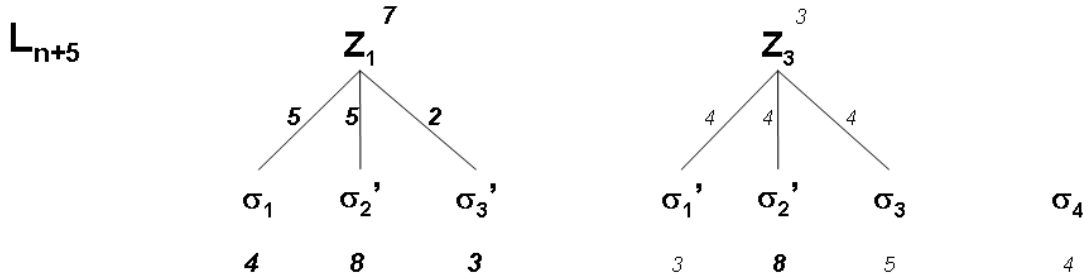
Dans la deuxième tentative, l'utilisateur apporte les informations suivantes :
- $\mathbf{R_C}$: il n'effectue aucune réaction avec les éléments symboliques, mais lance une mise à jour afin de modifier directement la définition de la signature Z_1 ;
- $\mathbf{P_C}$: il ajoute le symbole σ'_3 dans la définition de Z_1.

L'application de nos mécanismes, en tenant compte de l'ajout de symbole, fait que :
- ρ_{Z_1} est renforcé ;
- ρ_{σ_1}, $\rho_{\sigma'_2}$ et $\rho_{\sigma'_3}$ sont renforcés ;
- ρ_{Z_1,σ_1} et ρ_{Z_1,σ'_2} sont renforcés. La relation $Z_1 - \sigma'_3$ est créée et ρ_{Z_1,σ'_2} est fixé à une valeur initiale (ici, 1).

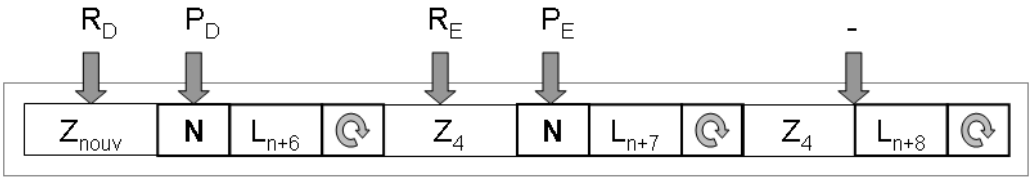
La figure ci-dessous décrit le nouvel état de langage obtenu (L_{n+4}).



Dans la dernière tentative, l'utilisateur termine la session en émettant un avis favorable. La figure ci-dessous décrit l'état L_{n+5} qui est construit selon la même méthode que pour l'état L_{n+2}.



Ensuite, l'utilisateur lance une autre session d'assistance en créant une nouvelle signature (Z_{nouv}), c.f., alternative A_4. La figure ci-dessous illustre cette session.



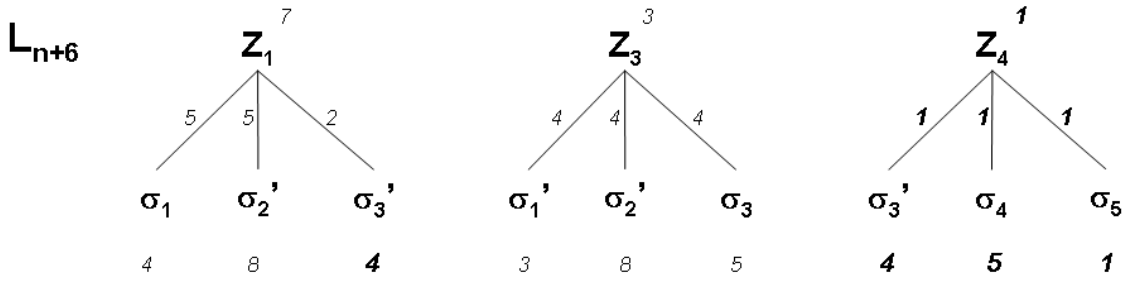
Dans la première tentative, l'utilisateur apporte les informations suivantes :
- $\mathbf{R_D}$: il réagit sur trois éléments symboliques ;
- $\mathbf{P_D}$: l'agent lui indique que deux de ses réactions ont conduit à des symboles existants (σ_3 et σ_4) ; l'utilisateur accepte ce point de vue. La dernière réaction est traitée comme la

création d'un nouveau symbole (σ_5).

L'application de nos mécanismes, en tenant compte des créations de symbole et de signature, fait que :

- Z_4 est créée et ρ_{Z_4} est initialisé à la valeur arbitraire de 1 ;
- ρ_{σ_3}, ρ_{σ_4} sont renforcés. σ_5 est créé et ρ_{σ_5} est initialisé à la valeur arbitraire de 1 ;
- Les relations $Z_4 - \sigma_3$, $Z_4 - \sigma_4$ et $Z_4 - \sigma_5$ sont créées. ρ_{Z_4,σ_3}, ρ_{Z_4,σ_4} et ρ_{Z_4,σ_5} sont initialisés à la valeur arbitraire de 1.

La figure ci-dessous décrit le nouvel état de langage obtenu (L_{n+6}).



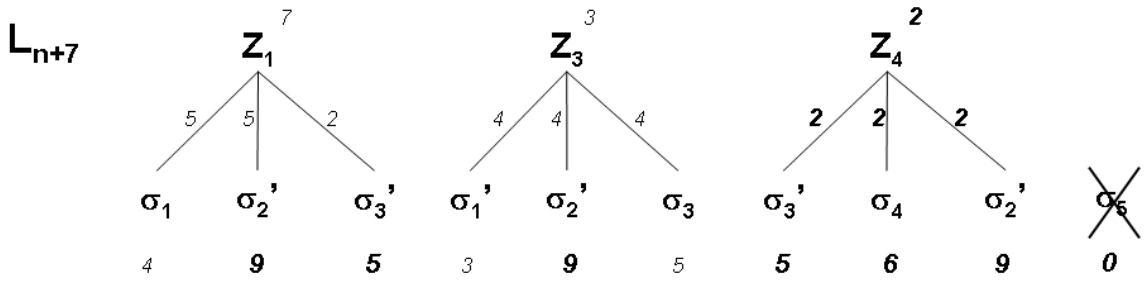
Dans la deuxième tentative, l'utilisateur apporte les informations suivantes :

- $\mathbf{R_E}$: il modifie σ_5 ;
- $\mathbf{P_E}$: l'agent indique que cette évolution a abouti à un symbole existant déjà (σ'_2) ; l'utilisateur accepte ce point de vue et σ'_2 prend la place de σ_2.

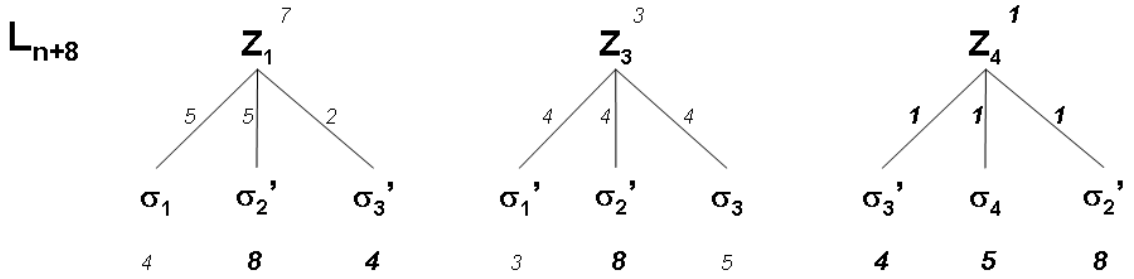
L'application de nos jeux de langage, en tenant compte du remplacement de symbole, fait que :

- ρ_{Z_4} est renforcé ;
- ρ_{σ_1} et $\rho_{\sigma'_2}$ sont renforcés. ρ_{σ_2} est inhibé ;
- ρ_{Z_4,σ'_3} et ρ_{Z_4,σ_4} sont renforcés. La relation $Z_1 - \sigma'_2$ remplace la relation $Z_1 - \sigma_5$; ρ_{Z_1,σ'_2} hérite de ρ_{Z_1,σ_5} et est renforcé.

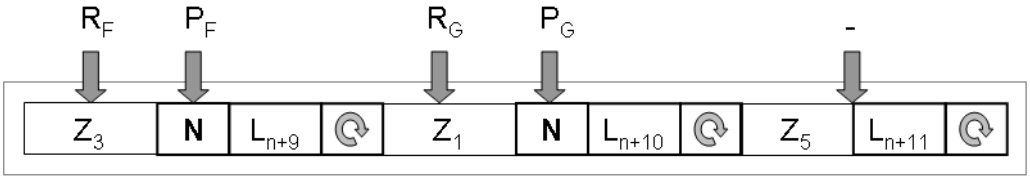
La pression sélective conduit à la suppression de σ_5, car de pondération nulle. La figure ci-dessous décrit le nouvel état de langage obtenu (L_{n+7}).



Dans la dernière tentative, l'utilisateur termine la session en émettant un avis favorable. La figure ci-dessous décrit l'état L_{n+8} qui est construit selon similaire à celle utilisée pour les états L_{n+2}, L_{n+5}, mais avec un échec de toutes les itérations donc une inhibition complète.



Dans une dernière session, l'utilisateur s'intéresse tout d'abord à la signature Z_3 ; puis, via le processus de négociation, il accède à la signature similaire Z_1 : nous illustrons l'alternative d'enchaînement A_2. La figure ci-dessous illustre cette dernière session.



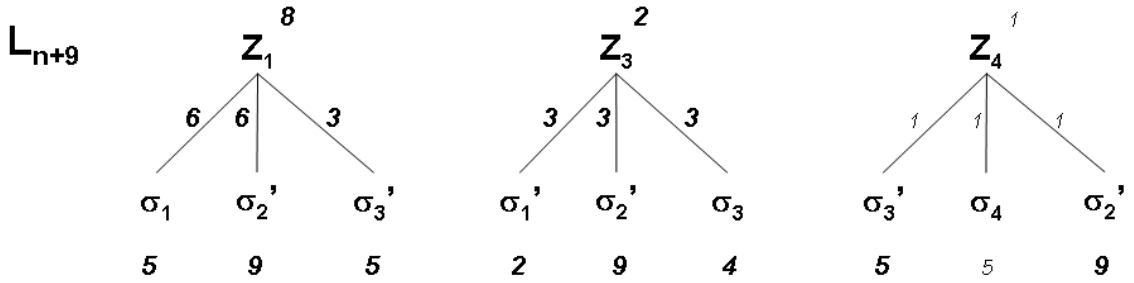
Les informations apportées par l'utilisateur sont donc :

- $\mathbf{R_F}$: aucune réaction ;
- $\mathbf{P_F}$: il sélectionne la signature Z_1 comme objet de la prochaine tentative.

L'application de nos jeux de langage pour ce changement de signature fait que :

- ρ_{Z_2} est inhibé et ρ_{Z_1} est renforcé ;
- ρ_{σ_1}, $\rho_{\sigma'_2}$ et $\rho_{\sigma'_3}$ sont renforcés. $\rho_{\sigma'_1}$ et ρ_{σ_3} sont inhibés ;
- ρ_{Z_1,σ_1}, ρ_{Z_1,σ'_2} et ρ_{Z_1,σ'_3} sont renforcés. ρ_{Z_3,σ'_1}, ρ_{Z_3,σ'_2} et ρ_{Z_3,σ_3} sont inhibés.

La figure ci-dessous décrit le nouvel état de langage obtenu (L_{n+9}).



Dans la deuxième tentative, les informations apportées par l'utilisateur sont :

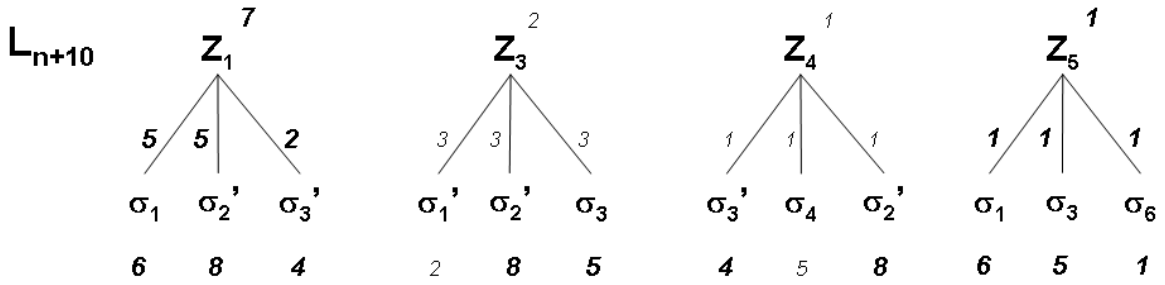
- $\mathbf{R_G}$: il modifie σ'_2 ;
- $\mathbf{P_G}$: il indique à l'agent alter ego de traiter sa modification comme une création de symbole (σ_6), et indique qui crée une nouvelle signature (Z_5) en remplaçant σ'_3 par σ_3 dans la définition.

L'application de nos jeux de langage, en tenant compte de la création de symbole et de signature,

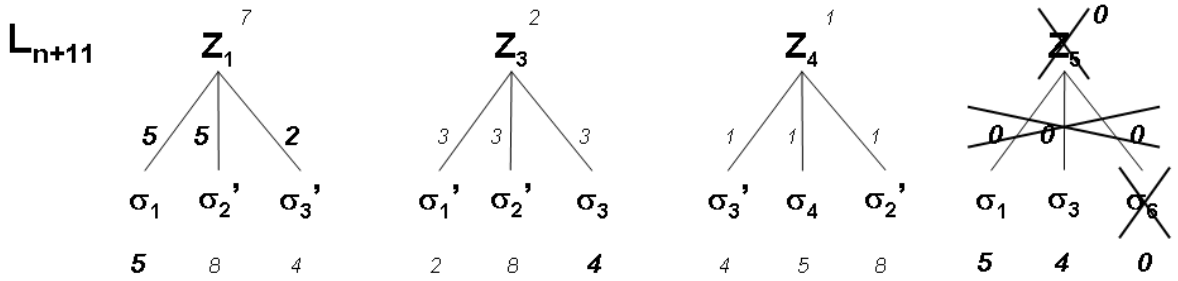
fait que :

- Z_5 est créée et ρ_{Z_5} est initialisé à la valeur arbitraire de 1. ρ_{Z_1} est inhibé ;
- ρ_{σ_1}, ρ_{σ_3} sont renforcés. σ_6 est créé et ρ_{σ_6} est initialisé à la valeur arbitraire de 1. $\rho_{\sigma'_2}$ et $\rho_{\sigma'_3}$ sont inhibés ;
- les relations $Z_5 - \sigma_1$, $Z_5 - \sigma_3$ et $Z_5 - \sigma_6$ sont créées ; ρ_{Z_5,σ_1}, ρ_{Z_5,σ_3} et ρ_{Z_5,σ_6} sont initialisés à la valeur arbitraire de 1. ρ_{Z_1,σ_1}, ρ_{Z_1,σ'_2} et ρ_{Z_1,σ'_3} sont inhibés.

La figure ci-dessous décrit le nouvel état de langage obtenu (L_{n+10}).



Dans la dernière tentative, entraînant l'alternative A'_3, l'utilisateur termine la session en émettant un avis favorable. La figure ci-dessous décrit l'état L_{n+11} qui est construit selon la même méthode que pour L_{n+8}. La pression sélective conduit à la suppression des relations $Z_5 - \sigma_1$, $Z_5 - \sigma_3$, $Z_5 - \sigma_6$, de la signature Z_5 et du symbole σ_6.



Pour faciliter la lecture de cet exemple, nous allons faire une illustration graphique de l'évolution des pondérations sur les figures suivantes. La figure 7.15 donne l'évolution des pondérations des signatures.

La figure 7.16 donne l'évolution des pondérations de symboles (s pour σ). Pour faciliter la lecture, les courbes superposées sont décalées.

La figure 7.17 illustre l'évolution des pondérations pour les relations de la signature Z_1. Les courbes $Z1 - s1$ et $Z1 - s2'$ sont en réalité superposées, mais elles ont été décalées pour des raisons de lisibilité.

Cet exemple est bien entendu trop court pour permettre l'établissement d'un langage. Néanmoins, il apparaît déjà sur les figures un renforcement des sens partagés, par exemple : la signature Z_1, le symbole σ'_2 et l'expression de Z_1 avec les symboles σ_1 et σ'_2, via les relations symbole-signature correspondantes.

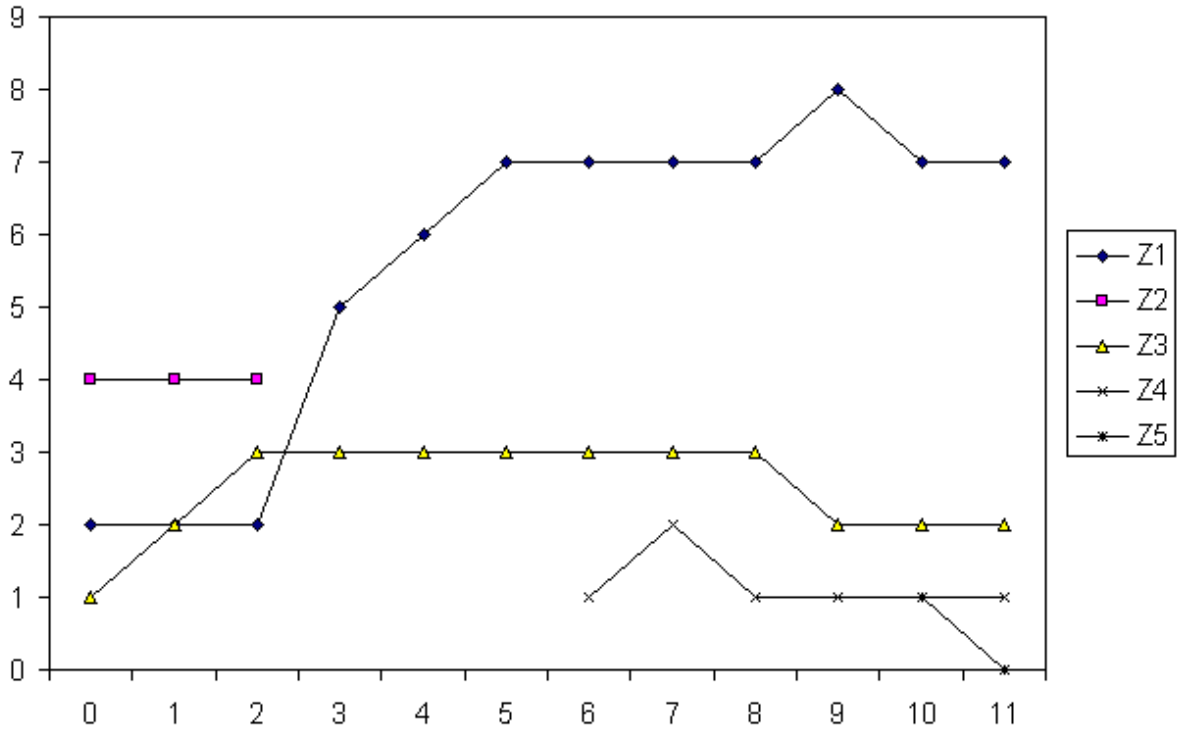


FIG. 7.15 – Evolution des pondérations de signatures

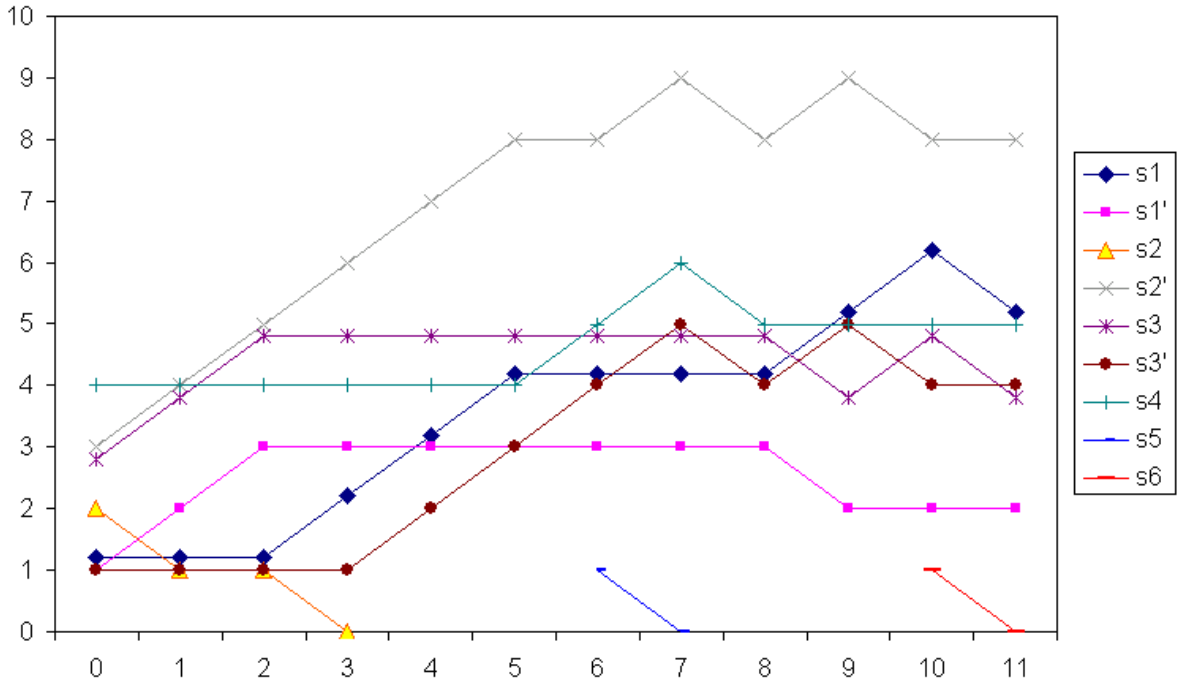


FIG. 7.16 – Evolution des pondérations de symboles

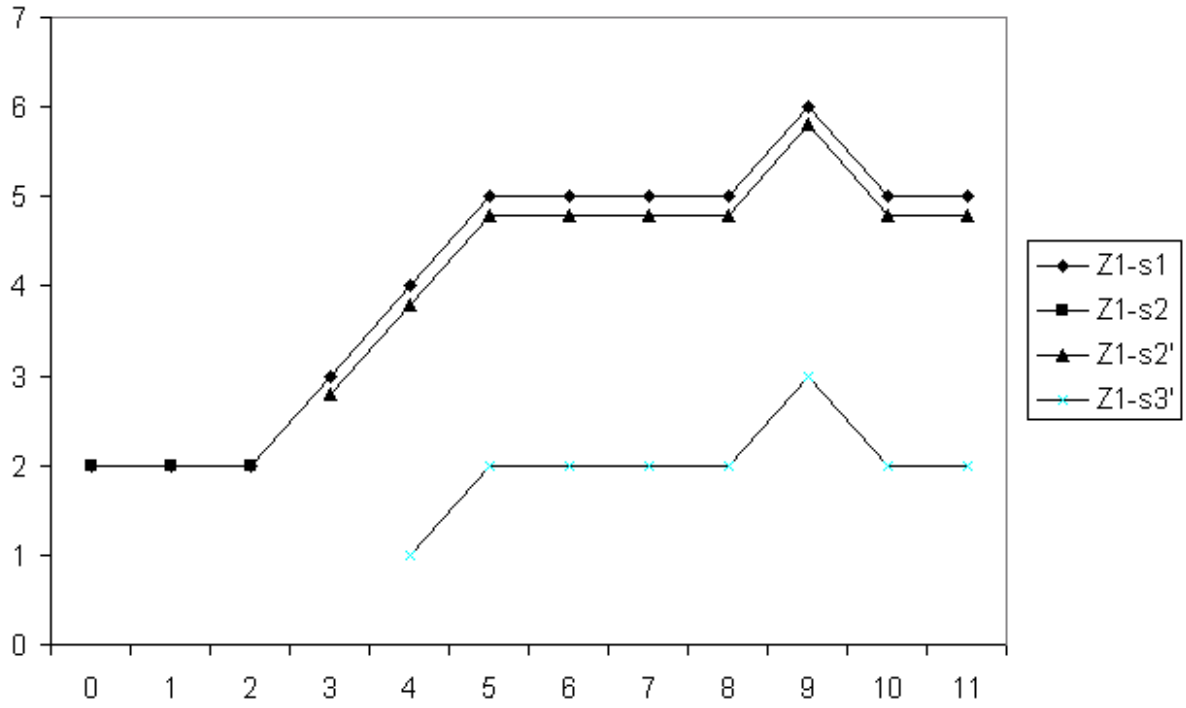


FIG. 7.17 – Evolution des pondérations de relations symbole-signature

Chapitre 8

Discussion et travaux connexes

Sommaire

Dans ce chapitre, nous allons discuter l'approche et ses deux principes fondamentaux, que nous venons de détailler dans les trois chapitres précédents. Nous mettrons ensuite notre travail en perspective par rapport à des travaux connexes.

8.1 Discussion de l'approche et des principes

Dans cette section, nous allons discuter les deux grandes grandes fonctionnalités de notre agent alter ego : son module de RàPET, et son module d'acquisition de sens par négociation. Ces remarques viennent compléter des points déjà abordés dans les chapitres précédents.

8.1.1 Le module de RàPET

Dans l'approche que nous avons présentée, nous proposons des mesures de similarité qui sont génériques, en s'appuyant sur les principes de [Tversky 77]. Ces différentes mesures peuvent être spécialisées pour un certain domaine d'application. Nous traiterons en perspectives (c.f., 10.1) de la possibilité d'affiner ces mesures en pondérant les captures dans un arbre d'observation, des possibilités d'initialiser ces pondérations, et des problèmes que cela soulève.

D'autre part, la vérification de la sémantique lors de l'appariement d'une variable de pattern avec une capture d'observation de trace est naturellement spécifique au modèle de domaine. Dans notre exemple, l'appariement d'une variable de pattern est une opération très simple. Pour d'autres domaines d'application, une capture variable pourrait représenter une vue partielle d'un OI et nécessiter un appariement plus complexe.

Dans notre approche, nous avons passé sous silence la question du *ciblage* de l'interprétation, i.e., de la sélection des actions de l'utilisateur qui doivent faire l'objet d'une interprétation. Dans l'absolu cette étape est importante si nous considérons un environnement informatique complexe, utilisé par l'acteur pour réaliser conjointement des tâches variées. En effet, la sélection

des observés est fondamentale pour pouvoir obtenir des interprétations pertinentes et éviter élaborations parasites.

Cette problématique est à rapprocher de la détermination d'*épisodes de conception*, traitée par [Champin 02] en application à CATIA[1], environnement de CAO. CATIA dispose d'un nombre très important de fonctionnalités et des études ergonomiques ont montré que l'utilisation de certaines d'entre elles (des opérations de visualisation, en vision globale, dans toutes les vues) marquent un contrôle du résultat par l'utilisateur et constitue une marque de fin d'épisode de conception. Il y a donc la possibilité de déterminer automatiquement les marqueurs de début et de fin d'épisode sur la bases de ces opérations significatives de l'acteur. Il est toujours possible de laisser à l'acteur le moyen de désigner explicitement ces épisodes.

Dans un environnement documentaire, la logique fonctionnelle adoptée lors de la conception d'une application cible peut faciliter la délimitation d'épisode. Cela vient du fait que les actions possibles de l'utilisateur sont peu nombreuses et qu'elles portent sur des objets précis (des documents, représentés par des ensembles de mots clés) qui sont structurés (décomposition en dossier et entité) simplement. Cela n'est aucunement comparable à la complexité fonctionnelle et à la nature des objets manipulés dans un environnement tel que CATIA ; nous reviendrons sur ce travail dans la section suivante, car il constitue un des principaux fondements du nôtre. Toutefois, même avec ces simplifications que permet un environnement documentaire, la caractérisation de certaines formes d'expérience nécessite un ciblage de l'interprétation non trivial, laissant toute sa complexité à la délimitation d'épisode ; l'ajout d'une dimension syntaxique pour le langage émergent sera discutée en perspective, elle peut être un support privilégié pour effectuer un ciblage pertinent.

La détermination du seuil d'intervention d'un assistant, que nous pouvons rapprocher de la problématique de détermination du seuil d'identification de signature τ_{idSign} abordée en 6.6.1, a été étudiée par [Mille 05]. En le transposant à notre cas, il apparaît que la détermination du seuil d'intervention ne peut se réduire à un seuil représentant simplement l'identification d'un certain nombre de symboles, mais qu'il est préférable de le mesurer par rapport à la complexité de la signature (le nombre total de symboles) ; en ce sens, la mesure $\tau_Z(trc, t)$ (c.f., équation 6.8) fait un calcul de ce ratio en le pondérant par l'importance que tient un symbole dans la signature considérée (exprimée par $\rho_{Z,\sigma} \cdot dim(\sigma)$). Néanmoins, un seuil τ_{idSign} fixe ne prend pas en compte une signature singulière, mais d'une utilisation moyenne des signatures. De plus, dans le cadre d'activités complexes, un acteur peut s'avérer expert dans un domaine et novice dans un autre. [Mille 05] proposent donc de laisser à l'acteur le moyen d'ajuster le seuil d'intervention. Cela se fait à l'aide de boutons +/- à coté de chaque épisode, qui permet au système d'évaluer un seuil en fonction de sa complexité. Les épisodes de même complexité rentrent dans la même catégorie. L'expérimentation montre que le seuil est élevé pour une complexité faible et diminue de façon non linéaire lorsque la complexité augmente. Nous discuterons en perspectives de l'intérêt d'ajouter un apprentissage de ce seuil d'intervention à une signature, à partir des utilisations qui en sont faites et retour de l'utilisateur via ses ajustements de seuil ; cette perspective soulève des problèmes similaires à celles de l'apprentissage des différentes pondérations d'un symbole.

Nous avons évoqué que certaines opérations nécessitent d'importantes ressources :

- la recherche d'épisodes antérieurs du module RàPET, qui se fait en interaction avec le module d'acquisition, est déclenchée à partir d'un seuil d'identification τ_{idSign}. Lors de

[1] CATIA - Dassault Systemes

nouvelles opérations de l'utilisateur, les épisodes déjà retrouvés doivent tenir compte de ces nouvelles acquisitions, et des algorithmes appropriés à cette mise à jour devraient être développés pour l'optimiser en tenant compte de l'identification d'épisodes déjà effectuée ;

– la mise à jour des propositions, qui est réalisée lors d'une nouvelle tentative d'assistance après la construction d'un nouvel état de langage. De même que la recherche d'épisodes, cette opération peut être optimisée en tenant compte de l'interprétation et des recherches de la tentative précédente.

A cela s'ajoute la problématique de sélectionner des traces candidates à la recherche d'épisodes, car il peut être coûteux de faire une recherche systématique dans toutes les traces antérieures.

8.1.2 Négociation de sens

L'émergence d'un langage de forme lexicale est théoriquement basée sur le respect de quatre principes énoncés par [Steels 00], et leur application à la communication entre robots a démontré expérimentalement la construction d'un système de communication partagé par les robots, i.e., l'agrément entre robots de répertoires de mots, de sens et de relations mot-sens pour interagir avec succès. Les quatre principes à respecter pour permettre l'émergence d'un tel système de communication sont : l'*apprentissage par renforcement*, l'*auto-organisation*, le *sélectionisme* et le *couplage structurel*. Notre transposition à la négociation de sens, détaillée en 7.5, suit ces principes en considérant : 1) l'étape d'élaboration, comme les capacités de perception suivie d'interprétation de l'agent alter ego, 2) l'étape de rapprochement avec des épisodes antérieurs comme ses capacités d'action et 3) le jugement explicite ou non de la pertinence de l'interprétation par l'utilisateur comme le moyen d'évaluer le succès d'une tentative de communication.

Nous avons effectué une transposition des jeux d'imitation et de nommage, et laissé à l'utilisateur le soin de créer des significations (exprimées en langage naturel par le label d'une signature) qu'il désigne par des symboles (via la définition de la signature), ce qui conduit à l'établissement de relations symboles-sens. Nous avons dépourvu l'agent de tout système donnant une capacité de discrimination, c'est pourquoi nous n'avons pas eu à transposer le jeu de discrimination. Pour des raisons évidentes d'expressivité, nous avons été néanmoins contraints de laisser s'établir plusieurs relations entre des symboles et un sens ; cette différence avec les approches détaillées par [Steels 00] ne va pas contre l'établissement d'un répertoire de relations symbole-sens, car il n'y a pas de mécanisme d'exclusion mutuelle des relations lors de leur renforcement ; la présence de plusieurs relations symbole-sens avec un poids fort pour une même signature est considérée par [Steels 00] comme l'expression d'une synonymie (plusieurs mots en concurrence pour exprimer un sens) tandis que nous le considérons comme une convention sur l'utilisation d'un ensemble de symboles pour désigner un sens.

La processus d'élaboration de l'agent alter ego, présentée en 6.6.1 est une méthode gloutonne, i.e., la première identification d'un symbole est celle qui est considérée dans l'identification d'une signature, cet appariement de symbole n'est pas remis en cause si un autre appariement de ce symbole, même meilleur, est trouvé par la suite. De plus, les instances manipulées par l'utilisateur (dans notre cas, des dossiers administratifs décomposés en entités documentaires, des formulaires), décrites avec des paquets d'OI (dans notre cas, des champs contenant des mots clés) ne peuvent pas figurer en tant que captures constantes dans un pattern de symbole, sinon ce pattern deviendrait spécifique à l'objet manipulé. En relâchant cette contrainte, considérons

deux symboles suffisants pour désigner une signature (i.e., après leur appariement, le taux d'identification de la signature est supérieur à τ_{idSign}) ; en appliquant notre mécanisme d'élaboration, un symbole peut avoir été apparié sur une section correspondant à une première action sur un document, et l'autre sur une section appartenant à une toute autre action sur un autre document. La recherche d'épisode sera déclenchée en tenant compte de ces appariements et il est très probable que les rapprochements effectuées soient dénués de sens pour l'utilisateur. Cette situation est principalement due à la pauvreté du modèle d'ensemble non ordonné pour exprimer une signature. L'introduction d'un moyen d'exprimer des relations et des contraintes entre et sur les symboles dans l'expression d'une signature est une nécessité pour augmenter son expressivité. Ce besoin nous conduit à nous intéresser en perspectives (c.f., 10.4) aux travaux sur l'émergence d'un langage de forme syntaxique.

8.2 Travaux connexes

En fin de section 4.3, portant sur la notion d'aide, nous avons parlé du travail de [Leray 07b] qui vise à intégrer, dès le début du design des environnements informatiques, des modèles des éléments cognitifs manipulés par l'utilisateur novice pour augmenter l'efficacité des fonctions d'assistance dont sont doté ces environnements ; ceci se faisant en voyant l'ensemble comme un mécanisme d'interaction entre une paire d'agents dialogiques (l'utilisateur et un environnement).

Antérieurement à ce travaux, [Ripoche 03] ont étudié les interactions langagières (en langue naturelle), sur la base d'une approche à base de traces, les acteurs d'un système collectif largement distribué : Bugzilla[2]. Ce travail fait l'étude des séquences que constituent les rapports de bugs et leur suivi, et particulièrement les parties informelles écrites collectivement en langue naturelle. Une approche selon des principes de traitement de la langue naturelle est adoptée, qui résulte entre autre sur une taxonomie des actes de dialogue, et des éléments qui permettent de parler d'une "anatomie" des bugs au regard des interactions qu'ils engendrent. Dans cette approche, il apparaît aussi le rôle central des traces, utilisés comme des objets à part entière dans les processus de résolution.

Dans la suite de ce travail, [Bonneaud 04] proposent divers modèles socio-cognitifs pour traiter des interactions telles qu'elles ont lieu dans différents collectifs. Ceci est fait dans l'hypothèse d'avoir à gérer d'importantes quantités de données et des situations par nature "anormales".

Enfin, [Turner 04] s'intéressent à la confiance et à son rôle dans les collectifs distribués, en particulier ceux ne pouvant passer par une médiation supervisée, en raison de leur taille ; l'application est encore le cas du la "conception continue" telle qu'elle a lieu dans les communautés *Open Source*.

En complément des points déjà abordé en 8.1.1 à propos de [Champin 02], qui est un des travaux fondateurs du modèle Musette, nous allons comparer les principes de notre approche (représentation symbolique et acquisition de sens par négociation) et les spécificités de notre domaine d'application (un environnement documentaire) par rapport aux siens (manipulation directe des observations de trace avec des signatures prédéfinies dans un environnement de CAO) ; ceci est fait dans la mesure où notre travail et le sien sont chacuns confrontés, dans leur application respective, à des situations extrêmes au regard des problématiques fondamentales de gestion et de manipulation d'expérience. Ce travail adopte une modélisation de la trace sous la forme d'une suite alternée d'états et de transitions, qui sera reprise dans le modèle Musette. Le point

[2]système de gestion de problèmes du projet Mozilla (http ://www.bugzilla.org/)

fondamental qui a distingué nos deux travaux vient du domaine d'application :

- [Champin 02] avait à observer la création d'un objet complexe (un modèle de conception, de nature tridimensionnelle) obtenu avec les centaines d'opérations de construction que CATIA met à disposition. Un modèle de graphe étiqueté est adopté pour représenter une observation. L'application des mécanisme de RàPC se concentrent sur l'interprétation et le rapprochement de la situation courante avec des situations antérieures. Nous pouvons parler d'interprétation, il est apparu le besoin de tenir compte d'un rôle commun à deux éléments distincts d'un modèle de conception qui doit être comparable au rôle d'un unique élément dans un autre modèle de conception ; cette abstraction est cherchée de façon automatique en dotant l'algorithme d'appariement de graphe d'une fonction de fusion de sommets d'un graphe pour être apparié à un unique sommet d'un autre graphe. Il est avancé le besoin de tenir compte des interactions de l'utilisateur pour diminuer la complexité de l'opération d'appariement et pour légitimer l'"interprétation" que représente une fusion. Dans cette perspective, les connaissances d'interprétation, comme la fusion, pourraient être médiées par une reformulation symbolique explicite et reflexive à l'utilisateur, qui demeure le plus à même d'indiquer les logiques de rapprochement qu'il convient d'effectuer ;

- à l'inverse, dans notre cas, nous observons des objets relativement simples, des documents naturellement décomposés en arbres étiquetés (les champs d'un document) comportant des mots clés (les OI). De plus, les captures d'OI dans un container d'arbre d'observation et les containers n'ont a priori à vérifier aucune relation entre eux. Même si une observation est stockée sous la forme d'un graphe quand on inclut les OI (c.f., figure 6.2), nous avons adopté sa lecture lors de l'appariement sous la forme d'un arbre étiqueté afin de réduire la complexité de cette opération. Pour dépasser la comparaison qu'offre les techniques de clustering documentaire et pour donner à l'acteur le moyen d'exprimer des modalités de rapprochement (l'équivalent d'une fusion), nous avons donné à l'utilisateur le moyen d'exprimer une abstraction ; une *variable* dans un pattern traduit cette notion, qui est dépendante du modèle de domaine. La pauvreté de la description d'une observation conduit à devoir caractériser une situation d'assistance au moins par un ensemble d'opérations réparties dans la trace. Notre grammaire formelle est proposée pour faciliter ce balisage des opérations dans la trace, en introduisant un modèle d'application. Cet ajout nous permet une reformulation symbolique de la trace, avec laquelle l'utilisateur peut interagir.

Une forte limite dans ces deux applications vient du caractère polymorphe des situations à identifier. L'introduction de la fusion dans l'appariement de graphe est une technique pour réduire cet écart ; dans notre cas, nous avons exclu dans un premier temps la prise en compte de relations/contraintes inter et intra symboles, mais le besoin apparaît clairement de les ajouter pour augmenter l'expressivité des signatures. Toute solution devra tenir compte de ce polymorphisme.

[Georgeon 06] utilise le principe de description à partir de trace en vue de l'affinement de la modélisation cognitive du conducteur de véhicule. L'acquisition de trace est faite par le biais de capteurs sur les différentes commandes du véhicule (rotation de volant, force de freinage..) et donne une trace temporelle de granularité très fine. Pour faciliter la lecture de cette trace, une reformulation symbolique est utilisée en s'appuyant sur des modèles de transformation et de visualisation. Ce travail est à rapprocher de [Sanderson 94], qui détaillent les études et l'expérience acquise dans l'acquisition, sous forme de données séquentielles, d'observations des interactions homme-machine, en vue de leur analyse exploratoire.

Le niveau syntaxique pour la représentation de l'expérience (c.f., 6.3.1) a des principes similaires à ceux présentés par [Gorodetski 02] pour des système de détection d'intrusion (IDS). Une structure grammaticale générale est proposée pour représenter une attaque, en la décomposant en plusieurs phases. Dans la pratique, un acteur humain mène une attaque pas à pas en tenant compte des réactions du système cible ; la structure grammaticale permet de représenter cette modularité. Une ontologie du domaine définit les éléments descriptifs qui permettent de construire une "phrase" d'attaque. La réponse de l'IDS est basée sur la reconnaissance d'une phrase.

Dans le même domaine d'application, avec une approche basée sur le RàPC, [Martín 04] présentent un mécanisme ascendant d'élaboration de cas d'attaque (sur la base de schémas d'attaque connus) appliqué au flux de connexions sur une machine ; ce flux est naturellement présenté sous forme de séquence. Ce mécanisme d'élaboration de cas a des similarités avec le mécanisme d'identification de signature ; pour des raisons d'expressivité, comme le fait [Georgeon 06], la possibilité est donnée de composer les éléments d'élaboration de cas entre eux pour exprimer une autre élaboration.

Ces deux derniers travaux s'intéressent à une situation différente de la nôtre : ils ont à faire à une situation d'adversité (la personne observée est pas nature adverse au système), ce qui rajoute une difficulté à l'"interprétation" dans la mesure où l'attaquant va chercher à trouver les failles de ce mécanisme et à agir en conséquent. Dans notre cas, la personne observée cherche à doter le système de connaissances supplémentaires pour mieux accompagner sa tâche ; par son appropriation, l'utilisateur tient compte des limites du système, et à intérêt à simplifier la tâche de l'agent alter ego.

Dans le contexte de l'alignement d'ontologies entre agents, [van Diggelen 05] présentent différentes stratégies de renforcement utilisées pour construire un vocabulaire de communication partagé. Ces stratégies peuvent être appliquées aux mécanismes de renforcement présentés en 7.5, et plus directement, à ceux que nous discuterons en 10.2 pour le partage d'expérience collective.

Chapitre 9

Démonstrateur

Sommaire

Ce chapitre présente les démonstrateurs d'application cible et d'agent alter ego que nous avons développés afin d'illustrer la mise en œuvre et confronter à l'expérimentation les principes d'assistance par RàPET et d'acquisition de sens d'interprétation par négociation, qui ont été détaillés les chapitres précédents. Ces démonstrateurs ont été programmés en JAVA. Nous allons les détailler en les mettant en relation avec les diagrammes de classes UML, qui figurent en annexe B. Pour manipuler les différentes descriptions RDF, que nous avons introduites, nous utilisons les fonctions de parsing RDF de JENA[1]. Notre schéma ontologique a été réalisé avec PROTÉGÉ[2].

L'application cible et l'agent alter ego sont intégrés dans une unique application, principalement pour éviter tous les problèmes d'interopérabilité, abordés en 4.2 via le travail de [Lieberman 98], et aussi pour faciliter le traçage fin des interactions. En effet, l'acquisition de trace se place à la frontière entre l'agent alter ego et l'application cible : la capture d'opérations significatives et des OI associés nécessite de spécifier l'application cible, car les événements de capture proviennent de cette dernière, mais la gestion et la traduction de ces captures en trace est assurée par l'agent alter ego.

[1] JENA : A Semantic Web Framework for Java – *http ://jena.sourceforge.net/*
[2] PROTÉGÉ – *http ://protege.stanford.edu/*

La collaboration avec *PCO Technologies* a permis d'identifier des besoins génériques aux applications "métier" qui nous ont conduits à développer une application cible analogue mais minimale, pour laquelle nous avons appliqué les principes de décomposition d'activité présentés en 2.2.3. Notre démonstrateur doit être considéré comme caractéristique de ce type d'applications métier, mais naturellement ces dernières disposent de fonctionnalités spécifiques. Nous nous intéressons uniquement aux fonctions de bases nécessaires pour un environnement documentaire.

9.1 Application jouet cible

Pour notre illustration, l'environnement visé est constitué d'un logiciel utilisé par un acteur, lui permettant de créer et d'accéder à des documents en rapport avec une activité considérée. Les documents sont les objets supports des différentes étapes de cette activité ; ils rassemblent les informations utiles à l'utilisateur pour effectuer les tâches liées à l'activité.

Dans l'application jouet, nous avons retenu comme activité la *procédure administrative* de gestion d'*ordre de mission*, telle qu'effectuée dans un laboratoire de recherche de l'université, et permettant la prise en charge par le laboratoire des frais de missions de ses membres. Cette activité est une activité coopérative complexe : les acteurs (le missionnaire, le directeur de laboratoire, le secrétariat, la comptabilité) n'ont qu'une vision partielle du processus, ils n'ont pas les mêmes rôles et ne disposent pas de toutes les connaissances pour réaliser leur tâche ; la tâche d'un acteur est donc menée en fonction du résultat de la tâche d'autres acteurs.

Dans la pratique, un dossier d'établissement d'*ordre de mission* est basé sur deux formulaires, dont les différentes parties sont remplies successivement, dans un ordre théorique imposé par la réglementation de la procédure. Toutefois, les contraintes réelles font que cet ordre ne peut pas toujours être respecté, ce qui entraîne des tâches particulières à réaliser par les acteurs de la procédure. Ce sont précisément ces situations exceptionnelles d'expérience dont nous visons la capture.

9.1.1 Activité et documents

Lors de notre conception d'un environnement informatique pour supporter cette activité collective, nous avons adopté le principe descriptif proposé par [Troussier 99] en considérant les parties de formulaire, qui sont successivement remplies par un acteur, comme des entités dans SG3C. Notre utilisation des principes de SG3C se limite dans un premier temps à cela. Dans SG3C, à des fins de gestion des connaissances dans le cadre de l'activité de conception intégrée, il est imposé aux acteurs d'expliciter chaque entité utilisée et produite lors d'une simulation, afin de construire un *cycle* de simulation. Cette structuration en cycles, spécifique à la simulation, est motivée par l'analyse des méthodes de simulation et elle a en soi un statut d'objet de connaissance métier. Dans notre cas, nous devons permettre aux acteurs de remplir dans un ordre indifférent les diverses parties des formulaires administratifs, qui constituent les entités appelées, dans l'ordre de leur exécution normale : 1) l'*Ordre de mission*, 2) l'*Acceptation de la Direction*, 3) l'*Enregistrement administratif*, 4) la réalisation effective de la *Mission*, 5) l'*Etat de frais*, 6) le déclenchement du *Remboursement* et 7) la réalisation effective du *Virement*. La notion de cycle, introduite par [Troussier 99], n'est pas utilisée, mais elle sera remplacée à terme par la trace collective, obtenue en concaténant les parties de trace individuelle qui correspondent aux diverses actions réalisées sur un formulaire. Nous reviendrons en 10.2 sur les problématiques de description d'une activité et de réutilisation des traces collaboratives.

Les entités documentaires introduites pour représenter l'activité d'établissement d'ordre de mission sont chacune constituée d'un ensemble de champs contenant du texte. Pour simplifier et expliciter l'acquisition, nous représentons le contenu d'un champs par un ensemble de mots clés, définis dans un modèle de domaine avec une organisation des mots-clés en hiérarchie de concepts, présentée en 6.1.

Nous avons introduit en 6.1 une structure pour organiser ces mots clés en une hiérarchie de concepts. Ce modèle nous permet de disposer d'une mesure de similarité entre mots clés et entre concepts. L'utilisateur pourra exprimer une abstraction en désignant un des concepts parents comme significatif dans un symbole.

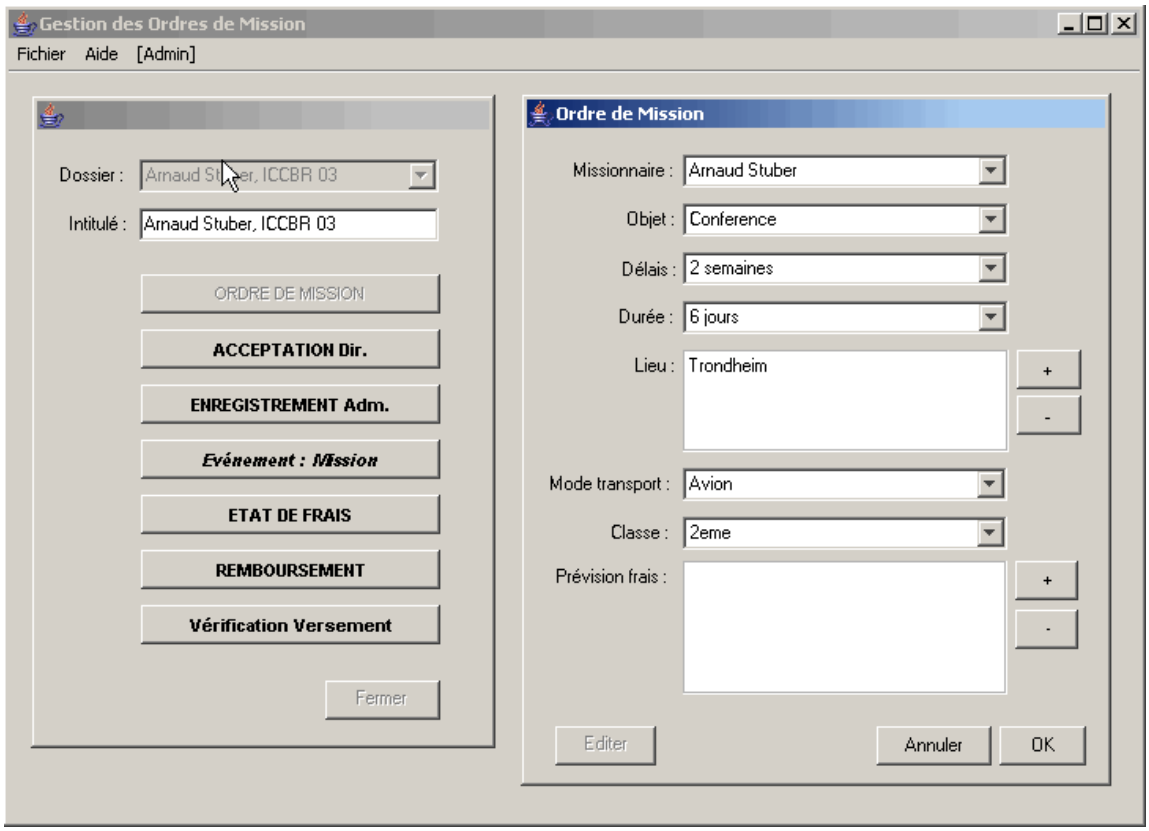


FIG. 9.1 – Application jouet cible

9.1.2 Fonctionnalités de l'application cible

Pour l'application cible que nous avons développée, les actions que peut faire un acteur sur les entités documentaires sont : l'*édition* et la *consultation*.

Bien qu'il y ait un intérêt évident pour l'acteur à disposer d'un module de recherche utilisant les techniques d'indexation et de clustering documentaire (à base des mesures *TFIDF* et *cosine*), nous considérons que son traitement pour exprimer une situation d'assistance et donc son traçage doivent faire l'objet d'une étude particulière pour définir l'élaboration et les actions de l'agent alter ego. Nous nous concentrons sur la réutilisation de l'expérience dans le contexte de la création ou de la modification de documents.

Notre approche permet à l'utilisateur d'effectuer une recherche d'épisodes antérieures similaires à ceux construits sur la trace en cours, via la possibilité de créer directement une nouvelle

signature (c.f., figure 9.2). La négociation du sens des symboles, crées par l'utilisateur lors de la définition de cette nouvelle signature, donne alors à l'agent alter ego un rôle d'assistant à la formulation de la requête de recherche.

L'établissement des fonctionnalités de notre démonstrateur d'application cible nous permet de définir le *modèle d'application*. Une description RDF est donnée en A.1, elle définit les instances des différents classes de *marqueurs* (*tag*) que nous avons introduits dans notre instantiation et extension de l'ontologie MUSETTE (c.f., figure 6.11).

En 6.3.1, nous avons argumenté le besoin de laisser à l'acteur la possibilité d'effectuer plusieurs actions conjointement sur différentes entités documentaires. Le marqueur *gainFocus* est utilisé pour ajouter une séquence $T \cdot E$ lors du passage d'une action à une autre (capturé par le changement de focus lors du passage entre deux fenêtres internes de détail d'entité).

La figure 9.1 donne une illustration de l'application cible, développée pour supporter l'activité d'établissement d'ordre de mission. Dans la fenêtre interne de gauche, nous avons une séquence de boutons décrivant les étapes de la procédure administrative, qui doivent normalement s'effectuer dans l'ordre indiqué. Pour chaque étape, une partie de formulaire papier (une entité documentaire) est associée. Un acteur peut accéder à un de ces formulaires en cliquant sur le bouton associé. A titre d'exemple, pour la première étape correspondant à la définition de la mission, le formulaire associé est illustré par la fenêtre interne de droite. Si une entité documentaire est simplement visualisée, l'acteur peut passer à son edition en utilisant le bouton "Editer", en bas à gauche dans la fenêtre interne de droite.

9.2 Assistant

9.2.1 Architecture de l'*Agent alter ego*

La définition de l'architecture du corps de l'agent alter ego part de la nécessité de prendre en compte le caractère asynchrone des mécanismes d'acquisition et de RàPET lorsque l'utilisateur interagit avec l'application cible, c.f., figure 5.1. Cela nous a conduit à adopter une architecture *multi-thread* pour le corps de l'agent alter ego. Le diagramme de classes de la figure B.1 définit ces éléments, l'héritage de la classe *Thread* n'est pas explicité :

- la classe thread *AgentAlterEgo* a pour rôle de coordonner les trois comportements de l'agent alter ego, tels qu'ils ont été décrits en 5.2. Elle initialise une session d'utilisation en créant une nouvelle trace afin de construire la trace en cours. Elle dispose de la classe *CommonOntologies* qui permet d'utiliser notre ontologie et d'accéder à l'état de langage en cours. Elle initialize et démarre les classes threads *BlackBoardThread*, *SymbolRecognitionManager*, *SignatureRecognitionManager* et *EpisodeRetrievalManager* détaillées ci-dessous ;

- la thread *BlackBoardThread* est chargée de coordonner la réalisation des deux premières étapes du cycle de RàPET. Elle stocke les résultats intermédiaire du processus d'interprétation et déclenche la recherche d'épisodes antérieurs. Elle reçoit les sections de trace du module d'acquisition et déclenche la recherche d'appariements de symboles en transmettant ces sections à la thread *SymbolRecognitionManager*. Elle reçoit en retour les divers appariements trouvés et les signale à la thread d'identification de signature qu'est *SignatureRecognitionManager*. Lorsqu'une signature est identifiée et lorsqu'une identification est complétée, elle est renvoyée à *BlackBoardThread*, qui déclenche alors la recherche d'épisode

en transmettant l'identification de signature à la thread *EpisodeRetrievalManager* ;

- la thread *SymbolRecognitionManager* reçoit un appel d'appariement de symboles de *BlackBoardThread* pour une section de la trace en cours. Elle interroge l'unique instance de la classe *SymbolTraceRepository*, qui assure l'interface avec la base de données SQL chargée d'enregistrer et d'indexer les traces et les symboles de langage avec les références aux fichiers contenant leurs descriptions en RDF, pour obtenir une liste de symboles candidats à l'appariement. Pour chaque symbole candidat, *SymbolRecognitionManager* déclenche une recherche d'appariement sur la section de trace en cours, via la création d'une *SymbolRecognitionThread*. Les appariements trouvés pour un symbole sont renvoyés à *BlackBoardThread* ;

- une thread *SymbolRecognitionThread* est chargée de trouver tous les appariements d'un symbole sur une section de trace. La classe *TreeMatching* est utilisée pour réaliser l'appariement d'un pattern sur un arbre d'observation. La forme commune $E \cdot T \cdot E$ implique trois appariements et les résultats de chaque appariement sont combinés de toute les manières entre eux afin de produire tous les appariements possibles. Ils sont renvoyés à *SymbolRecognitionManager* ;

- la thread *SignatureRecognitionManager* effectue toutes les identifications de signature. Lors de la réception d'un appariement de symbole, elle consulte l'état de langage en cours contenu dans *CommonOntologies* pour connaître les signatures candidate. Une liste enregistre la progression de l'identification des signatures candidates. Lorsque le taux d'identification d'une signature dépasse le seuil τ_{idSign}, l'identification de la signature est renvoyée à *BlackBoardThread* ;

- lors du signalement par *BlackBoardThread* d'une nouvelle identification de signature, la thread *EpisodeRetrievalManager* interroge *SymbolTraceRepository* pour avoir une liste de traces antérieures candidates à la recherche d'épisode. Avec cette liste et l'identification de la signature sur la trace en cours, elle lance une *EpisodeRetrievalThread*.
  Lors du signalement par *BlackBoardThread* d'un complément dans l'identification d'une signature, la thread *EpisodeRetrievalThread* associée est avertie ;

- une thread *EpisodeRetrievalThread* est chargée de la recherche d'épisodes dans les traces candidate et du calcul de la similarité d'un épisode antérieur avec l'épisode en cours au regard de l'identification de signature. Lors d'un complément d'identification notifié par *EpisodeRetrievalManager*, elle doit mettre à jour les épisodes déjà identifiés.

Nous reviendrons en 9.2.4 sur la réalisation du cycle de RàPET et nous présenterons plus en détail les objets manipulés durant ce cycle par les threads introduites ci-dessus.

9.2.2 Acquisition de trace

Le diagramme de classe de la figure B.2, page 223, définit schématiquement les éléments du package *java.Swing* utilisés pour concevoir les interfaces graphiques de l'application cible et de l'agent alter ego ; ces derniers sont intégrés en une seule application. Nous résumons ces éléments *Swing* par :

- la fenêtre de l'application intégrée, *RootFrame*, qui hérite de *java.Swing.JFrame* ;

- les fenêtres internes, désignées par *InternalFrameXYZ*, qui héritent de *java.Swing.JInternalFrame*. Elles sont utilisées pour afficher les dossiers et les entités documentaires, c.f., figure 9.1. Les interfaces graphiques de l'agent alter ego (*InterpretationGUI* et *ExperienceGUI*), présentées en 9.2.4, sont construites de la même manière ;

- les fenêtres internes sont composés de *java.SwingObject*. C'est également le cas des interfaces de négociation de sens (*SymbolNegociationGUI* et *SignatureNegociationGUI*), présentées en 9.2.5, à la différence qu'elles héritent de *java.Swing.JDialog*.

Nous avons détaillé cela car le module d'acquisition de trace se situe à la frontière avec l'application. Ce module de l'agent alter ego est chargé de collecter les observations provenant de l'application et, en incluant le modèle d'application, de construire une séquence (de la forme $T \cdot E$) qui vient compléter la trace en cours. L'acquisition d'une nouvelle séquence est déclenchée par la capture d'un événement envoyé par un *SwingObject*. Une capture, sous forme d'arbre d'observation, de l'état de la fenêtre interne contenant cet objet est effectuée pour rendre compte des modifications de l'application cible après cette opération de l'utilisateur. Les paramètres de l'opération sont représentés sous la forme d'arbre d'observation, et constitue avec la capture d'état les deux arbres d'observation de la séquence.

9.2.3 Le cycle de RàPET

En 9.2.1, nous avons décrit l'enchaînement d'opérations effectuées, par les différentes threads composant le corps de l'agent alter ego, afin de réaliser les deux premières étapes du cycle de RàPET, l'*élaboration/interprétation* et la *remémoration*, selon les principes définis au chapitre précédent. Les deux étapes suivantes, i.e., la *réutilisation* et la *réparation*, nécessitent l'acquisition et la gestion de connaissances propres ; nous reviendrons sur cette problématique en perspectives (c.f., 10.5).

Pour la dernière étape, la *rétention* de trace, nous choisissons de procéder à un stockage systématique des traces à la fin de chaque session de l'utilisateur. Nous n'avons pas traité la problématique liée à cette étape. Pour sélectionner a priori les traces antérieures à explorer lors de la recherche d'épisodes, une indexation des traces a été utilisée, elle est faite sur la base des types de d'entités documentaires qui sont manipulées dans une trace.

L'interprétation de la trace en cours est réalisée pour un certain état de langage contenu dans l'instance de *CommonOntologies* dont dispose l'*AgentAlterEgo* ; les classes permettant de construire et d'interroger un état de langage sont introduites dans le diagramme de classes de la figure B.3. Le diagramme de classes de la figure B.4 définit les objets utilisés pour représenter une interprétation et le rapprochement de la trace en cours avec une trace antérieure, nous le résumons par :

- la classe *Trace* contient le model RDF d'une trace, les fonctions pour sa lecture et les fonctions nécessaires à la description de la trace en cours lors de l'acquisition. La classe *Section* représente le concept du même nom introduit au chapitre précédent, elle est constituée des références positionnant une fenêtre de lecture, de la forme $E \cdot T \cdot E$, sur une trace ;

- l'appariement d'un symbole, dont le modèle RDF est contenu dans une instance de *Sym-*

bol, est représenté par une instance de *SymbolMatch*. Cette instance est construite par une thread *SymbolRecognitionThread*, via l'appel à *TreeMatching* pour l'appariement des patterns de *Symbol* avec les arbres d'observations d'une *Section* ; elle est composée de trois listes de *Pair* ;

- lors de la transmission d'un *SymbolMatch* à la thread *SignatureRecognitionThread*, celle-ci vérifie si le symbole reconnu initie ou complète l'identification d'une signature. L'état d'identification d'une *Signature* dans une trace est représenté par une instance de *SignatureMatch*. Elle contient les symboles reconnus, des *SymbolMatch*, et les références des symboles restant à identifier. Lorsqu'une signature a un taux d'identification est supérieur à τ_{idSign}, la reconnaissance contenue dans *SignatureMatch* est transmise à *BlackBoardThread* pour recherche d'épisode ;

- lorsque *BlackBoardThread* notifie l'identification d'une nouvelle signature à *EpisodeRetrievalManager* via la transmission d'un *SignatureMatch*, *EpisodeRetrievalManager* crée une *EpisodeRetrievalThread*, qui utilise un *EpisodeMatch* pour stocker le *SignatureMatch* de la trace en cours et les épisodes trouvés dans les traces candidates antérieures représentés via un ensemble de *SignatureMatch*. Lors d'un complément d'identification est trouvé pour une signature, *EpisodeRetrievalManager* avertit l'*EpisodeRetrievalThread* associée, qui est chargé de mettre à jour les rapprochements avec des épisodes antérieurs en tenant compte de ce complément.
  Le calcul de similarité sémantique entre termes du modèle de domaine d'application est réalisé avec la classe *Similarity*, elle est utilisée pour calculer la similarité entre l'épisode en cours et un épisode antérieur.

9.2.4 Interfaces de proposition d'interprétation et de suggestion d'épisodes

Dans notre présentation du comportement de suggestion de l'agent alter ego, c.f., 5.2, nous avons introduit la présence d'une interface sur la figure 5.2 servant de support à ce comportement. Notre travail a pour but de définir les principes des éléments d'interface, qui lui sont nécessaires, et de les illustrer simplement, en utilisant des objets graphiques courants. Ainsi, ces interfaces sont à mi-chemin entre une interface utilisateur et une interface pour le "degugging".

Nous devons donner à l'acteur humain la possibilité de consulter et de comparer les interprétations déjà effectuée par l'agent alter ego, simplement pour suivre la progression de l'interprétation et retourner à sa tâche ou pour en sélectionner une et accéder aux épisodes rapprochés. Pour cela nous introduisons deux niveaux qui structurent et explicitent les interactions entre l'agent alter ego et l'utilisateur :

1. une interface d'accès aux interprétations effectuées sur la trace en cours, en fonction d'un état de langage. Cette interface ne déclenche aucune modification du langage ;

2. une interface d'accès aux épisodes pour une signature et d'acquisition de réactions de l'utilisateur, démarrant une session d'assistance et pouvant appeler le comportement de négociation lorsque l'utilisateur demande la mise à jour des propositions. Cela conduit à la construction de nouveaux états de langage.

Pour décrire notre implémentation de ces deux niveaux, nous le faisons du point de vue de l'utilisateur, ou plus exactement, de la perception qu'il doit avoir pour interagir avec son agent alter ego :

1. après une série d'opérations effectuées en interaction avec l'application cible, l'utilisateur souhaite consulter les interprétations déjà effectuées. A ce niveau, l'interface doit donc lui permettre de consulter les différentes interprétations de la trace en cours, elles sont présentées sous forme d'une liste faite avec les labels des signatures associées. Le détail de l'interprétation sur la trace en cours d'une signature sélectionnée dans cette liste est présenté à l'utilisateur en adoptant un modèle graphique pour la représentation d'une trace symbolique. Notre modèle graphique est fait d'un enchaînement de boîtes colorées allant de gauche à droite, cela est fait pour traduire la temporalité de la trace ; chaque boîte représente un symbole et contient le début du label de ce symbole. Nous choisissons de mettre en évidence les symboles de langage qui ont été identifiés, afin de les distinguer des symboles génériques. La figure 9.2 illustre une telle interface, où figurent de haut en bas la liste des interprétations de la trace en cours et le détail affiché sous forme symbolique pour l'interprétation sélectionnée. Les boîtes encadrées avec leurs labels en gras mettent en évidence les symboles de langage. Dans la liste des interprétations, le chiffre entre parenthèses derrière le label d'une signature Z correspond au taux d'identification de cette signature ($\tau_Z(trace\ en\ cours)$, c.f., équation 6.8).

   Si l'utilisateur juge insuffisantes les interprétations proposées et souhaite créer un nouveau sens d'interprétation, il dispose pour cela de l'item [*Nouvelle Signature*]. En double-cliquant sur celui là ou sur un autre, il initie une session d'assistance pour la signature visée.

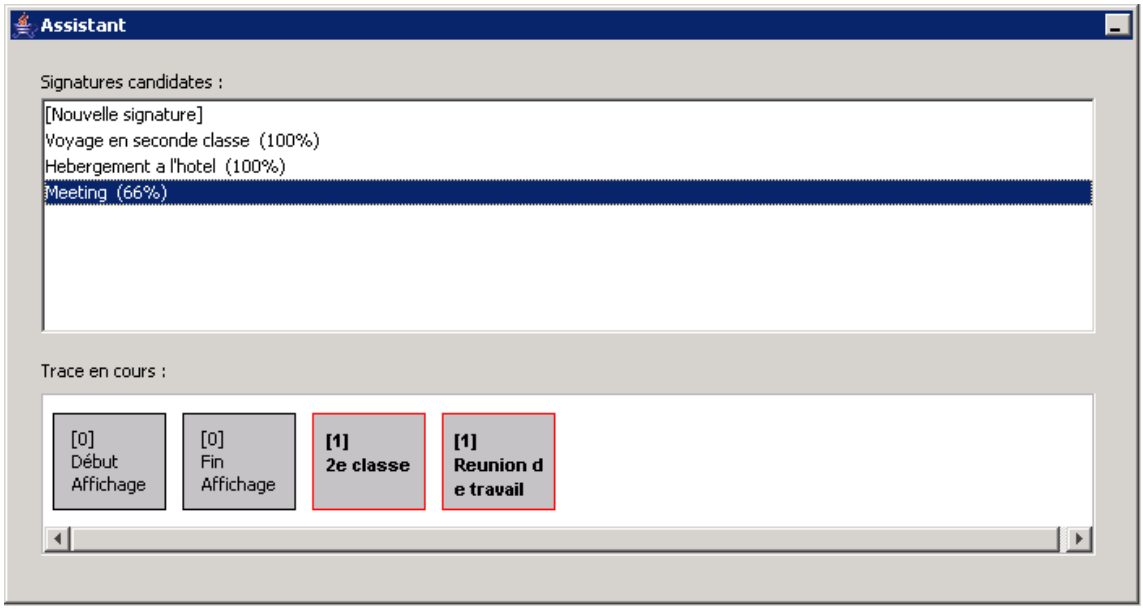


FIG. 9.2 – Interface d'accès aux interprétations

2. en sélectionnant une signature, l'utilisateur peut simplement souhaiter consulter les épisodes rapprochés sans intervenir sur l'interprétation, ou bien s'être tout simplement trompé de signature en double-cliquant. Dans ces cas, l'interface concernée va apparaître, et l'uti-

lisateur n'effectuera qu'une seule opération, sa clôture. Cette situation est le cas limite d'une session d'assistance, qui se termine lors de cette clôture. A ce moment, l'agent alter ego doit résoudre l'ambiguïté entre ces deux situations ; une boite de dialogue permet à l'utilisateur d'expliciter cela en :

- acceptant/rejetant l'interprétation. Cela implique le succès/échec des itérations de jeux de langage et la création d'un nouvel état de langage ;
- annulant cette session d'assistance, dans le cas d'une erreur tout comme pour laisser l'agent alter ego continuer son interprétation. Cela n'implique aucune modification du langage.

Après le choix d'une interprétation, notre seconde interface doit permettre à l'utilisateur de consulter les épisodes d'expérience qui ont été rapprochés, et lui donner le moyen de créer ou de modifier des symboles pour affiner l'interprétation. Les interventions de l'utilisateur au cours d'une tentative d'assistance sont mémorisées, nous l'avons représenté sur la figure 5.2 par la base intitulée *Réactions*. Ces informations seront les objets cibles du processus de négociation. La figure 9.3 illustre l'interface que nous avons conçue pour supporter cette consultation de l'expérience remémorée.

Le principe de cette interface est d'afficher simultanément le détail de l'interprétation sur la trace en cours et dans une trace antérieure, de laisser l'utilisateur intervenir sur les sections de ces traces pour créer ou affiner un symbole. Ainsi, dans notre interface, nous avons de haut en bas :

- le label de la signature Z considérée ("Meeting") ;

- la trace en cours sous sa forme symbolique, avec le même modèle graphique que dans l'interface précédente, mettant en évidence les symboles de la signature qui ont été identifiés ;

- la liste des épisodes candidats retrouvés, le label d'un épisode correspond au numéro d'identification de la trace dont il est issu. Les épisodes sont classés en ordre de similarité décroissant avec la trace en cours ($sim_Z(trc_{en\ cours}, trc_{anterieure})$, c.f., équation 6.10). Le chiffre entre parenthèses correspond à la valeur du produit $couverture \times représentativité$ (c.f., équations 6.12 et 6.13), cela est fait pour informer l'utilisateur de la "pertinence" de l'épisode pour Z ;

- le détail sous la forme icônique (juste une suite de symboles présentés avec leurs labels) de l'interprétation d'un épisode sélectionné dans la liste ;

- et tout en bas, deux zones permettant à l'utilisateur d'interagir avec le détail d'une section, qu'elle provienne de la trace en cours ou bien d'une trace antérieure.

La structure arborescente des trois observations, qui sont associées aux états et à la transition d'une section de trace (de forme $E \cdot T \cdot E$), nous conduit à utiliser l'objet graphique standard d'arbre (*JTree* dans notre cas) pour représenter le détail d'une section à l'utilisateur.

Les deux zones d'affichage de détail de section présentent dans l'interface fonctionnent de la même façon : elles sont composées d'un arbre donnant la définition d'une section. Ainsi, sur la zone de gauche, nous voyons cette représentation hiérarchique sous sa forme réduite : nous avons à la racine le marqueur associé à ce symbole ; puis successivement en tant que sous arbres : l'observation pour l'état précédant l'opération (dont l'affichage est

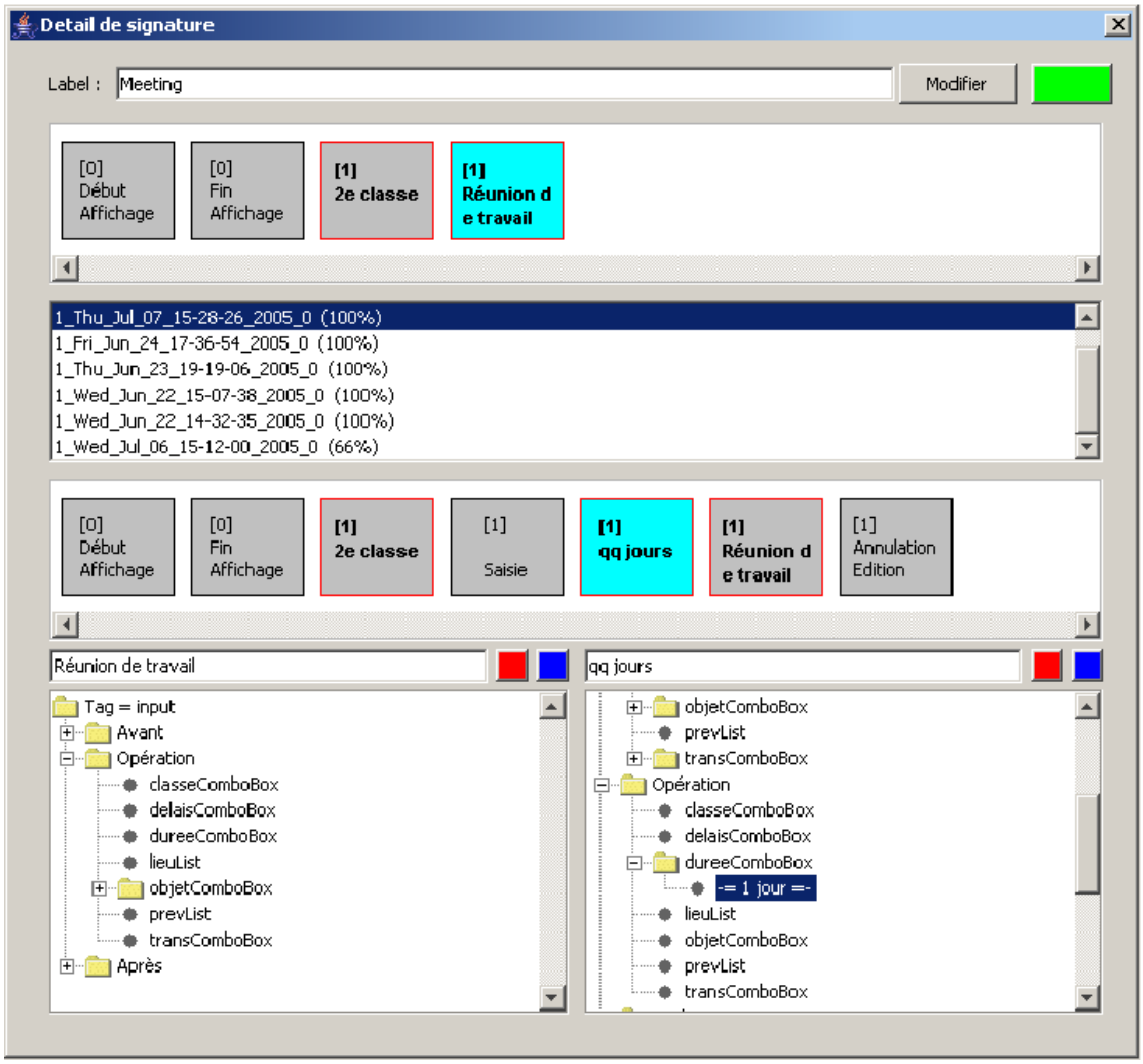


FIG. 9.3 – Interface d'accès à l'expérience pour une signature

réduit à la racine), l'observation associée à la transition (affichée sous sa forme développée), et l'observation pour l'état suivant l'opération (la racine).

Dans le cadre de l'appariement d'une section avec un symbole, il est mis en évidence l'appariement d'une capture d'un pattern de ce symbole avec une capture d'arbre d'observation de la section. Sur la zone de droite, nous avons développé le sous arbre correspondant à l'observation pour l'opération ; nous pouvons voir qu'un élément (ici, le mot clé "1 jour" placé dans le container muni du label "dureeComboBox") a été déclaré comme significatif. Cette déclaration est effectuée à l'aide des deux boutons situés en haut à droite de l'arbre descriptif, à coté du label du symbole considéré. Ces boutons permettent à l'acteur d'indiquer si un élément d'observation doit être présent dans l'absolu pour définir un symbole (il est alors représenté par le code -= mot =-) ou bien généralisé au concept qu'il représente (il est alors représenté par le code ** mot **), ce qui dépend du modèle de domaine utilisé.

9.2.5 Interfaces pour la négociation de sens

Après un certain nombre d'interactions, l'utilisateur demande à ce que les modifications qu'il a effectuées soient prises en compte par l'agent alter ego lors d'une nouvelle interprétation et la mise à jour des rapprochements. Cela initie le comportement de négociation de l'agent alter ego, c.f., figure 5.3, qui applique le mécanisme de négociation décrit par la figure 7.6, afin de résoudre les ambiguïtés de sens provenant des réactions de l'utilisateur. Le mécanisme de négociation introduit deux interfaces pour supporter 1) la négociation de symbole et 2) la négociation d'une nouvelle définition de la signature.

Nous résumons les principes des interfaces que nous proposons pour ces deux étapes par :

1. une première source d'ambiguïté vient des symboles créés ou modifiés ayant conduit à des symboles existant déjà dans le langage. Dans tous les cas, pour un symbole créé ou modifié, l'utilisateur doit vérifier son label et en choisir un commun pour les symboles identiques ; il doit également décider s'il souhaite créer un nouveau symbole ou bien le faire évoluer. Cette négociation de symbole s'applique à tous les symboles qui ont été manipulés par l'utilisateur. La figure 9.4 donne une illustration de notre interface permettant cette négociation de symbole.

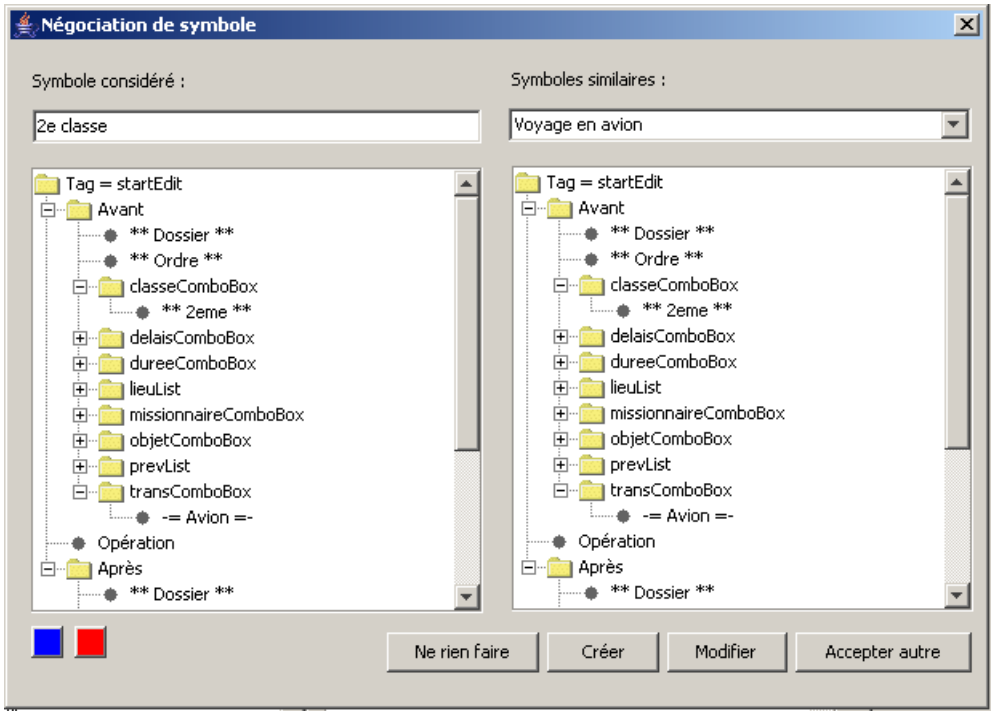


FIG. 9.4 – Interface de négociation de symbole

Dans la partie gauche, nous avons le symbole faisant l'objet de la négociation, dont le détail est représenté de la même façon que dans la partie inférieure de la figure 9.3 ; les boutons situés en dessous permettent encore à l'utilisateur de modifier la définition du symbole.

Dans la partie droite, les symboles de langage similaires au symbole négocié sont soumis à l'utilisateur, qui peut choisir de remplacer le symbole négocié par l'un d'entre eux. La similarité de symboles est donnée dans l'équation 6.5. Dans le cas où un symbole négocié

est identique à des symboles de langage existants, un label commun doit être donné à ces symboles et ils seront fusionnés dans le langage. L'utilisateur doit spécifier si une modification de symbole doit être considérée comme une évolution de ce dernier ou bien comme la création d'un nouveau symbole.

2. dès lors que les symboles ont été négociés, l'agent alter ego prend en compte les réactions de l'utilisateur et lui demande de valider la nouvelle définition symbolique de la signature, générée par la synthèse de la négociation de symbole, c.f., mécanisme de négociation de la figure 7.6. La figure 9.5 montre l'interface que nous avons développé pour assurer cette négociation.

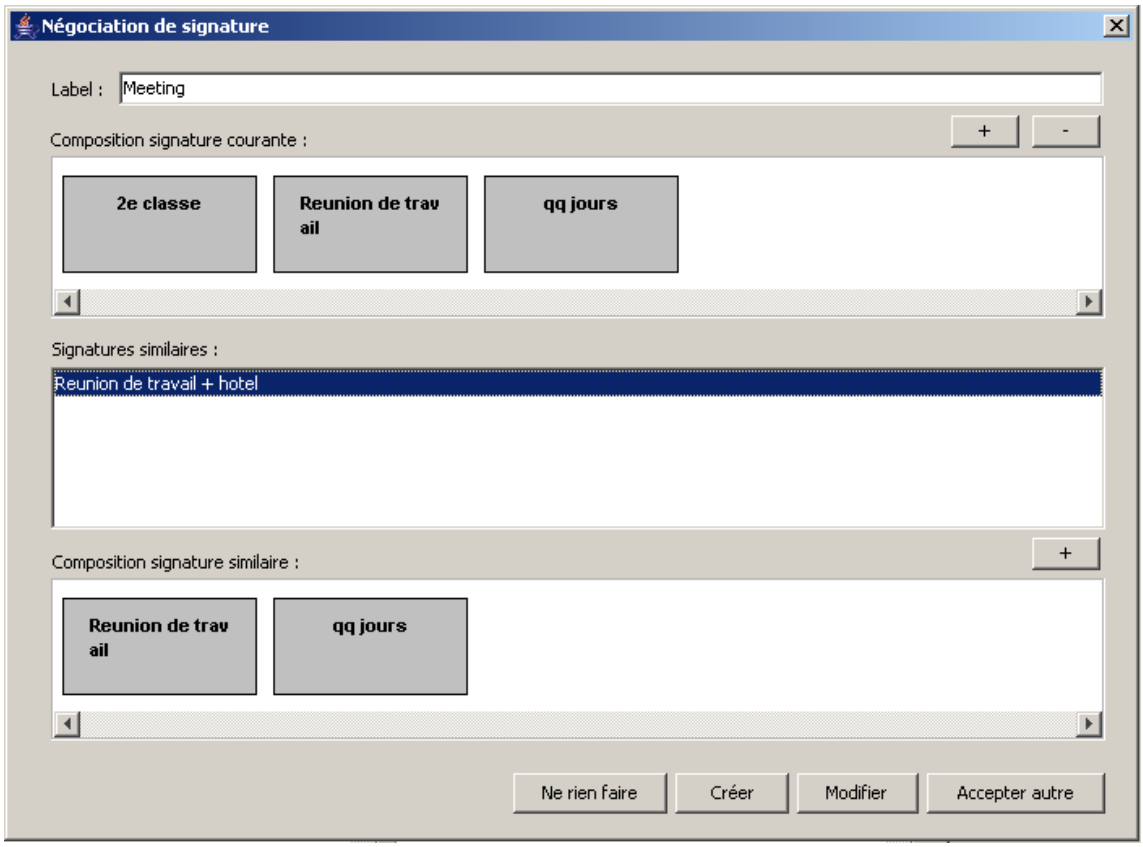


FIG. 9.5 – Interface de négociation de signature

Tout comme pour les symboles, les interactions peuvent conduire à des signatures similaires voire identiques. La similarité entre signatures est donnée par l'équation 6.6. Les signatures similaires sont soumises à l'utilisateur qui peut en accepter une. Il peut également décider de modifier la définition de la signature considérée, ou bien de créer une nouvelle signature. En cas d'égalité de signature, l'utilisateur doit leur choisir un label commun, et ces signatures seront fusionnées en une seule.

Après cette dernière étape de négociation, l'agent alter ego dispose des informations pour compléter les itérations de jeu de langage. Les fusions de symboles et de signatures sont effectuées, ainsi que les renforcements ou les inhibitions de symboles/signatures selon les principes définis en

7.5. Un nouvel état de langage est obtenu, à partir duquel l'agent alter ego effectue une nouvelle interprétation. Le résultat est soumis à l'utilisateur via l'interface de la figure 9.2.

9.3 Illustration sur un scenario d'usage

Afin d'illustrer l'usage d'un tel environnement, considérons l'acteur humain, membre du laboratoire, souhaitant effectuer une procédure d'*ordre de mission* (ODM). Pour cela, il initie une session dans un environnement à base documentaire dédié à cette tâche, dont notre application cible constitue une illustration "jouet".

En s'appuyant sur la figure 9.1, considérons que l'acteur humain (soit, "Arnaud STUBER") reprend une procédure d'ODM intitulée *Arnaud Stuber, ICCBR'03* qu'il a initiée dans une session antérieure. A ce stade de la procédure, il doit déclarer convenablement son futur déplacement ; pour cela, il édite la première entité documentaire, appelée *Ordre de mission*. Le détail de cette entité est donné dans la fenêtre interne de droite de la figure 9.1. Dans une session antérieure, l'utilisateur a déjà indiqué que cette mission est à son nom, qui se tiendra dans "2 semaines" à "Trondheim", pour une durée de "6 jours" et qu'il se déplacera en "Avion". Dans cette session, il indique successivement que cette mission est dans le cadre d'une "Conférence" et qu'il voyagera en "2ème classe"[3].

Arrivé à ce point, l'utilisateur souhaite vérifier si sa déclaration est complète, et il choisit de faire appel à son agent alter ego pour consulter des situations antérieures analogues. La figure 9.2 présente l'interface permettant à l'agent alter ego de proposer ses élaborations à l'utilisateur. Parmi les propositions d'élaboration, l'utilisateur est intéressé par celle intitulée *Meeting*, dont le taux d'identification est de 66%, et la sélectionne ; dans la partie inférieure de l'interface de la figure 9.2, l'utilisateur peut consulter la forme symbolique de l'épisode source qui a été identifié dans la trace en cours pour la signature de label *Meeting*.

Après le choix de cette signature par l'utilisateur, l'agent alter ego met à disposition les épisodes sources qu'il a trouvés, via l'interface de la figure 9.3. En consultant les épisodes proposés, l'utilisateur se rend compte qu'il a oublié d'effectuer la déclaration de frais. Il apprend également les éléments qui doivent y figurer dans le cadre d'un voyage en avion en 2ème classe (symbole de label *2e classe*), d'une durée de plusieurs jours (symbole de label *qq jours*) et afin de participer à une réunion de travail (symbole de label *Réunion de travail*), à savoir les frais de transport et d'hébergement. A ce stade, l'utilisateur peut estimer que cette aide basée sur ces élaborations est convenable, et mettre fin à cette session d'assistance en émettant un avis favorable. Nous serions alors dans le cas d'une session sans négociation suivant le scénario Sc_2, c.f., 7.3. Un nouvel état de langage serait créé en renforçant la signature, les symboles qui la définissent ainsi que les relations symbole-signature.

L'utilisateur peut également souhaiter réagir avec les éléments symboliques, afin d'affiner ou de réviser les connaissances d'élaboration. Par exemple, il peut être surpris par la reconnaissance du symbole de label *Réunion de travail* alors qu'il a indiqué qu'il se déplace pour une conférence. Après un certain nombre de réactions sur des éléments symboliques, l'utilisateur demande une mise à jour des propositions afin que ses réactions soient prises en compte. Le processus de négociation est initié ; pour le symbole de label *Réunion de travail*, l'utilisateur réalise qu'il contient en réalité une abstraction ayant permis sa perception malgré la présence du mot-clé "conférence", il décide de seulement revoir le label de ce symbole. Pour les autres réactions,

[3] nous avons mis entre guillemets les mots-clés qui relèvent du modèle de domaine, i.e., la terminologie de l'activité d'ODM

considérons que l'utilisateur crée de nouveaux symboles, par exemple, en détaillant les éléments devant figurer dans les frais de mission. Durant la négociation de signature, l'utilisateur accepte tous les symboles qu'il a ainsi créés et il estime que cela constitue une évolution de la signature de label *Meeting*. Nous sommes dans le cas de l'alternative d'enchaînement de tentatives A_1 pour le scénario Sc_3. Un nouvel état de langage est créé, à partir duquel la mise à jour des propositions est effectuée. L'utilisateur pourrait continuer à interagir, cela conduirait à un nouvel état de langage après chaque négociation. Dans cet exemple, nous considérons que l'utilisateur est satisfait de sa nouvelle définition pour la situation d'assistance intitulée Meeting ; il ferme l'interface de suggestion d'expérience en donnant un avis favorable. Un nouvel état de langage est créé en renforçant les éléments de langage impliqués dans cette nouvelle définition.

9.4 Premier retour d'expérience

9.4.1 Processus d'interprétation

La manipulation de notre démonstrateur d'agent alter ego a mis en évidence les limites qu'impliquent l'absence de ciblage de l'interprétation, introduit en 8.1.1, et l'impossibilité d'exprimer des contraintes inter et intra-symbole pour définir une signature. Cela ne permet pas de nuancer suffisamment le mécanisme glouton d'identification de signature, via la prise en compte du premier appariement d'un symbole, de telle sorte que notre démonstrateur a des chances de proposer une interprétation porteuse de sens que si elle se limite à l'action en cours. En effet, dans le cas d'une trace représentant des actions menées conjointement, notre mécanisme glouton d'interprétation de signature (présenté en 6.6.1.2) entraînerait facilement la prise en compte de perceptions portant sur des actions distinctes, ce qui a très peu de chance d'être porteur de sens pour l'utilisateur. Les points, que nous abordons ici, relèvent de la problématique d'une interprétation pertinente (conduisant à la construction d'un épisode) que nous avons présentée en 6.6.1.2.

Cependant, même dans ce cas mais aussi dans le cadre plus précis de l'interprétation d'une action, certaines situations viennent perturber l'interprétation ; de plus, notre modèle d'interprétation requiert de l'utilisateur une connaissance du fonctionnement interne de l'agent alter ego, afin de formuler les symboles de façon appropriée. Nous allons nous appuyer sur l'exemple d'une action d'édition d'un document, décrit par deux containers, qui sont remplis et modifiés suite à des opérations de saisie d'OI de l'utilisateur. La figure 9.6 donne un exemple de trace en cours pour cette action et une trace antérieure, respectant la grammaire formelle de description de trace.

Nous figurons les OI observés dans les arbres d'observation de ces deux traces sous des formes (triangle ou cercle, auxquels nous donnons le rôle de "concept") avec une certaine texture (dont le rôle est de représenter un "mot-clé"). Nous considérons que dans la trace en cours l'utilisateur hésite et fait des saisies qu'il corrige par la suite ; c'est également le cas dans la trace antérieure. Par "hésitation", nous entendons la situation suivante pour la trace en cours : après ses deux premières saisies, l'utilisateur a affecté un triangle (avec une certaine texture) dans le container du haut, et un cercle (avec une certaine texture) dans le container du bas. Après la quatrième saisie, nous avons toujours un triangle en haut et un cercle en bas, mais leurs textures ont changé. Nous désignons par ces variations de textures, une situation où l'utilisateur "cherche" ou bien "hésite" dans la détermination des "mots-clés" qui instancient le mieux un "concept". La même situation est présente dans la trace antérieure.

Dans l'absolu, le mécanisme d'interprétation ne doit pas être perturbé par les hésitations de l'utilisateur, qui constituent dans une certaine mesure une forme de "bruit", même si notre

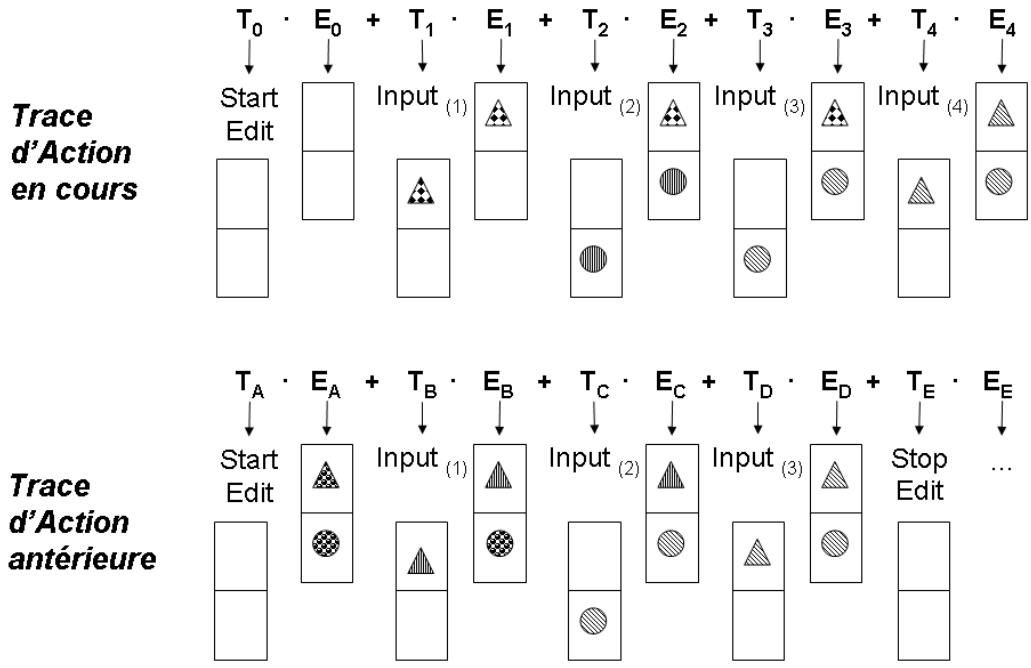


FIG. 9.6 – Trace de l'action en cours et trace d'action antérieure

approche n'exclue pas de réaliser une assistance particulière en cas d'hésitation.

Considérons une signature Z, définie par les symboles S_1 et S_2 décrits sur la figure 9.7, au regard de laquelle les deux traces de la figure 9.6 peuvent être rapprochées, si l'on considère que les formes triangle ou cercle sans texture dans un pattern expriment un généralisation qui autorise le rapprochement d'OI de même forme mais avec des textures différentes.

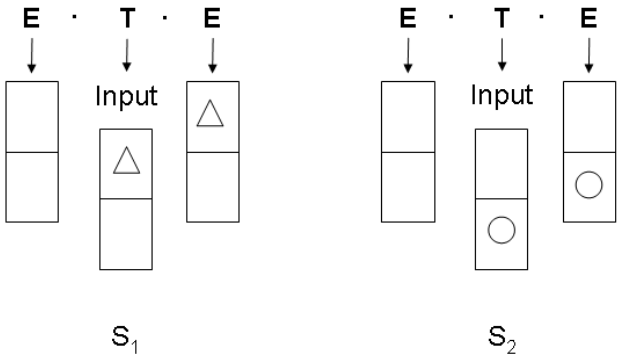


FIG. 9.7 – Les symboles S_1 et S_2 définissant une signature Z

Ces symboles ont été construits en adoptant la logique suivante : la présence d'un OI (ici une variable) dans l'arbre d'observation de la transition est fait pour pour exprimer qu'on s'intéresse

à l'opération de saisie d'une quelconque instance (texture indifférente) d'une certaine forme (triangle ou rond) à l'endroit que constitue le container. La présence d'une autre variable, de la même forme, dans l'arbre d'observation de l'état final, à la même position que pour celui de la transition, est faite pour exprimer que cette opération de saisie entraîne une modification d'état. Cela est fait pour tenter de filtrer les opérations d'hésitation, i.e. celles qui n'ont pas d'influence ; mais nous allons voir que cela n'est pas suffisant, et ce en raison du caractère "local" d'un motif de symbole.

L'application de notre mécanisme glouton d'interprétation de la trace en cours et de la trace antérieure au regard de Z fait que ce sont les appariements suivants qui ont été considérés :
- S_1 sur la section $E_0 \cdot T_1 \cdot E_1$ et S_2 sur la section $E_1 \cdot T_2 \cdot E_2$, pour la trace en cours ;
- S_1 sur la section $E_A \cdot T_B \cdot E_B$ et S_2 sur la section $E_B \cdot T_C \cdot E_C$, pour la trace antérieure.

Nous voyons dans le détail des trace de la figure 9.6 que l'identification de Z devrait plutôt être :
- S_1 sur la section $E_3 \cdot T_4 \cdot E_4$ et S_2 sur la section $E_2 \cdot T_3 \cdot E_3$, pour la trace en cours ;
- S_1 sur la section $E_C \cdot T_D \cdot E_D$ et S_2 sur la section $E_B \cdot T_C \cdot E_C$, pour la trace antérieure.

Si l'appariement de S_2 sur la trace antérieure est correct, cela est simplement dû à l'absence de "bruit" que constituerait la présence avant cette section d'une section représentant une "hésitation".

Un autre inconvénient de notre mécanisme d'interprétation s'illustre sur l'absence de remise en cause du rapprochement de $E_0 \cdot T_1 \cdot E_1$ avec $E_A \cdot T_B \cdot E_B$ pour S_1 alors que la deuxième version de rapprochement permet de comparer des sections identiques, donc de meilleure similarité. Cet exemple illustre le besoin de mettre en place un processus de révision de l'interprétation, qui permette la recherche d'une interprétation dont la similarité est maximale.

Nous avons introduit nos exemples de trace afin d'illustrer l'impact des hésitations de l'utilisateur sur notre mécanisme d'interprétation. Par hésitation, nous entendons une opération de saisie qui est corrigée par la suite et qui n'a pas d'influence sur le résultat final. Pour distinguer ces opérations, il serait intéressant de pouvoir exprimer un symbole en indiquant que les arbres d'observations de ses états E_I et E_F (pour sa forme $E_I \cdot T \cdot E_F$) doivent être comparer respectivement avec l'état initial et l'état final de l'action. Pour l'action en cours, l'état final est l'état en cours, il est par nature sujet à évolution. La possibilité d'une telle expression dans un symbole, imposerait un mécanisme de remise en cause des appariements de symboles durant la réalisation de de l'action en cours, afin de tenir compte de l'évolution de l'état en cours. Si les symboles S_1 et S_2 étaient exprimés de la sorte, notre mécanisme d'appariement conduirait aux appariements souhaités pour ces symboles. De manière plus générale, une grande place est laissée à l'étude d'interfaces spécifiques aux actions, afin d'aider l'acteur à formuler l'expression symbolique d'une signature d'interprétation pour une action.

Enfin, nous avons considéré les hésitations comme du "bruit" pour le mécanisme d'interprétation. Une évolution de l'expression d'un symbole contribue à filtrer les opérations sans influence. Dans notre approche d'assistant, la vision de ces opérations sous forme de bruit est exagérée. Nous cherchons à privilégier les opérations ayant des répercussions durables sur l'état de l'entité manipulée, mais nous n'écartons pas la possibilité d'assister les situations d'hésitation. Dans cette perspective, pour décrire ses situations, notre modèle de pattern devrait certainement être complété en permettant d'exprimer l'absence, dans l'état final ou l'état en cours, d'une capture

d'OI comme significative.

9.4.2 Négociation de sens

Dans notre description d'un état de langage, nous utilisons les relations symbole-sens pour exprimer une signature dans son ensemble, et donc indifféremment :

- sa *partie problème*. C'est elle qui devrait être considérée pour évaluer le niveau d'identification d'une signature ;

- sa *partie solution*, qui ne doit pas être complètement identifiée si l'on souhaite pouvoir faire des suggestions. Dans le cas contraire, la signature a un rôle seulement descriptif mais pourrait faire partie de l'expression d'une autre signature.

En RàPC, ces deux parties sont fixées lors de la conception du système. En RàPET, étant donné que nous nous intéressons à des situations non décrites à l'avance, il n'est pas possible de les délimiter. Il serait donc intéressant de les apprendre, et de nuancer le calcul du taux d'identification.

Pour cela, une relation symbole-sens devrait être spécifiée pour exprimer l'habitude d'utiliser un symbole soit pour déterminer la partie problème, soit la partie solution. Dans l'absolu, un symbole pourrait ainsi apparaître deux fois dans l'expression d'une signature.

Un autre problème vient de la prise en compte des indications de l'utilisateur, dans notre transposition du jeu de nommage, lors de la modification de la définition symbolique d'une signature. Nous avons fait des écarts par rapport aux jeux qui nous inspirent en :

- supprimant la relation d'un symbole remplacé et en créant, avec héritage de la pondération, une relation pour le symbole accepté ;

- permettant à l'utilisateur de forcer une définition, ce qui revient à supprimer les relations pour les symboles rejetés.

Ces mécanismes sont introduits pour laisser à l'utilisateur le moyen d'intervenir directement sur les sens de l'agent alter ego, tout en laissant à ce dernier le moyen d'apprendre tant que possible, via l'héritage.

Dans la pratique, étant donné que nous sommes face à des tâches partiellement définies, l'utilisateur hésite entre des symboles pour exprimer une signature ou peut encore choisir en alternance différents symboles. Si l'utilisateur n'intègre pas tous les symboles qu'il peut utiliser dans la définition d'une signature, lors de chaque définition, alors les symboles utilisés en alternance ou remplacés ne sont pas correctement renforcés ; en effet, en cas d'utilisation alternée, c'est, dans le meilleur des cas, la dernière relation symbole-sens admise qui hérite de l'apprentissage par renforcement s'il y a remplacement de symbole, dans le pire des cas, les relations ne se renforcent pas s'il y a à chaque fois redéfinition de la signature (impliquant à chaque fois l'initialisation des pondérations des relations).

Pour palier cela, outre l'amélioration de l'expressivité des signatures, il serait intéressant de nuancer la négociation en conservant les relations qui ont déjà été établies par l'utilisateur lors de l'expression d'une signature, et en ajoutant des informations détaillant son acceptation ou son rejet à différents degrés (traités, en cas d'échec, de l'inhibition à la suppression directe). Une

telle nuance implique de revoir en conséquence la mesure de similarité entre signatures, qui est nécessaire à l'agent alter ego pour rapprocher des signatures similaires et qui conditionne leur fusion.

Troisième partie

Perspectives et Conclusion

Chapitre 10

Perspectives

Sommaire

Dans ce chapitre, nous abordons les perspectives qui s'ouvrent à notre travail. En 10.1, nous allons discuter de la possibilité d'introduire des pondérations pour nuancer notre modèle d'arbres d'observation, en présentant tout d'abord les évolutions que cela implique sur nos mesures. Nous allons ensuite discuter de l'initialisation de ces pondérations et enfin aborder l'utilisation de techniques d'apprentissage afin d'augmenter les capacités de perception de l'agent alter ego. En 10.2, nous allons discuter de l'application de notre approche au partage d'expériences individuelles, puis in fine le partage d'expérience collective, selon l'objectif initial du projet OSCAR. En 10.4, nous traiterons de l'enrichissement de la sémantique des signatures de tâche, qui est pour l'instant limitée à un ensemble de symboles, et qui gagnerait donc à être plus expressive. L'accomplissement du cycle de RàPET sera abordé en 10.5. Enfin, nous traiterons en 10.6 de perspectives de validation expérimentale de notre approche.

10.1 Gestion des pondérations de symbole

Notre première approche donne à l'agent alter ego une perception strictement symbolique d'une situation, via notre modèle d'arbre d'observation et de pattern. Aucune nuance n'est exprimée dans ce modèle, quand à l'importance relative d'un symbole (via celle de ses une captures variables ou constantes) par rapport aux autres dans l'expression d'une signature. En ajoutant des pondérations aux captures des symboles, nous voulons donner à l'utilisateur un moyen de "nuancer" (du peu important au très important) les indications qu'il donne. D'autre part, notre agent alter ego n'a, pour l'instant, qu'un rôle d'observateur et de gestionnaire des symboles, sans aucune pro-activité visant à améliorer ses capacités "sensorielles". Pour nuancer les observations

et les patterns, il conviendrait de pondérer les captures qu'ils contiennent. La section 10.1.1 introduit de telles pondérations et présente les évolutions que cela implique et leurs répercussions sur la problématique de comparaison de symboles. La section 10.1.2 présente et discute un moyen d'initialiser les pondérations de capture.

Dans nos mesures de similarité, présentés au chapitre 6, nous avons déjà introduit les pondérations suivantes, qui sont propres à chaque symbole :

- $p_{pattern}$ et p_E, qui représentent respectivement les poids donnés à la partie appariée et aux parties non appariées dans le calcul de la similarité entre deux observations, au regard d'un appariement partiel de chaque observation avec un pattern, c.f., équation 6.2.

- p_{pos} où $pos \in \{Av, Op, Ap\}$, qui sont les poids donnés aux trois similarités, obtenues avec l'équation 6.2 en comparant respectivement les états initiaux, les transitions et les états finaux de deux sections. Ces similarités et leurs poids permettent, via une moyenne pondérée, le calcul de similarité entre sections, au regard de l'appariement avec un symbole, c.f., équation 6.9.

Il n'est pas envisageable de demander à l'utilisateur d'ajuster directement ces pondérations, car elles relèvent plutôt de la connaissance du fonctionnement interne du système. Toutefois, notre démarche nous conduit naturellement à chercher à acquérir de l'utilisateur des indications pour fixer les pondérations de captures ; mais l'hypothèse de son ignorance du système implique le besoin d'introduire un mécanisme de correction des valeurs de pondération, qui doit être géré par l'agent alter ego. Cette intégration d'indications de l'utilisateur pour fixer le poids d'une capture et leur correction par l'agent alter ego doit tenir compte du caractère potentiellement polysémique d'un symbole, qui pourrait conduire à des évolutions de ce dernier. Les mécanismes d'acquisition et de correction devront tenir compte de cela. La section 10.1.3 aborde cette problématique.

Schématiquement, dans cette perspective, la recherche sur chaque section acquise d'appariements de symboles peut être vue comme un flux de tentatives de perception. Pour une tentative d'appariement d'un symbole sur une section, les catégories de pondérations, qui sont caractéristiques d'une polysémie, précédemment acquises de l'utilisateur doivent être considérées séparément. Considérons que l'agent alter ego dispose d'une valeur de correction pour chacune d'elles. Nous discutons l'intérêt d'étudier et d'utiliser ce flux de valeurs de pondération pour augmenter les capacités de perception et d'acquisition de sens de l'agent alter ego, via l'amélioration des valeurs de correction.

10.1.1 Pondération des captures d'arbre d'observation

Considérons qu'une valeur de pondération $p_c \in \mathcal{R}^+$ est associée à chaque capture c (constante ou variable) d'un arbre d'observation (de trace ou de symbole). Le calcul de similarité entre deux observations, partiellement appariées à un pattern, doit être modifié pour tenir compte de ces pondérations.

Nous rappelons l'équation 6.3, déterminant la similarité entre deux parties d'arbre d'observation appariées à un pattern :

$$sim_{Obs_{pattern}}(M_i, M_j) = \frac{dim(M_i \cap M_j) + \sum_{v \in variables(Obs_{pattern})} sim(m_i(v), m_j(v))}{dim(M_i \cup M_j) - nombre\_de\_variables(Obs_{pattern})}$$

où $m_i(v)$ et $m_j(v)$ représentent respectivement l'appariement d'une variable de pattern v avec une capture de Obs_i et Obs_j, et avec la dimension d'une observation $dim(Obs)$ qui est définie par :

$$dim(Obs) = \textit{nombre de captures d'OI (constantes ou variables) dans Obs}$$

En incluant les pondérations de captures, l'équation ci-dessus devient :

$$sim_{Obs_{pattern}}(M_i, M_j) = \frac{\sum_{c \in C(M_i \cap M_j)} p_c + \sum_{v \in V(Obs_{pattern})} p_v \cdot sim(m_i(v), m_j(v))}{\sum_{c \in C(M_i \cup M_j)} p_c - \sum_{v \in V(Obs_{pattern})} p_v} \quad (10.1)$$

où :
- $C(X)$ représente l'ensemble des captures, constantes ou variables, de l'arbre d'observation X ;
- $V(X)$ représente l'ensemble des captures variables de l'arbre d'observation X ;
- $m_i(v)$ représente la capture de M_i appariée avec la variable v, et $m_j(v)$ celle de M_j,

Dans cette nouvelle définition, nous gardons les principes génériques de mesure de similarité, en remplaçant $dim(Obs)$ par $\sum_{c \in C(Obs)} p_c$.

D'autre part, nous ne tenons pas compte des pondérations des captures d'arbre d'observation de trace lorsque ces captures sont appariées à une variable de pattern ; c'est la pondération de variable qui est utilisée. Cette définition est une version simplifiée, car dans l'absolu on devrait tenir compte des trois pondérations et avoir :

- au numérateur, p_v dans $\sum_{v \in V(Obs_{pattern})} p_v \cdot sim(m_i(v), m_j(v))$ remplacé par une fonction $f(p_v, p_{m_i(v)}, p_{m_j(v)})$;
- au dénominateur, $\sum_{v \in V(Obs_{pattern})} p_v$ devrait refléter la fonction $f()$ introduite ci-dessus.

Similarité de patterns : nous avons également défini une mesure de similarité entre deux patterns O_i^* et O_j^*, c.f., équation 6.5, qui est utilisée par l'agent alter ego pour rapprocher les symboles similaires au symbole en cours de négociation et qui permet de tester l'égalité de symboles. Nous rappelons cette mesure de similarité :

$$\begin{aligned} sim(O_i^*, O_j^*) &= \frac{Card(similaire)}{Card(totalité)} \\ &= \frac{Card(identique) + Card(approx.\ variables)}{Card(totalité)} \end{aligned}$$

où, en s'appuyant sur les masques :

- I, le plus grand sous-arbre de containers commun aux arbres de containers de O_i^* et O_j^* ;
- U, l'arbre de containers correspondant à l'union des arbres de containers de O_i^* et O_j^*.

On a :

$$\begin{aligned} Card(identique) &= \sum_{(c,c') \in I_i \times I_j} |C_i \cap C_j| + |V_i \cap V_j| \\ Card(approx.\ variables) &= \sum_{(c,c') \in I_i \times I_j} \left(\sum_{v \in V_i - V_i \cap V_j} \max_{v' \in V_j - V_i \cap V_j} sim(v, v') \right) \\ Card(totalité) &= \sum_{(c,c') \in U_i \times U_j} |C_i \cup C_j| + |V_i \cup V_j| \end{aligned}$$

avec :

- en notant $\sum_{(c,c') \in I_i \times I_j}$, respectivement $\sum_{(c,c') \in U_i \times U_j}$, nous exprimons que la somme est effectuée en parcourant les paires de containers (c, c') qui se correspondent dans I_i et I_j, respectivement U_i et U_j. Les ensembles C_i, C_j, V_i et V_j sont fonction d'une telle paire de containers ;
- C_i est l'ensemble des constantes d'un container c de I_i ou de U_i. De même, C_j est l'ensemble des constantes d'un container c' de I_j ou U_j ;
- V_i est l'ensemble des variables d'un container c de I_i ou de U_i. De même, V_j est l'ensemble des variables d'un container c' de I_j ou U_j ;
- pour la somme sur $(c, c') \in I_i \times I_j$ de $Card(approx.\ variables)$, nous mettons la condition que pour chaque paire de containers traitée, on a : $|V_i - V_i \cap V_j| < |V_j - V_i \cap V_j|$, sinon les rôles de i et de j sont intervertis.
  Cette condition est mise afin que la somme des similarités maximales pour chaque paire de containers soit majorée par le cardinal du plus petit des deux ensembles de variables non identiques ($V_i - V_i \cap V_j$ et $V_j - V_i \cap V_j$). v et v' sont des variables de l'un et l'autre de ces ensembles.

En incluant les pondérations de captures, l'équation ci-dessus deviendrait de la forme :

$$
\begin{aligned}
Card(identique) &= \sum_{(c,c')\in I_i\times I_j}\left(\sum_{x\in C_i\cap C_j} p_x + \sum_{x\in V_i\cap V_j} p_x\right) \qquad (10.2)\\
Card(approx.\ variables) &= \sum_{(c,c')\in I_i\times I_j}\left(\sum_{v\in V_i - V_i\cap V_j}\ \max_{v'\in V_j - V_i\cap V_j} g(p_v, p_{v'})\cdot sim(v,v')\right)\\
Card(totalité) &= \sum_{(c,c')\in U_i\times U_j}\left(\sum_{x\in C_i\cup C_j} p_x + \sum_{x\in V_i\cup V_j} h\left(p_{V_i}(x), p_{V_j}(x)\right)\right)
\end{aligned}
$$

où, le cardinal d'un ensemble est remplacé par la somme des pondérations de ses éléments. Cependant, pour des raisons similaires au cas général de la mesure de similarité pondérée entre deux parties d'arbre d'observation appariées à un pattern, mais cette fois incontournables, $g()$ et $h()$ sont des fonctions à déterminer. $g(p_v, p_{v'})$ est une fonction de combinaison des pondérations de variables v et v' rapprochées par similarité et $h\left(p_{V_i}(x), p_{V_j}(x)\right)$ une fonction de combinaison entre les pondérations de captures de V_i et V_j, représentant une même variable x dans un container commun à O_i^* et O_j^*. Dans les deux cas, il s'agit de trouver une fonction effectuant un "choix" porteur de sens entre deux pondérations ; pour cela, il peut être intéressant de discretiser de telles variables réelles.

10.1.2 Initialisation des pondérations

L'introduction de pondérations pour les captures d'arbres d'observation de trace ou de pattern implique le besoin de traiter leur initialisation. Notre application à un espace documentaire, nous amène tout naturellement à nous intéresser à la technique de pondération des vecteurs de mots-clés, utilisée dans le domaine de la recherche/classification de documents (*Information Retrieval*).

Un mot-clé kw est pondéré à l'aide de la mesure $TFIDF$[1] qui exprime l'importance que tient kw dans la description d'un document D_j ; kw est présent au travers d'un ensemble total de documents considérés (corpus documentaire). Nous donnons une expression simple de la mesure $TFIDF$, pour laquelle de nombreuses définitions existent :

$$TFIDF(kw, D_j) = TF(kw, D_j)\cdot IDF(kw)$$

Avec la fréquence de kw dans un document D_j du corpus donnée par :

$$TF(kw, D_j) = \frac{N\cdot N(kw, D_j)}{N(kw)\cdot N(D_j)}$$

où :
- N est le nombre de mots-clés dans tout le corpus ;
- $N(kw, D_j)$ est le nombre d'occurrences de kw dans D_j ;
- $N(kw)$ est le nombre d'occurrences de kw dans le corpus ;
- $N(D_j)$ est le nombre de mots-clés dans D_j.

[1] Term Frequency Inverse Document Frequency

Et l'*Inverse Document Frequency* donnée par :

$$IDF(kw) = \log(\frac{nbDoc}{DF_{kw}})$$

où
- $nbDoc$ est le nombre total de documents dans le corpus ;
- DF_{kw} est le nombre de documents du corpus contenant kw.

Avec cette mesure, les mots-clés se trouvant dans la majorité des documents du corpus et ceux qui sont propres à quelques documents ont un poids faible et ne tiennent donc pas un rôle important dans les mécanismes de catégorisation.

Dans notre cas, cette mesure peut être une base à l'initialisation automatique des pondérations de captures d'arbre d'observation ; en proposant d'appliquer les principes de TFIDF, nous faisons l'approximation que le poids d'une capture par rapport à un *corpus documentaire* est un moyen d'exprimer le poids au regard d'un usage réel au travers des interactions. La mesure TFIDF est très sensible au corpus documentaire utilisé et elle n'est pas convenable pour traiter des mots singuliers lorsqu'elle est appliquée à un ensemble trop général. La détermination du corpus doit refléter un champs d'interactions pour être représentatif. La description de l'activité sur la base d'une décomposition en entités documentaires ainsi que les traces d'opérations sur ces documents peuvent servir à la définition du corpus ; il pourrait aussi être intéressant de donner à l'utilisateur des moyens d'apporter des précisions permettant à l'agent alter ego de construire ce corpus en situation.

Nous désignons par $p_{corpus}(c)$ la pondération initiale, obtenue en appliquant de tels principes, pour une capture d'arbre d'observation. Cette approche peut être convenable pour initialiser les pondérations de captures de trace, mais elle est bien plus discutable pour celles de symbole. En effet, un symbole exprime une connaissance par la mise en évidence d'un caractère significatif en soi, et c'est donc cette connaissance qui doit guider la pondération des captures de symbole. Pour ces raisons, dans la section suivante, nous abordons la prise en compte des indications de l'utilisateur et la possibilité de doter l'agent de mécanismes d'apprentissage, intervenant comme une rétro-action pour corriger les pondérations de captures de symboles.

10.1.3 Prise en compte des indications de l'utilisateur et apprentissage

Nous avons déjà argumenté dans la description de notre approche, que nous considérons que l'utilisateur est le mieux à même d'indiquer à l'agent alter ego ce qui est significatif pour lui, et cela nous a conduit à représenter ces indications par des symboles, en vue de la négociation de sens. Avec la même démarche, il serait intéressant de donner à l'utilisateur le moyen d'insister sur l'importance d'un élément dans la définition d'un symbole. En considérant que cette indication puisse se ramener à une valeur $p_{utilisateur}(c)$ pour une capture de symbole c, son intégration à la pondération totale p_c de c pourrait se faire simplement par :

$$\forall\ c \in captures(Symbole),\quad p_c = p_{utilisateur}(c) \cdot p_{corpus}(c)$$

où $p_{corpus}(c)$ est la pondération initiale obtenue avec l'approche discutée dans la section précédente. Une modification de l'interface est nécessaire, elle peut se faire simplement dans un premier temps en permettant à l'utilisateur d'influer sur $p_{utilisateur}(c)$ à l'aide de boutons $+/-$.

L'ensemble des pondérations d'un symbole, apprises d'indications provenant de l'utilisateur, peuvent être représentées par le vecteur de pondération V_U. Ce vecteur est constitué des poids p_{pos}, $p_{pattern_{pos}}$ et $p_{E_{pos}}$, dont les rôles ont été rappelés en 10.1, ainsi que des poids de toutes les captures du pattern situé à une position *pos*. Les positions sont $\{Av, Op, Ap\}$, elles désignent respectivement l'état initial, la transition et l'état final de la section de trace considérée pour l'appariement. La forme générale de V_U est la suivante :

$$V_U = \left(\left[p_{pos},\ p_{pattern_{pos}}, p_{E_{pos}}, [\ p_{utilisateur}(c),\ ]^{c \in C(Obs_{pattern_{pos}})} \right]^{pos\ \in\ \{Av,Op,Ap\}} \right)$$

Etant donné qu'un symbole est voué à évoluer, il peut à un moment avoir un rôle polysémique, ce qui a de grande chance de se traduire en une catégorie (*cluster*) de vecteurs de pondération V_{U_i} pour chaque sens. Dans cette logique, il conviendrait d'établir des mécanismes de clustering sur l'ensemble des vecteurs de pondération appris pour un symbole.

Soient deux vecteurs de pondération $V_{U_1} = (x_1, \ldots, x_N)$ et $V_{U_2} = (y_1, \ldots, y_N)$ parmi un ensemble $\mathcal{V}_S$ de vecteurs de dimension N, appris pour un symbole S. L'application des techniques de clustering existantes sur $\mathcal{V}_S$, sur la base d'une métrique entre V_{U1} et V_{U2}, par exemple la distance euclidienne (rappel : $d(V_{U1}, V_{U2}) = \sum_{i=1}^{N}(x_i - y_i)^2$), pourrait mener à l'identification de catégories distinctes. De telles catégories peuvent servir à l'agent alter ego pour :

- synthétiser un éventuel grand nombre de vecteurs de pondération acquis de l'utilisateur ;
- faciliter l'initialisation des pondérations d'un nouveau symbole, à partir des pondérations apprises de la catégorie mère du symbole d'origine ;
- proposer une deuxième mesure de similarité entre sections au regard de l'appariement avec un symbole, plus fine, dès lors que l'agent alter ego connaît une catégorie sémantique à privilégier dans l'interprétation.

La distance euclidienne n'est bien entendu qu'une première approche, elle peut être pondérée et/ou être ramenée, pour plus de finesse, à la taille de l'intervalle de variation de la $i^{ème}$ composante des instances de vecteur comparées. Il est important de noter que les composantes d'un vecteur de pondération n'ont pas les mêmes statuts :

- certaines représentent des pondérations "systèmes", i.e., p_{pos}, $p_{pattern_{pos}}$, $p_{E_{pos}}$;
- d'autres sont vraiment visibles à l'utilisateur, i.e., les pondérations de captures.

Le clustering doit prendre cela en compte, et il peut se faire par étape, en fonction de ces statuts.

Toutefois, la prise en compte d'indications de pondération de l'utilisateur ne doit pas se faire en partant du principe que ce dernier est seul garant des performances du système, car il ne dispose pas des connaissances sur son fonctionnement interne. Dans notre première approche, nous avons écarté ce problème en dotant notre agent alter ego d'aucun mécanisme de pondération et donc de correction ; dans notre démonstrateur, nous avons même totalement écarté la problématique des pondérations en posant $\forall pos, p_{pattern_{pos}} = 1$, $p_{E_{pos}} = 0$ et $\forall c, p_c = 1$.

L'introduction d'une correction $p_{corr}(c)$ dans la valeur de pondération finale p_c d'une capture pourrait se faire de la façon suivante :

$$\forall\ c \in captures(Symbole),\quad p_c = p_{corr}(c) \cdot p_{utilisateur}(c) \cdot p_{corpus}(c)$$

et en ajoutant, de la même manière, à p_{pos}, $p_{pattern_{pos}}$ et $p_{E_{pos}}$ des valeurs de correction pc_{pos}, $pc_{pattern_{pos}}$ et $pc_{E_{pos}}$, où $pos \in \{Av, Op, Ap\}$. Ces valeurs de correction se rassemblent en un vecteur V_{corr}, symétrique de V_U , dont la forme générale est :

$$V_{corr} = \left(\left[pc_{pos},\ pc_{pattern_{pos}}, pc_{E_{pos}}, [\ p_{corr}(c),\ ]^{c \in C(Obs_{pattern_{pos}})} \right]^{pos\ \in\ \{Av,Op,Ap\}} \right)$$

La détermination de V_{corr} laisse place à l'utilisation des diverses techniques d'apprentissage automatique existantes. Il serait légitime de chercher à déterminer un vecteur de correction spécifique à chaque catégorie, issue du clustering abordé ci-dessus. L'apprentissage doit se faire dans la perspective d'améliorer les "capacités cognitives" des mécanismes de perception de l'agent alter ego. Tout comme les pondérations apprises de l'utilisateur, les pondérations de correction peuvent faciliter l'initialisation des pondérations en cas d'évolution d'un symbole. Il est possible d'activer la mise à jour des valeurs de correction de manière transparente à l'utilisateur, lors de chaque appariement (accepté par l'utilisateur) du symbole associé, en cherchant à exprimer, via le classement des propositions et sa correction, le fait que l'utilisateur estime qu'une proposition satisfait sa recherche d'expérience.

10.2 Vers le partage d'expérience

Au chapitre 5, nous avons indiqué que notre travail sur la réutilisation de l'expérience individuelle s'inscrivait dans un contexte collectif, selon les besoins initiaux du projet Oscar. Nous avons argumenté que, pour permettre la suggestion en situation lors de tâches partiellement définies, une condition d'efficacité de l'agent alter ego résidait dans le fait qu'il dispose de sens partagés avec l'acteur humain. Pour permettre l'acquisition de ces sens, nous avons proposé une approche inspirée des travaux sur l'émergence de langage entre robots, en la transposant à un mécanisme de négociation de sens entre l'utilisateur et son alter ego.

Ces travaux, qui constituent pour nous une démonstration expérimentale d'une théorie d'émergence de système de communication à base de jeux, ont été menés dans le contexte d'agents artificiels (des robots). Il est donc légitime de concevoir que cette approche puisse s'appliquer à la communication entre agents alter ego. Une distinction est à apporter à ce niveau, dans la mesure où :

- les agents alter ego peuvent assister des utilisateurs, qui tiennent le même rôle dans la communauté. Nous allons traiter dans un premier temps cette situation, où les agents alter ego ont des rôles d'*agents pairs* ;

- les agents alter ego peuvent assister des utilisateurs, qui tiennent des rôles distincts dans la tâche collective. Nous allons ensuite aborder les problématiques soulevées par cette situation d'*agents hétérogènes*.

Le partage d'expérience individuelle

Dans le domaine du RàPC, de nombreux travaux de Plaza et al. (c.f., [Prasad 96], [Plaza 97], [Martin 98], [Martin 99], [Plaza 01], [Ontanon 03]) ont étudié la collaboration entre agents pairs pour les différentes étapes du cycle. Ils mettent clairement en évidence le gain que constitue le partage de cas et de méthodes entre des agents pairs, tant au niveau de l'initialisation d'un agent que dans le fonctionnement régulier. Ces travaux argumentent l'intérêt de nous intéresser au partage d'expérience, sous toutes ses formes.

Dans notre approche, l'émergence d'un langage commun aux agents alter ego, qui assistent des acteurs tenant le même rôle au sein de la communauté, serait une forme de *partage de méthodes*, i.e., des éléments d'interprétation de trace desquels dépendent le calcul de similarité. Le partage de traces serait quant à lui directement à rapprocher d'un *partage de cas*, mais qui resterait guidé par une interprétation.

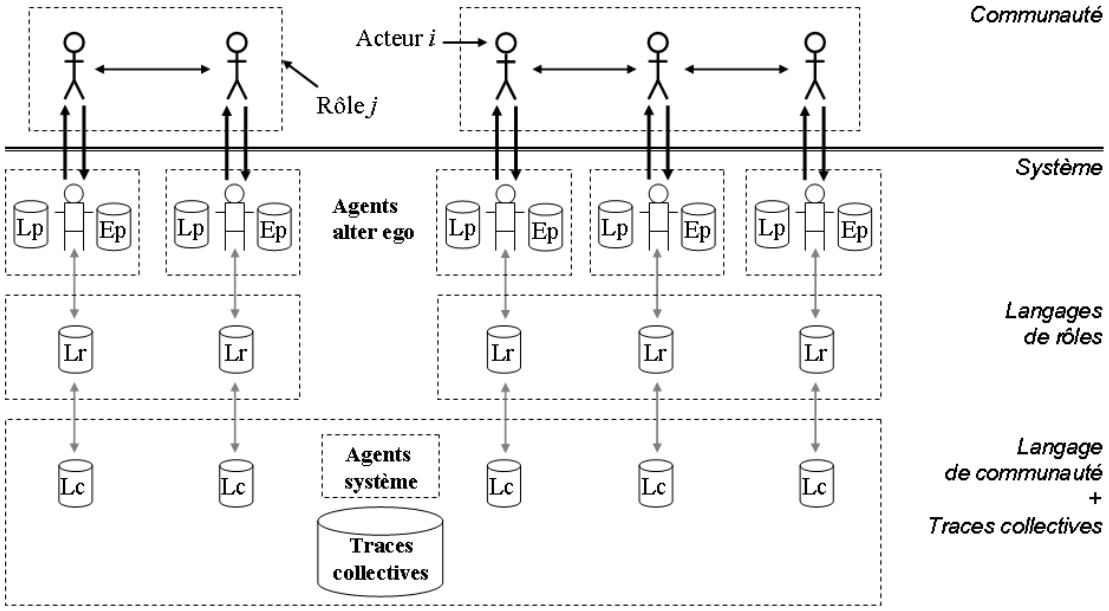


FIG. 10.1 – Langages pour le partage et l'échange d'expérience dans une communauté

Cette perspective doit être abordée en laissant aux agents alter ego toutes leurs capacités de personnalisation et en veillant à ce que l'utilisateur garde le contrôle de son assistant. Il conviendrait donc de faire évoluer conjointement un langage personnel et un langage de rôle, via un *mapping*, en n'écartant pas la possibilité de laisser à l'utilisateur le moyen de contrôler directement ce mapping. La figure 10.1 illustre l'architecture d'un système de partage d'expérience, en se plaçant au niveau de la communauté. Un langage de rôle, noté L_r, est associé à chaque agent, et le passage bi-directionnel de symboles et de signatures entre le langage personnel (noté L_p) et le langage de rôle permet de construire le mapping. L'agent alter ego doit, autant que possible, chercher à établir un mapping bijectif :

- en cas d'incompétence, l'agent doit acquérir (dans L_r et dans L_p) les éléments d'interprétation couramment admis par les agents du même rôle. Ceci va dans le sens de l'emergence d'un langage de rôle ;

- dans le cas de la mise en commun d'éléments d'interprétation entre agents alter ego, chaque agent doit privilégier le langage personnel, afin que l'utilisateur reste maître de son assistant, tout en cherchant à établir des éléments communs pour développer le langage de rôle.

La notion de rôle sur laquelle se base notre décomposition mérite discussion. Nous pouvons définir intuitivement un rôle, par le nom donné à un ensemble d'acteurs humains effectuant une même catégorie de tâches sur les mêmes entités de connaissance au regard d'une activité

collaborative. De plus, les acteurs peuvent tenir plusieurs rôles, avec des compétences propres dans chacun d'eux.

Pour ces raisons, il reste à étudier la construction d'un modèle de rôle et de compétence de façon dynamique, au fil de la manipulation du système par la communauté. En 4.2, nous avons présenté le travail de [Lashkari 94], qui permettrait de construire un "réseau de compétences", et donc de rôles. Un tel réseau permettrait d'appliquer les travaux de [van Diggelen 05] pour fixer des politiques de propagation des éléments d'interprétation au sein d'un rôle et de la communauté. Dans cette approche, il est montré que la prise en compte du statut que tient un acteur dans la communauté permet l'établissement plus rapide d'un vocabulaire commun.

Vers un système d'agents assistants pour la réutilisation de l'expérience collective

Notre approche a été développée dès le début dans le but final de permettre la réutilisation de l'expérience collective. Nous avons argumenté que cela passait par la prise en compte des expériences individuelles, i.e., les traces, et par l'établissement de connaissances d'interprétation, au moins partagées par un acteur humain et son agent alter ego et au mieux communément admises par un rôle. Toutefois, l'expérience collective de la communauté ne peut pas simplement se ramener à une simple concatenation des expériences individuelles des membres de chaque rôle.

En adoptant une représentation de l'expérience collective sous forme de trace, à l'instar de l'expérience individuelle, la construction d'une telle trace collective peut se faire en s'appuyant sur une ontologie descriptive d'activité collaborative, telle que celle proposée par [Troussier 99] pour l'activité de conception intégrée. Outre sa valeur métier, son approche peut être mise en rapport avec les travaux sur la *théorie de l'activité* (TA) ; pour la conception d'interfaces co-évolutives, [Bourguin 05] effectuent une synthèse de ces travaux et proposent une ontologie opérationnelle descriptive d'une activité à partir des principes de la TA. Cette ontologie est intéressante dans notre cas car elle permettrait d'apporter les informations complémentaires permettant de positionner les traces individuelles par rapport à l'activité de la communauté, et ainsi guider la construction de traces collectives.

Les traces individuelles ne tiennent par le même rôle dans la construction d'une trace collective, en premier lieu par le fait qu'elles décrivent tantôt les étapes de transfert d'objets de travail entre acteurs de la communauté, tantôt les différentes opérations effectuées par un acteur pour réaliser une tâche sur un objet. La construction de la trace collective doit faire la distinction entre ces deux rôles, et cela peut faciliter la présentation de la trace collective à l'utilisateur.

Les travaux de la communauté du CSCW[2] sont une base d'outils et de modèles pour le support informatique de la coopération. La coopération se situe dans la trace collective au niveau des phases de transfert des objets de travail, et on ne peut pas exclure des situations de collaboration dans les phases de réalisation "individuelle" de tâche. Cette dernière situation se ramène au partage d'expérience individuelle abordé ci-dessus. Les situations de coopération et de collaboration sont des lieux privilégiés pour la négociation, le partage et l'affinement des connaissances entre acteurs, dont les agents alter ego pourraient bénéficier afin de faire évoluer les ontologies d'activité et de domaine. Dans notre approche à base de langage, avec une structuration des acteurs en rôles multiples qui peut être intégrée au modèle d'activité, ces phases de négociations pourraient être utilisées pour construire un langage de communauté utilisé pour effectuer un mapping entre

[2] *Computer Supported Cooperative Work*

les langages de rôle, avec leurs ontologies de domaine propres.

L'objectif d'un système d'agents assistants pour la réutilisation en situation de l'expérience collective impliquerait de nombreux traitements. En effet, pour construire des traces collectives et permettre leur rapprochement selon la "vue" qu'a un acteur tenant un certain rôle dans une trace collaborative en cours, et selon la nature de la coopération de l'acteur, des structures de données et des algorithmes appropriés doivent être étudiés. Compte tenu du caractère dynamique et incertain de l'interprétation, il serait intéressant d'introduire des "agents système" pour réaliser les tâches de construction de trace collective selon l'affinement des besoins d'assistance des acteurs, ainsi que pour les effectuer les opérations d'alignement et de construction d'ontologies de domaines et d'activité issues des négociations ; de tels agents système auraient pour tâche de conduire à l'émergence de structures stables dans ces ontologies.

10.3 Vers la proactivité de l'agent alter ego dans la discrimination des situations d'assistance

Dans notre retour d'expérience au sujet de la négociation de sens, c.f., 9.4.2, nous avons discuté du fait que notre approche nécessite de l'utilisateur qu'il énumère à chaque négociation de signature l'ensemble des symboles qui sont en mesure de participer à l'expression de la signature cible. Ceci constitue une contrainte, qui n'a pas d'utilité directe pour l'utilisateur ; en particulier dans le cas où des ensembles de symboles servent à exprimer des alternatives pour la signature. Dans la section suivante, nous discuterons de l'évolution du modèle de signature. Avant cela, nous allons discuter pour notre modèle de signature, i.e., un ensemble de symboles, de la possibilité de donner à l'agent alter ego une relative proactivité dans la distinction de sens d'assistance ; la création de sens d'assistance doit néanmoins rester sous le contrôle de l'utilisateur, la fusion de sens d'assistance ce faisant en cas d'ambiguïté de signature après accord de l'utilisateur.

Nous pensons qu'il peut être intéressant d'ajouter à l'agent un historique de l'utilisation des symboles dans la définition d'un sens d'assistance, une signature. Considérons une signature Z ayant jusqu'ici été exprimée avec les symboles $\{\sigma_i\}_{i\in[\![1;N]\!]}$. La $j^{ème}$ expression de Z par l'utilisateur, dans cet espace de symbole, peut être représentée par un vecteur binaire $V_{Z,j}$ de dimension N, dont la $i^{ème}$ composante vaut 1 si le symbole σ_i est utilisé dans cette expression, 0 sinon.

Pour un espace de symbole fixe, l'ensemble des expressions de la signature, ayant déjà été convenues avec l'utilisateur après négociation, est représenté par l'ensemble de *vecteurs d'expression* $\{V_{Z,j}\}_j$. Si un nouveau symbole σ_{N+1} apparaît dans l'expression de Z, l'espace de symbole pour Z change et devient de dimension $N+1$, et il est nécessaire de revoir l'ensemble $\{V_{Z,j}\}_j$ selon ce nouvel espace avant d'inclure le nouveau vecteur d'expression avec σ_{N+1}. Un vecteur d'expression porte des informations proches de celles des relations symbole-signature d'une signature. Il serait intéressant d'en faire une représentation unifiée en y intégrant les pondérations ; dans notre cas, les relations symboles-signature d'un état de langage correspondraient au dernier vecteur d'expression d'une signature.

Considérons à nouveau le cas d'espace de symbole fixe pour une signature Z, il serait intéressant d'analyser l'ensemble des vecteurs d'expression $\{V_{Z,j}\}_j$ à l'aide de techniques de clustering afin d'identifier des clusters de vecteurs d'expression dont les membres partagent le plus de symboles. De tels clusters seraient le signe d'expressions alternatives pour Z dans diverses situations, sans pour autant exclure un tronc symbolique commun aux clusters. A l'aide de tels clusters, l'élaboration des épisodes peut être affinée en restreignant dès que possible à un cluster le calcul

du taux d'identification et la mesure de similarité entre épisodes, c.f., équations 6.8 et 6.10. Cela augmenterait la pertinence de ces mesures : pour la similarité entre épisodes, via celle des facteurs de couverture et de représentativité.

La présence de clusters pourrait être la base à une proactivité de l'agent alter ego dans la distinction de situations d'assistance, en lui donnant la possibilité de suggérer à l'utilisateur de "scinder" une signature suivant des clusters identifiés, i.e., créer une signature distincte pour chaque cluster, ce qui est la condition du bon fonctionnement de nos mesures. Cette opération doit rester supervisée par l'utilisateur, et il peut par exemple choisir de garder le même label pour les signatures ainsi créées. Un tel mécanisme peut profiter de mécanismes d'héritage pour initialiser les pondérations des relations symbole-signature, puis chaque signature suivrait une évolution indépendante.

Un tel mécanisme de discrimination de situations d'assistance nécessiterait de revoir la mesure de similarité entre signature, qui conditionne leur fusion, c.f., équation 6.6. Par ailleurs, elle ne permet pas de prendre convenablement en compte des interdépendances entre clusters, comme cela serait le cas dans l'expression complexe d'alternatives d'une situation d'assistance. Dans la section suivante, nous allons discuter de l'enrichissement de la sémantique des signatures par ajout de relations inter et intra-symbole et par composition de signature. Dans cette perspective, une proactivité de l'agent alter ego doit faire l'objet d'étude spécifique.

10.4 Vers l'enrichissement sémantique des signatures par émergence de grammaire

Il est bien évident que la définition des signatures de tâche adoptée jusqu'ici, i.e., un ensemble de symboles sans relation entre eux, demeure trop simple pour exprimer certaines signatures. Nous allons discuter de son enrichissement de manière constructive, en gardant à l'esprit que cela doit être guidé par l'acteur, et donc lui faire sens, tout en n'écartant pas la possibilité de doter l'agent alter ego de proactivité dans la discrimination des situations d'assistance afin de libérer tant que possible l'acteur humain de la tâche de formulation de signature.

Nous allons traiter des besoins d'expressivité d'une signature apparus dans notre travail et dans les perspectives abordées jusqu'ici et les difficultés qu'ils soulèvent. Nous allons ensuite introduire un modèle opérationnel proposé dans des travaux fondamentaux sur les mécanismes d'émergence de grammaire, abordés en 4.6.3 comme suite logique des résultats sur l'émergence de lexique et de sens. Ces travaux sur l'émergence de grammaire seraient à prendre en considération afin de poursuivre notre effort de construction de sens partagés par un acteur et son agent alter ego, mais un fossé existe à l'heure actuelle entre les formes de grammaires obtenues expérimentalement et les grammaires naturelles. Pour ces raisons, nous allons positionner notre approche et indiquer les pistes intermédiaires permettant de combler ce fossé.

Besoins et difficultés : dans notre première approche, nous avons fait le choix d'exprimer une signature à l'utilisateur par une phrase en langue naturelle, provenant de sa propre saisie, et de lui présenter sa formulation symbolique, i.e., un ensemble non ordonné. Ceci est une simplification extrême, qui offre peu de capacité de discrimination à l'agent alter ego, mais qui constitue une simplification allant dans le sens de l'émergence d'un langage lexical dans la mesure où l'utilisateur a un accès total aux éléments de sens de l'agent alter ego. Pour augmenter la perception par l'agent alter ego du sens exprimé dans le label d'une signature exprimée en langue naturelle

par l'utilisateur, il serait intéressant de donner à l'acteur humain le moyen de faire référence à une capture d'OI située dans un symbole de la signature, le reste du symbole jouant le rôle de "contexte de perception", tout comme utiliser une signature pour exprimer une autre signature ; cela correspond à la composition de signatures.

Dans notre premier retour d'expérience sur le module de RàPET, c.f., 9.4.1, nous avons argumenté le besoin de faire évoluer l'expression des symboles et leur traitement pour pouvoir distinguer les opérations de l'utilisateur qui relèvent de l'*hésitation* de celles qui participent à l'obtention du résultat "final". Cette évolution participe aussi à l'augmentation de l'expressivité d'une signature, via celle des symboles.

Un apport significatif viendrait à ajouter des contraintes de différentes natures entre les captures de symboles et/ou de signatures imbriquées. D'autre part, nous avons discuté en 9.4.2 les limites et les travers de notre transposition des mécanismes de négociation, en s'intéressant au cas où l'utilisateur choisit alternativement différents symboles pour exprimer une signature, ou simplement hésite entre des symboles. Enfin, il doit être distingué le double rôle que jouent les relations symbole-sens, i.e., identification d'une interprétation de la *partie problème* d'une situation d'assistance et expression du rapprochement avec une situation antérieure (parties *problème* et *solution*).

L'expression de telles contraintes ne peut être uniquement à la charge de l'utilisateur, il conviendrait donc de doter l'agent alter ego d'une proactivité dans la construction d'une signature, en tirant partie par exemple des régularités et différences dans la vérification des contraintes. Il n'est pas envisageable de demander à l'agent alter ego d'acquérir une expression d'une signature, permettre à l'utilisateur d'interagir avec cette définition est donc nécessaire ; des principes d'interface doivent être étudiés pour exprimer une signature et permettre l'interaction de l'utilisateur.

Introduction du modèle de *Fluid Construction Grammar* : dans [Steels 05], il est discuté la nécessité d'introduire une dimension grammaticale constructive pour l'expression de certains sens afin d'obtenir des systèmes de langage entre agents, sans élément prédéfini. Le travail présenté est appliqué à l'immersion dans un monde réel de robots munis d'appareils sensori-moteurs. Ce travail propose un modèle opérationnel appelé *Fluid Construction Grammar* (FCG), dont nous allons brièvement résumer les points importants sans chercher à en faire une description détaillée.

Le modèle de FCG est basé sur l'utilisation de règles bidirectionnelles permettant de relier une expression syntaxique à une expression sémantique et inversement. Les règles, dont différentes catégories existent, sont combinées tantôt pour identifier un sens à partir de la perception d'une phrase, tantôt pour émettre une phrase dans le but d'exprimer un sens. Des catégorisations sémantiques et syntaxiques sont exprimées également sous forme de règles, afin de réduire la complexité de la combinaison des règles ; ces catégories résultent de la confrontation de l'appareil sensori-moteur d'un robot aux tentatives de communication. La suite du travail de [Steels 05] explore l'effet d'un mécanisme d'abduction sur les formes de grammaires obtenues avec le formalisme de FCG.

Positionnement : nous avons un but distinct de [Steels 05] dans le sens où nous ne cherchons pas à déterminer les mécanismes fondamentaux d'émergence de grammaire, mais à construire un outil de gestion des sens d'interprétation, avec une expressivité convenable pour les situations d'assistance recherchées, et en situation d'interaction avec l'acteur humain. L'utilisation de résultats ultérieurs de cette communauté n'est pas exclue, mais cela ne permet pas à l'heure actuelle

de combler le *fossé* mentionné ci-dessus.

Tout en gardant une approche constructiviste, il est envisageable de mettre à la disposition de l'utilisateur une bibliothèque de contraintes d'expression prédicative (telle que SUJET - RELATION - OBJET), qui fassent sens pour l'agent alter ego. Cette proposition peut être vue comme un compromis entre l'approche traditionnelle, qui considère que le langage est le résultat de capacité cognitives supérieures de l'homme, et l'approche constructiviste, qui défend l'idée que le langage résulte des tentatives de communication entre agents immergés. L'utilisation dans un premier temps de contraintes prédicatives prédéfinies peut à terme être substituée par des mécanismes d'émergence.

Le besoin de laisser à l'utilisateur le contrôle de son agent alter ego implique de concevoir une interface graphique de représentation et d'acquisition de signature qui fournisse un accès ergonomique à l'expression grammaticale de la signature. Pour cette interface, il serait intéressant de faciliter l'interaction et l'appropriation en effectuant un mapping d'un discours "à haute voix" construit par l'utilisateur et la forme grammaticale utilisable par l'agent alter ego. Dans ce sens, une première approche existe déjà dans la configuration des règles de protection des logiciels pare-feux ou antivirus : une représentation en hyper-texte reformule les paramètres d'une règle en un discours et les liens du discours hyper-texte permettent à l'utilisateur d'acceder au détail des différents paramètres, la modification d'un paramètre conduit à une reformulation de la règle. Cette approche peut être combinée à une représentation graphique de la règle, qui permettrait d'exprimer facilement la composition de signatures et des contraintes, telles que l'enchaînement temporel, la synchronisation, le parallélisme.

Nous avons rappelé ci-dessus le double rôle d'une signature, i.e., identifier une situation et exprimer une assistance. Nous avons schématiquement distingué une partie problème d'une situation d'assistance et une partie solution. Ces termes viennent du RàPC, mais ils n'ont pas de correspondance directe en RàPET. En effet, dans ce dernier paradigme, nous nous intéressons à des tâches non prédéfinies et notre approche vise la capture en interaction d'une description d'identification et de proposition. Si la description d'une identification peut être comprise et jugée par l'utilisateur, l'énonciation de la proposition est plus difficile ; ceci est principalement dû au fait que l'utilisateur cherche une suggestion, il n'est donc par principe pas en mesure de définir les éléments qu'il lui faut. D'autre part notre approche fait l'hypothèse forte de pouvoir décrire la proposition à l'aide d'éléments symboliques. Il serait intéressant d'introduire des mécanismes d'annotation, qui relâcheraient cette hypothèse en permettant à l'utilisateur d'exprimer une proposition directement en langue naturelle ; cette expression peut être enrichie par la possibilité d'ajouter des références aux éléments de la signature, afin de les rendre perceptibles par l'agent alter ego.

La composition de signatures, l'ajout de contraintes, l'introduction de formes grammaticales élémentaires et de mécanismes d'annotation demandent d'étudier un modèle pour décrire une signature. Dans le but d'immerger l'agent alter ego dans la tâche de l'utilisateur, ce modèle doit permettre à la fois l'interprétation de la situation par l'agent alter ego afin d'identifier des sens d'assistance et l'expression, à partir du sens identifié, d'une action d'assistance à l'utilisateur ; dans ce sens, le modèle de FCG est une approche opérationnelle particulièrement intéressante. Cependant, notre objectif d'assistance à un acteur humain, en lui laissant le contrôle de son assistant, rend difficile l'introduction de mécanismes de discrimination propres à l'agent alter ego. En effet, cela reviendrait à rechercher des capacités humaines. Il est toutefois envisageable de doter l'agent d'une proactivité dans l'ajout de contraintes, qui seraient candidates à l'acceptation ou au rejet par l'utilisateur. La recherche de ces contraintes pourraient être faite en dotant l'agent alter ego de mécanismes d'inférence, l'acteur humain restant seul juge des sens à considérer et de leur expression.

10.5 Vers la réalisation complète du cycle de RàPET

Dans notre module de RàPET, nous avons considéré uniquement les deux premières étapes du cycle , i.e., l'élaboration et la remémoration, qui permettent à l'agent alter ego de proposer des situations analogues. Cette première approche ne tient pas compte de la capacité, qu'ont les traces rapprochées, à être réutilisées par l'acteur humain, ni de la possibilité d'améliorer les connaissances de domaine à partir des traces.

Dans le cadre du RàPC, [Cordier 07a] propose une approche d'acquisition des connaissances du domaine à partir des échecs d'adaptation du système. Ce travail vise à combler le fossé entre les connaissances disponibles au système, via la modélisation du domaine, et celles de l'expert. En effet, ce dernier est supposé fournir une solution cohérente avec les connaissances du domaine dont il dispose, mais les solutions peuvent être incohérentes avec celles de l'expert. Sur la base d'un processus interactif, il est proposé d'acquérir des connaissances sur un tel conflit. Une autre situation d'échec de l'adaptation vient d'une solution partielle, conduisant naturellement à l'ajout de connaissances de l'expert.

D'autre part, [Cordier 07b] s'intéressent à l'acquisition interactive de connaissances d'adaptation, en adoptant une démarche opportuniste, revenant à intégrer un cycle d'adaptation en interaction avec l'expert. La démarche adoptée est motivée par la volonté de réduire l'effort d'ingénierie de la connaissance en ce concentrant sur les avis de l'expert fourni en situation de résolution de problème. Dans notre présentation des travaux sur l'adaptation faite en 3.2.2, nous avons fait référence à [Cordier 06] qui proposait une typologie des connaissances de RàPC et montrait en quoi les connaissances d'adaptation sont liées. Dans [Cordier 07b], il est fait l'hypothèse qu'une adaptation est décrite par une *méthode d'adaptation*, composée d'un ensemble d'opérations d'adaptation élémentaires. La recherche d'un cas source est faite à l'aide d'une mesure prenant en compte les connaissances d'adaptabilité. La solution obtenue est soumise à l'expert et il est proposé une méthode incrémentale pour tester et réviser les opérations d'adaptations jusqu'à l'obtention d'une méthode d'adaptation valable pour le cas cible. Les méthodes d'adaptation ainsi générées sont mémorisées en les associant aux cas qu'elles couvrent.

Ces travaux sont particulièrement intéressants dans le cas de notre agent alter ego, car il permettraient de conditionner la recherche de situation antérieures en fonction de la capacité qu'il a à effectuer une action d'assistance, i.e., une adaptation au sens général, avec ces traces.

10.6 Validation expérimentale du modèle d'assistance et des principes d'acquisition d'expérience

Le projet OSCAR nous a orienté au départ de la thèse sur la réalisation d'un démonstrateur utilisable en situation réelle. Il est apparu le problème théorique difficile d'acquisition de sens, qui conditionne la pertinence de l'interprétation de situation par un agent assistant immergé dans la tâche de l'utilisateur. Cela nous a amené à développer une application cible "jouet" et un premier démonstrateur qui couple les mécanismes d'acquisition et de suggestion d'expérience en situation avec un module de négociation de sens, afin de construire un langage lexical.

Le démonstrateur ne permet pas une utilisation dans une situation réelle telle que l'activité de conception intégrée, car l'application cible est une application "jouet", sans les fonctionnalités qui seraient utiles pour l'activité mais qui sont sans rapport avec l'évolution de langage. Des expérimentations dans ce sens permettraient d'évaluer la pertinence du paradigme d'émergence de langage pour l'appropriation d'un système d'assistance par un acteur d'une communauté. Il

serait ainsi possible de tester la dynamique sémiotique de l'agent alter ego. Cela permettrait d'évaluer les limites pratiques de notre modèle de langage, où une signature est simplement un ensemble de symboles. Une telle évaluation pourrait mettre en évidence les relations entre symboles qui sont à prendre en compte de façon prioritaire dans l'expression des signatures.

Destiné à un environnement documentaire, le démonstrateur présenté reste néanmoins assez général et d'autres activités pourraient être choisies pour une validation. Le terrain de recherche de la co-conception continue dans l'équipe avec le projet ANR PROCOGEC[3], le Projet GOSPI[4] du cluster Rhône-Alpes. Une bonne partie des travaux seront poursuivis sur ces terrains pour y intégrer les mécanismes de négociation du sens dans un objectif de partage de connaissances hétérogènes.

[3]PROgiciel COllaboratif de GEstion des Connaissances
[4]http ://www.cluster-gospi.fr/

Chapitre 11

Conclusion

Ce travail avait pour objectif, dans le cadre du projet OSCAR, la construction d'un assistant personnalisé permettant la réutilisation de l'expérience individuelle pour l'activité de conception intégrée. Cet assistant est associé à une application à base documentaire servant d'environnement informatique commun à une communauté de travail. Cet assistant doit être en mesure de capturer l'expérience individuelle, sans que cela nécessite une action de la part d'un acteur de cette communauté, et de proposer en contexte des rapprochements pertinents avec des situations antérieures. C'est dans cette optique que l'assistant permet de capitaliser l'expérience.

Le chapitre 2 synthétise les aspects relevant de la problématique de capitalisation des connaissances dans l'entreprise ; l'activité de conception intégrée est présentée, et le modèle SG3C qui est à la base du projet OSCAR est détaillé. A partir de cela, nous avons mis en évidence les aspects sur lesquels nous nous sommes concentrés, i.e., l'expérience en situation.

Dans le chapitre 3, nous présentons le paradigme de raisonnement à partir de cas, et nous discutons de son rôle et aussi de ses limites dans l'évolution des approches d'IA qui tendent à passer de systèmes expert à des systèmes d'assistance. Le chapitre 4 part du constat des limites du RàPC et les met en relation avec les limites que rencontrent les méthodes à base de profils. Ces dernières sont aussi présentes dans le domaine des agents interface, que nous présentons et qui nous intéresse pour construire notre assistant personnalisé ; nous mettons cela en perspective avec la problématique plus générale des systèmes d'aide. Une évolution est alors présentée, elle consiste à passer d'une approche de profils d'utilisateur à des profils d'utilisation, afin de répondre à l'écart qui existe dans tout système entre sa logique de conception et les logiques de son utilisation. Dans ce sens, le raisonnement à partir d'expérience tracée est présenté, c'est une évolution du RàPC à des cas non structurés a priori, i.e., des traces d'utilisation ; à cela est adossé un modèle d'agent assistant pour manipuler ces traces. La problématique fondamentale sous-jacente à cette approche est d'avoir à disposition des signatures de tâches permettant d'identifier des situation d'assistance et donc d'effectuer des rapprochements pertinents. Dans le but de co-construire ces sens partagés par l'acteur humain et son assistant, nous présentons les travaux sur l'émergence de langage.

La deuxième partie de ce mémoire est consacrée à notre contribution dans le domaine de l'assistance à la réutilisation de l'expérience tracée, et plus particulièrement à la co-construction, selon une transposition des principes d'émergence de langage, de sens partagés. Ces sens n'existent que dans la mesure où ils permettent à l'agent alter ego d'assurer des rapprochements pertinents avec des situations antérieures. Le chapitre 6 décrit dans un premier temps notre mise en

œuvre du modèle MUSETTE, en le basant sur un mécanisme de RàPET. La modélisation des traces d'utilisation de l'application cible est présentée. Au chapitre 7, nous détaillons la transposition des mécanismes d'émergence de langage à la négociation de sens entre l'acteur humain et son agent assistant personnalisé. Nous avons adopté une description RDF pour représenter et manipuler l'ensemble des modèles que nous introduisons. Enfin, le chapitre 9 décrit notre démonstrateur pour illustrer les principes présentés précédemment, nous l'appliquons à un environnement à base documentaire qui constitue un démonstrateur simplifié d'application métier, telle que SIMMANAGER ; néanmoins, ce démonstrateur illustre les fonctionnalités les plus usuelles de ces environnements, et fait seulement recours à des objets standards pour les interfaces graphiques qui sont proposées.

La dernière partie traite des perspectives de notre travail. Nous avons précisé au chapitre 5 que notre travail sur la réutilisation de l'expérience individuelle, avec pour condition d'efficacité un partage de sens entre l'acteur humain et son alter ego informatique, s'inscrivait dans un contexte collectif. Ceci constitue la première étape avant le partage et l'échange d'expériences individuelles, puis in fine le partage d'expérience collective. De façon complémentaire, l'enrichissement de la sémantique des signature est à étudier ; se limitant dans notre étude simplement à un ensemble non ordonné de symboles, il est envisageable d'utiliser les approches issues des travaux sur l'émergence de grammaire, afin d'être en mesure d'utiliser les propriétés de notre structure de trace permettant à l'agent alter ego de percevoir et de tenir un "discours" plus sophistiqué.

Enfin, plus généralement, ce travail constitue un modèle pour la capture de sens partagés entre un utilisateur et un assistant informatique immergé dans sa tâche, en vue d'un partage de sens entre utilisateurs via le système. Il est plus général que l'application à un système à base d'expérience tracée que nous avons faite, car il peut s'appliquer dès lors que le mécanisme de négociation de sens s'appuie sur une démarche constructiviste et située.

Bibliographie

[Aamodt 94] Agnar Aamodt & Enric Plaza. *Case-Based Reasoning : Foundational Issues, Methodological Variations, and System Approaches.* AI Communications, vol. 7, no. 1, pages 39–59, mars 1994.

[Bachimont 04] Bruno Bachimont. *Pourquoi n'y-a-til pas d'expérience en ingénierie des connaissances ?* In Matta Nada, editeur, Actes de IC'2004, pages 53–64, Lyon, mai 2004.

[Baizet 04] Yoan Baizet. *La gestion des connaissances en Conception - Application à la simulation numérique chez Renault-DIEC.* PhD thesis, Université Joseph Fourier - Grenoble 1, 2004.

[Ballay 97] J.F. Ballay. *Capitaliser et transmettre les savoir-faire de l'entreprise.* Collection de la Direction des Etudes et Recherche d'Electricité de France, 1997.

[Balmisse 02] G. Balmisse. Gestion des connaissances - Outils et applications du knowledge management, entreprendre informatique. Edition Vuibert, 2002.

[Bonneaud 04] S. Bonneaud, G. Ripoche & J.-P. Sansonnet. *A Socio-Cognitive Model for the characterization of schemes of Interaction in Distributed Collectives.* In Distributed Collective Practice : Building new Directions for Infrastructural Studies - Workshop of the CSCW'04 conference, Chicago, USA, 2004.

[Bouchon-Meunier 96] Bernadette Bouchon-Meunier, Maria Rifqi & Sylvie Bothorel. *Towards general measures of comparison of objects.* Fuzzy sets and systems, vol. 84, no. 2, pages 111–116, 1996.

[Bourguin 05] Gregory Bourguin & Alain Derycke. *Systèmes interactifs en Co-évolution - Réflexions sur les apports de la Théorie de l'Activité au support des Pratiques Collectives Distribuées.* Interaction Homme Machine, vol. 6, no. 1, 2005.

[Champin 01] Pierre-Antoine Champin. RDF *Tutorial.* W3C, avril 2001.

[Champin 02] Pierre-Antoine Champin. *Modéliser l'expérience pour assister la réutilisation - De la Conception Assistée par Ordinateur au Web Sémantique.* PhD thesis, Université Claude Bernard Lyon 1 - EDIIS, 2002.

[Champin 03] Pierre Antoine Champin, Yannick Prié & Alain Mille. *Musette : Modelling Uses and Tasks for Tracing Exprerience.* In B. Fuchs, editeur, Workshop From structured cases to unstructured problem solving episodes - WS 5 of ICCBR'03, Trondheim, 2003.

[Chi 81] Michelene T.H. Chi, P.J. Feltovitch & Robert Glaser. *Categorization of physics problems by experts and novices.* Cognitive Science, vol. 5, pages 121–152, 1981.

[Cordier 06] Amélie Cordier, Béatrice Fuchs & Alain Mille. *Engineering and Learning of Adaptation Knowledge in Case-Based Reasoning.* In Lecture Notes in Artificial Intelligence, pages 303–317, 2006.

[Cordier 07a] Amélie Cordier, Béatrice Fuchs, Jean Lieber & Alain Mille. *Acquisition des connaissances du domaine d'un système de RàPC : une approche fondée sur l'analyse interactive des échecs d'adaptation - le système FRAKAS.* In Plateforme AFIA, editeur, Atelier de Raisonnement à Partir de Cas, Grenoble, France, 2007.

[Cordier 07b] Amélie Cordier, Béatrice Fuchs, Jean Lieber & Alain Mille. *Acquisition interactive des connaissances d'adaptation intégrée aux sessions de raisonnement à partir de cas - Principes, architecture IAKA et prototype KAYAK.* In Plateforme AFIA, editeur, Atelier de Raisonnement à Partir de Cas, Grenoble, France, 2007.

[De Boer 97] B. De Boer. *Self-Organisation in Vowel Systems through Imitation.* In P. Husbands & I. Harvey, editeurs, Fourth European Conference On Artificial Life, Cambridge, Ma., 1997.

[Delgado 99] Joaquin Delgado & Naohiro Ishii. *Formal Models for Learning User Preferences, A Preliminary Report.* In International Joint Conference on Artificial Intelligence (IJCAI-99), 1999.

[Delgado 00] Joaquin Delgado. *Agent-based information filtering and recommender systems.* PhD thesis, Nagoya Institute of Technology, mar 2000.

[Egyed-Zsigmond 03] Elöd Egyed-Zsigmond. *Gestion des Connaissances dans une base de documents multimédias.* PhD thesis, Institut National des Sciences Appliquées de Lyon - EDIIS, 2003.

[Ermine 96] J.L. Ermine, M. Chaillot, P. Bigeon, B. Charreton & D. Malavieille. *MKSM : Méthode pour la gestion des connaissances.* Ingénierie des systèmes d'information, vol. 4, pages 541–575, 1996.

[Ermine 01] J.L. Ermine. Les processus de la gestion des connaissances, pages 17–30. Hermès Science Publications, 2001.

[Euzenat 95] Jérôme Euzenat & François Rechenmann. SHIRKA, *10 ans, c'est* TROPES *?* In Amedeo Napoli, editeur, Langages et modèles à objets, pages 13–34, LORIA, Nancy, octobre 1995.

[Fuchs 01] Béatrice Fuchs, Jean Lieber, Alain Mille & Amedeo Napoli. *Un algorithme pour la phase d'adaptation du raisonnement à partir de cas.* In Andreas Herzig, editeur, Journées nationales sur les modèles de raisonnement, pages 79–92, Arras (FR), mai 2001.

[Gapenne 06] Olivier Gapenne. *Introduction : Relation d'aide et transformation cognitive.* Intellectica, vol. 44, no. 2, pages 7–16, 2006.

[Gentner 91] Dedre Gentner & Kenneth D. Forbus. *MAC/FAC : A Model of Similarity-based Retrieval.* In Annual Conference of the Cognitive Science Society, pages 504–509. Cognitive Science Society, Cincinnati, OH (US), 1991.

[Georgeon 06] Olivier Georgeon, Alain Mille & Thierry Bellet. *Analyzing behavioral data for refining cognitive models of operator.* In IEEE Computer Society, editeur, Philosophies and Methodologies for Knowledge Discovery - Seventeenth International Workshop on Database and Expert Systems Applications, pages 588–592, Krakow, Poland, 2006.

[Gorodetski 02] Vladimir Gorodetski & Igor Kotenko. *Attacks against Computer Network : Formal Grammar-based Framework and Simulation Tool.* In Proceedings of the 5th International Conference "Recent Advances in Intrusion Detection", pages 219–238, Zrich, Switzerland, october 2002. Springer Verlag.

[Grundstein 00] M. Grundstein. *Repérer et mettre en valeur les connaissances cruciales pour l'entreprise.* In 10e Congrès International de l'AFAV, Paris, 2000.

[Hammond 90] Kristian J. Hammond. *Case-Based Planning : A Framework for Planning from Experience.* Cognitive Science, vol. 14, no. 3, pages 385–443, 1990.

[Hanney 96] Kathleen Hanney. Learning Adaptation Rules from Cases. Msc thesis, Trinity College, Dublin (IE), 1996.

[Kifer 95] Michael Kifer, Georg Lausen & James Wu. *Logical Foundations of Object-Oriented and Frame-Based Languages.* Journal of the ACM, vol. 42, pages 741–843, 1995.

[Kolodner 83] Janet Kolodner. *Reconstructive Memory : A Computer Model.* Cognitive Science, vol. 7, no. 4, pages 281–328, 1983.

[Lashkari 94] Yezdi Lashkari, Max Metral & Pattie Maes. *Collaborative Interface Agents.* In Twelfth National Conference on Artificial Intelligence, 1994.

[Leake 97] David B. Leake, Andrew Kinley & David Wilson. *Case-Based Similarity Assessment : Estimating Adaptability from Experience.* In National Conference on Artificial Intelligence, Rhode Island Convention Center, Providence, RI (US), juillet 1997. American Association for Artificial Intelligence, Menlo Park, CA (US).

[Lebowitz 83] Michael Lebowitz. *Memory-Based Parsing.* Artificial Intelligence, vol. 21, no. 4, pages 363–404, 1983.

[Leray 07a] David Leray & Jean-Paul Sansonnet. *Acquisition de connaissances perceptives pour un agent assistant.* In IC'2007, Grenoble, France, 2007.

[Leray 07b] David Leray & Jean-Paul Sansonnet. *Assisting Dialogical Agents Modeled from Novice User's Perceptions.* In LNAI 4693, editeur, 11th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems, KES'07, pages 1122–1129, Grenoble, France, 2007.

[Lieber 98] Jean Lieber & Amedeo Napoli. *Correct and Complete Retrieval for Case-Based Problem-Solving.* In Henri Prade, editeur, European Conference on Artificial Intelligence, pages 68–72, Brighton (GB), aot 1998. John Wiley & Sons Ltd, Chichester (GB).

[Lieber 02] Jean Lieber. *Recopier c'est déjà adapter : six types d'adaptation par copie.* In Atelier Raisonnement à Partir de Cas, INSERM, Faculté de Médecine Broussais Hotel Dieu, Paris (FR), mai 2002.

[Lieberman 95] Henry Lieberman. *Letizia : An Agent That Assists Web Browsing.* In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, pages 924–929, 1995.

[Lieberman 96] Henry Lieberman & David Maulsby. *Instructible Agents : Software that Just Keeps Getting Better.* IBM Systems Journal, vol. 35, no. 3/4, pages 539–556, 1996.

[Lieberman 97] Henry Lieberman. *Autonomous Interface Agents.* ACM Conference on Human-Computer Interface [CHI-97], Atlanta, March 1997, 1997.

[Lieberman 98] Henry Lieberman. *Integrating User Interface Agents with Conventional Applications.* In ACM Conference on Intelligent User Interfaces, San Francisco, 1998.

[Maes 94] Pattie Maes. *Agents that Reduce Work and Information Overload.* Communications of the ACM, vol. 37, pages 30–40, 146, 1994.

[Martin 98] Francisco J. Martin, Enric Plaza & Josep L. Arcos. *Knowledge and experience reuse through communication among competent (peer) agents.* International Journal of Software Engineering and Knowledge Engineering, 1998.

[Martin 99] F. J. Martin & E. Plaza. *Auction-based Retrieval.* In Proceedings of the "2n Congrés Català d'Intel.ligència Artificial", pages 136–145, 1999.

[Martín 04] Francisco J. Martín & Enric Plaza. *Ceaseless Case-Based Reasoning.* In Peter Funk & Pedro A. González-Calero, editeurs, ECCBR, volume 3155 of *LNCS*, pages 287–301, 2004.

[Mille 98] Alain Mille. *Experience et expertise : les connaissances mobilisées en coopération homme-machine pour la résolution de problème.* Habilitation à diriger des recherches, Université Claude Bernard Lyon I, Lyon (FR), 1998.

[Mille 05] Alain Mille, Guy Caplat & Mick Philippon. *Faciliter les activités des utilisateurs d'environnements informatiques : quoi, quand, comment ?* In , 2005.

[Mille 06a] Alain Mille. Raisonner à partir de l'expérience tracée (ràpet) - définition, illustration et résonnances avec le "story telling". 2006.

[Mille 06b] Alain Mille & Yannick Prié. *Une théorie de la trace informatique pour faciliter l'adaptation dans la confrontation logique d'utilisation/logique de conception.* In Rochebrune, pages 183–195, Megève, 2006.

[Napoli 97] Amedeo Napoli. *Une introduction aux logiques de descriptions.* Rapport de Recherche RR 3314, INRIA, Nancy (FR), 1997.

[Nonaka 95] Ikujiro Nonaka & Hirotaka Takeuchi. The Knowledge Creating Company. Oxford University Press, Oxford (GB), 1995.

[Ontanon 03] Santiago Ontanon & Enric Plaza. *Collaborative Case Retention Strategies for CBR Agents.* pages 392–406, Trondheim, Norway, june 2003. Springer Verlag.

[Pachet 97] François Pachet. *Représentation de connaissances et langages à objets.* Habilitation à diriger des recherches 97-21, LIP6, Paris (FR), 1997.

[Plaza 95] Enric Plaza. *Cases as terms : A feature term approach to the structured representation of cases.* In Manuela M. Veloso & Agnar Aamodt, editeurs, International Conference on Case-Based Reasonning, numéro 1010 in Lecture Notes in Computer Science, pages 265–276, Sesimbra (PT), octobre 1995. Springer Verlag, Berlin (DE).

[Plaza 97] E. Plaza, J.L. Arcos & F. Martin. *Cooperative Case-Based Reasoning.* In Lecture Notes in Artificial Intelligence, pages 180–201, 1997.

[Plaza 01] E. Plaza & S. Ontanon. *Ensemble Case-based Reasoning : Collaboration Policies for Multiagent Cooperative CBR.* In In Case-Based Reasoning Research and Development : ICCBR 2001 - Lecture Notes in Artificial Intelligence 2080, pages 437–451, 2001.

[Prasad 96] M.V. Nagendra Prasad & E. Plaza. *Corporate Memories as Distributed Case Libraries.* In Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-based Systems Workshop, pages 1–19, 1996.

[Prax 00] J.Y. Prax. Le guide du knowledge management : concepts et pratiques du management des connaissances. Edition Dunod, 2000.

[Richter 95] Michael M. Richter. *The Knowledge Contained in Similarity Measures.* Conférence invitée, ICCBR'95, Sesimbra (PT), octobre 1995.

[Rifqi 00] Maria Rifqi, Vincent Berger & Bernadette Bouchon-Meunier. *Discrimination power of measures of comparison.* Fuzzy sets and systems, vol. 110, no. 2, pages 189–196, 2000.

[Ripoche 03] Gabriel Ripoche, Jean-Paul Sansonnet & Les Gasser. *Extraction des pratiques collectives distribuées à partir des traces interactionnelles langagières.* In Workshop COMETE - Action spécifique CNRS, Paris, France, 2003.

[Roth-Berghofer 01] Thomas Roth-Berghofer & Ioannis Iglezakis. *Six Steps in Case-Based Reasoning : Towards a maintenance methodology for case-based reasoning systems.* pages 198–208, 2001.

[Rousset 00] Stéphane Rousset. *Les conceptions "système unique" de la mémoire aspects théoriques.* Revue de Neuropsychologie, vol. 10, no. 1, pages 27–52, 2000.

[Sabouret 02] Nicolas Sabouret. *Étude de modèles de représentations, de requêtes et de raisonnement sur le fonctionnement des composants actifs pour l'interaction homme-machine.* PhD thesis, LIMSI-CNRS, Université Paris XI, dec 2002.

[Sanderson 94] Penelope M. Sanderson & Carolanne Fisher. *Exploratory Sequential Data Analysis : Foundations.* Human-Computer Interactions, vol. 9, 1994.

[Sansonnet 04] Jean-Paul Sansonnet. *Composants dialogiques génériques : perspectives et méthodes pour une approche intégrée des outils assistants langagiers et de la programmation objet.* In Revue "L'objet", editeur, Conference "Langages et Modèles à Objets", LMO 04, Lille, France, 2004.

[Schank 77] Roger C. Schank & Robert P. Abelson. Scripts, Plans, Goals and Understanding : an Inquiry into Human Knowledge Structures. Lawrence Erlbaum Associates, Mahwah, NJ (US), 1977.

[Schank 82] Roger C. Schank. Dynamic Memory : a Theory of Learning in Computers and People. Cambridge University Press, Cambridge (GB), 1982.

[Sebag 93] Michèle Sebag & Marc Shoenauer. *A Rule Based Similarity Measure.* In 1st European Workshop On Case-Based Reasoning, volume 837 of *Lecture Notes in Artificial Intelligence*, pages 65–70, University of Kayserslautern, (DE), 1993. Springer Verlag, Berlin (DE).

[Selker 94] Ted Selker. *COACH : A Teaching Agent that Learns.* ACM Communications, vol. 37, no. 7, pages 92–99, 1994.

[Smyth 95a] Barry Smyth & Mark T. Keane. *Remembering to Forget : a Competence-preserving Case Deletion Policy for Case Based Reasoning Systems.* In C.S. Mellish, editeur, International Joint Conference On Artificial Intelligence, volume 1, pages 377–382, Montréal, Québec (CA), aot 1995. Morgan Kaufmann, San Mateo, CA (US).

[Smyth 95b] Barry Smyth & Matk T. Keane. *Retrieval and Adaptation in Déjà Vu, a Case-Based Reasoning System for Software Design.* In David W. Aha & Ashwin Ram, editeurs, AAAI Fall Symposium on Adaptation of Knowledge for Reuse, MIT Campus, Cambridge, MA (US), novembre 1995. American Association for Artificial Intelligence, Menlo Park, CA (US).

[Sowa 99] John Sowa. Knowledge Representation : Logical, Philosophical, and Computational Foundations. PWS Publishing Co., Pacific Grove, CA (US), 1999.

[Steels 99] Luc Steels. The Talking Head Experiment, volume 1 - words and meanings. Special pre-edition for LABORATORIUM, Antwerpen, 1999.

[Steels 00] Luc Steels. *Language as a complex adaptive system.* Lecture Notes in Computer Science. Parallel Problem Solving from Nature - PPSN, vol. 4, 2000.

[Steels 05] Luc Steels. *The Role of Construction Grammar in Fluid Language Grounding.* To appear in Elsevier Science, 2005.

[Stuber 01] Arnaud Stuber & Christine Solnon. *Dopage des algorithmes à base de fourmis par des techniques de fouille de données.* In Journées Nationales sur la résolution Pratique de problèmes NP-Complets (JNPC 01), Toulouse, France, 2001.

[Stuber 03] Arnaud Stuber, Salima Hassas & Alain Mille. *Combining MAS and Experience Reuse for assisting collective task achievement.* In Lorraine McGinty, editeur, Workshop 'From structured cases to unstructured problem solving episodes for Experience-based Assistance' - 5th International Conference on Case-based Reasoning (ICCBR'03), Trondheim, Norway, 2003.

[Stuber 05] Arnaud Stuber, Salima Hassas & Alain Mille. *Language games for meaning negotiation between human and computer agents.* In Springer Verlag LNAI 3963, editeur, Engineering Societies in Agents' World (ESAW'05), Kusadasi, Aydin, Turkey, 2005.

[Trousse 99] B. Trousse, M. Jaczynski & R. Kanawati. *Une approche fondée sur le raisonnement à partir de cas pour l'aide à la navigation dans un hypermédia.* In Product, Tools and Methods (H2PTM'99), Paris, France, 1999.

[Troussier 99] Nadège Troussier. *Contribution à l'intégration du calcul mécanique dans la conception de produits techniques : proposition méthodologique pour l'utilisation et la réutilisation.* PhD thesis, Université Joseph Fourier - Grenoble 1, octobre 1999.

[Truong 86] Jean-Michel Truong, Alain Bonnet & Jean-Paul Haton. Systèmes experts. InterEditions, Paris, 1986.

[Tsuchiya 95] S. Tsuchiya. *Commensurability, a key concept of bfusiness re-engineering.* In 3rd International Symposium of the Management of the Information and Corporate Knowledge, Institut National pour l'Intelligence Artificielle, Compiègne, 1995.

[Turner 04] W. A. Turner, G. Ripoche, Jean-Paul Sansonnet & Les Gasser. *Confidence-Based Organizational Metrics.* In Distributed Collective Practice : Building new Directions for Infrastructural Studies - Workshop of the CSCW'04 conference, Chicago, USA, 2004.

[Tversky 77] Amos Tversky. *Features of Similarity.* Psychological Review, vol. 84, no. 4, pages 327–352, 1977.

[Valencia 00] Erika Valencia. *Outils de topologie algébrique pour la gestion de l'hétérogénéité sémantique entre agents dialogiques.* PhD thesis, LIMSI-CNRS, Université Paris XI, dec 2000.

[van Diggelen 05] Jurriaan van Diggelen, Robert Jan Beun, Franck Dignum, Rogier M van Eijk & J.-J.Ch. Meyer. *A decentralized approach for establishing a shared communication vocabulary.* In International workshop on agent mediated knowledge management 2005, held with AAMAS2005, 2005.

[Voss 96] Angi Voss. *Principles of Case Reusing Systems.* In Ian Smith & Boi Faltings, editeurs, European Workshop on Case Based Reasoning, volume 1168 of *Lecture Notes in Computer Science*, Lausanne (CH), novembre 1996. Springer Verlag, Berlin (DE).

[Wexelblat 99] Alan Wexelblat & Pattie Maes. *Footprints : History-Rich Tools for Information Foraging.* In ACM SIGCHI Conference on Human Factors in Computing Systems, Pittsburgh, Pennsylvania, USA, 1999.

[Whittlesea 87] Bruce W.A. Whittlesea. *Preservation of Specific Experiences in the Representation of General Knowledge.* Journal of Experimental Psychology : Learning, Memory and Cognition, vol. 13, no. 1, pages 3–17, 1987.

[Wilke 98] Wolfgang Wilke & Ralph Bergmann. *Techniques and Knowledge Used for Adaptation During Case-Based Problem Solving.* In Angel P. del Pobil, Jose Mira & Moonis Ali, editeurs, International Conference On Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, volume 1416 of *Lecture Notes in Computer Science*, pages 497–506, Castellón (ES), juin 1998. Springer Verlag, Berlin (DE).

Annexes

Annexe A

Descriptions RDF

A.1 Modèle d'application

```
<rdf:RDF
    xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#"
    xmlns:application="http://localhost/application/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:topOnto="http://localhost/topOnto/"
    xmlns:domain="http://localhost/domain/"
    wmlns:language="http://localhost/language">


<topOnto:IntermStep rdf:about="&application;tag_gainFocus"
 topOnto:intermStepLabel="gainFocus"/>
<topOnto:InitStep rdf:about="&application;tag_openDossier"
 topOnto:initStepLabel="openDossier"/>
<topOnto:FinalStep rdf:about="&application;tag_closeDossier"
 topOnto:finalStepLabel="closeDossier"/>


<topOnto:InitStep rdf:about="&application;tag_startEdition"
 topOnto:initStepLabel="startEdition"/>
<topOnto:FinalStep rdf:about="&application;tag_stopEdition"
 topOnto:finalStepLabel="stopEdition"/>
<topOnto:FinalStep rdf:about="&application;tag_cancelEdition"
 topOnto:finalStepLabel="cancelEdition"/>
<topOnto:PermStep rdf:about="&application;tag_input"
 topOnto:permStepLabel="input"/>


<topOnto:InitStep rdf:about="&application;tag_startViewing"
 topOnto:initStepLabel="startViewing"/>
<topOnto:FinalStep rdf:about="&application;tag_stopViewing"
 topOnto:finalStepLabel="stopViewing"/>
```

```
</rdf:RDF>
```

A.2 Observation de trace

```
<!-- Description de la hierarchie d'observation --!>

  <rdf:Description rdf:about="&trace;trcX_rootContainer">
    <rdf:type rdf:resource="&topOnto;Container"/>
    <topOnto:contains rdf:resource="&trace;trcX_container_a"/>
    <topOnto:contains rdf:resource="&trace;trcX_container_b"/>
  </rdf:Description>

  <rdf:Description rdf:about="&trace;trcX_container_a">
    <rdf:type rdf:resource="&topOnto;Container"/>
    <topOnto:containerLabel>a</topOnto:containerLabel>
    <topOnto:contains rdf:resource="&trace;trcX_Capture_1"/>
  </rdf:Description>

  <rdf:Description rdf:about="&trace;trcX_container_b">
    <rdf:type rdf:resource="&topOnto;Container"/>
    <topOnto:containerLabel>b</topOnto:containerLabel>
    <topOnto:contains rdf:resource="&trace;trcX_Capture_2"/>
    <topOnto:contains rdf:resource="&trace;trcX_container_c"/>
    <topOnto:contains rdf:resource="&trace;trcX_container_d"/>
  </rdf:Description>

  <rdf:Description rdf:about="&trace;trcX_container_c">
    <rdf:type rdf:resource="&topOnto;Container"/>
    <topOnto:containerLabel>c</topOnto:containerLabel>
    <topOnto:contains rdf:resource="&trace;trcX_Capture_3"/>
  </rdf:Description>

  <rdf:Description rdf:about="&trace;trcX_container_d">
    <rdf:type rdf:resource="&topOnto;Container"/>
    <topOnto:containerLabel>d</topOnto:containerLabel>
    <topOnto:contains rdf:resource="&trace;trcX_Capture_4"/>
    <topOnto:contains rdf:resource="&trace;trcX_Capture_5"/>
  </rdf:Description>


<!-- Captures des objets d'intérêt --!>

  <rdf:Description rdf:about="&trace;trcX_Capture_1">
    <rdf:type rdf:resource="&topOnto;Capture"/>
    <topOnto:captureOf rdf:resource="&domain;mot_A"/>
  </rdf:Description>
```

```
  <rdf:Description rdf:about="&trace;trcX_Capture_2">
    <rdf:type rdf:resource="&topOnto;Capture"/>
    <topOnto:captureOf rdf:resource="&domain;mot_B"/>
  </rdf:Description>

  <rdf:Description rdf:about="&trace;trcX_Capture_3">
    <rdf:type rdf:resource="&topOnto;Capture"/>
    <topOnto:captureOf rdf:resource="&domain;mot_C"/>
  </rdf:Description>

  <rdf:Description rdf:about="&trace;trcX_Capture_4">
    <rdf:type rdf:resource="&topOnto;Capture"/>
    <topOnto:captureOf rdf:resource="&domain;mot_D"/>
  </rdf:Description>

  <rdf:Description rdf:about="&trace;trcX_Capture_5">
    <rdf:type rdf:resource="&topOnto;Capture"/>
    <topOnto:captureOf rdf:resource="&domain;mot_A"/>
  </rdf:Description>


<!-- Objets d'intérêt observés --!>

  <rdf:Description rdf:about="&domain;mot_A">
    <rdf:type rdf:resource="&topOnto;Keyword"/>
    <topOnto:keywordLabel>Label mot_A</topOnto:keywordLabel>
  </rdf:Description>

  <rdf:Description rdf:about="&domain;mot_B">
    <rdf:type rdf:resource="&topOnto;Keyword"/>
    <topOnto:keywordLabel>Label mot_B</topOnto:keywordLabel>
  </rdf:Description>

  <rdf:Description rdf:about="&domain;mot_C">
    <rdf:type rdf:resource="&topOnto;Keyword"/>
    <topOnto:keywordLabel>Label mot_C</topOnto:keywordLabel>
  </rdf:Description>

  <rdf:Description rdf:about="&domain;mot_D">
    <rdf:type rdf:resource="&topOnto;Keyword"/>
    <topOnto:keywordLabel>Label mot_D</topOnto:keywordLabel>
  </rdf:Description>
```

A.3 Pattern

```
<!-- Description de la hierarchie d'observation --!>
```

```
  <rdf:Description rdf:about="&language;symbX_rootContainer">
    <rdf:type rdf:resource="&topOnto;Container"/>
    <topOnto:contains rdf:resource="&language;symbX_container_a"/>
    <topOnto:contains rdf:resource="&language;symbX_container_b"/>
  </rdf:Description>

  <rdf:Description rdf:about="&language;symbX_container_a">
    <rdf:type rdf:resource="&topOnto;Container"/>
    <topOnto:containerLabel>a</topOnto:containerLabel>
    <topOnto:contains rdf:resource="&language;symbX_Capture_1"/>
  </rdf:Description>

  <rdf:Description rdf:about="&language;symbX_container_b">
    <rdf:type rdf:resource="&topOnto;Container"/>
    <topOnto:containerLabel>b</topOnto:containerLabel>
    <topOnto:contains rdf:resource="&language;symbX_Capture_2"/>
    <topOnto:contains rdf:resource="&language;symbX_container_d"/>
  </rdf:Description>

  <rdf:Description rdf:about="&language;symbX_container_d">
    <rdf:type rdf:resource="&topOnto;Container"/>
    <topOnto:containerLabel>d</topOnto:containerLabel>
    <topOnto:contains rdf:resource="&language;symbX_Capture_3"/>
    <topOnto:contains rdf:resource="&language;symbX_Capture_4"/>
  </rdf:Description>


<!-- Captures des objets d'intérêt, avec abstraction --!>

  <rdf:Description rdf:about="&language;symbX_Capture_1">
    <rdf:type rdf:resource="&topOnto;Capture"/>
    <topOnto:captureOf rdf:resource="&domain;mot_A"/>
    <topOnto:MATCHING_ID></topOnto:MATCHING_ID>
  </rdf:Description>

  <rdf:Description rdf:about="&language;symbX_Capture_2">
    <rdf:type rdf:resource="&topOnto;Capture"/>
    <topOnto:captureOf rdf:resource="&domain;mot_B"/>
    <topOnto:MATCHING_ID></topOnto:MATCHING_ID>
  </rdf:Description>

  <rdf:Description rdf:about="&language;symbX_Capture_3">
    <rdf:type rdf:resource="&topOnto;Capture"/>
    <topOnto:captureOf rdf:resource="&domain;mot_D"/>
    <topOnto:MATCHING_ID></topOnto:MATCHING_ID>
  </rdf:Description>

  <rdf:Description rdf:about="&language;symbX_Capture_4">
```

```
    <rdf:type rdf:resource="&topOnto;Capture"/>
    <topOnto:captureOf rdf:resource="&domain;mot_A"/>
    <topOnto:MATCHING_ID>*1</topOnto:MATCHING_ID>
  </rdf:Description>


<!-- Objets d'intérêt observés --!>

  <rdf:Description rdf:about="&domain;mot_A">
    <rdf:type rdf:resource="&topOnto;Keyword"/>
    <topOnto:keywordLabel>Label mot_A</topOnto:keywordLabel>
  </rdf:Description>

  <rdf:Description rdf:about="&domain;mot_B">
    <rdf:type rdf:resource="&topOnto;Keyword"/>
    <topOnto:keywordLabel>Label mot_B</topOnto:keywordLabel>
  </rdf:Description>

  <rdf:Description rdf:about="&domain;mot_D">
    <rdf:type rdf:resource="&topOnto;Keyword"/>
    <topOnto:keywordLabel>Label mot_D</topOnto:keywordLabel>
  </rdf:Description>
```

A.4 Structure de trace : exemple

```
<rdf:RDF
    xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#"
    xmlns:application="http://localhost/application/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:topOnto="http://localhost/topOnto/"
    xmlns:trace="http://localhost/trace/"
    xmlns:action="http://localhost/action/">


  <rdf:Description rdf:about="&trace;trcX">
    <rdf:type rdf:resource="&topOnto;Trace"/>
    <topOnto:fisrtObservation rdf:resource="&trace;trcX_state_0"/>
    <topOnto:lastObservation rdf:resource="&trace;trcX_state_8"/>
  </rdf:Description>


  <rdf:Description rdf:about="&trace;trcX_state_0">
    <rdf:type rdf:resource="&topOnto;State"/>
    <topOnto:obsRoot rdf:resource="&trace;trcX_rootContainer_0.0"/>
  </rdf:Description>

  <rdf:Description rdf:about="&trace;trcX_transition_0">
```

```
    <rdf:type rdf:resource="&topOnto;Transition"/>
    <topOnto:targetDossier rdf:resource="&trace;trcX_Capture_0"/>
    <topOnto:targetDocument rdf:resource="&trace;trcX_Capture_1"/>
    <topOnto:obsRoot rdf:resource="&trace;trcX_rootContainer_0.1"/>
    <topOnto:previousState rdf:resource="&trace;trcX_state_0"/>
    <topOnto:associatedActionInstance rdf:resource="&action;actionInstance_Y"/>
    <topOnto:hasForTag rdf:resource="&application;tag_startViewing"/>
  </rdf:Description>

  <rdf:Description rdf:about="&trace;trcX_state_1">
    <rdf:type rdf:resource="&topOnto;State"/>
    <topOnto:associatedActionInstance rdf:resource="&action;actionInstance_Y"/>
    <topOnto:previousTransition rdf:resource="&trace;trcX_transition_0"/>
    <topOnto:obsRoot rdf:resource="&trace;trcX_rootContainer_1.0"/>
  </rdf:Description>

  <rdf:Description rdf:about="&trace;trcX_transition_1">
    <rdf:type rdf:resource="&topOnto;Transition"/>
    <topOnto:targetDossier rdf:resource="&trace;trcX_Capture_2"/>
    <topOnto:targetDocument rdf:resource="&trace;trcX_Capture_3"/>
    <topOnto:obsRoot rdf:resource="rdf:resource="&trace;trcX_rootContainer_1.1"/>
    <topOnto:previousState rdf:resource="&trace;trcX_state_1"/>
    <topOnto:associatedActionInstance rdf:resource="&action;actionInstance_Z"/>
    <topOnto:hasForTag rdf:resource="&application;tag_startEdition"/>
  </rdf:Description>

  <rdf:Description rdf:about="&trace;trcX_state_2">
    <rdf:type rdf:resource="&topOnto;State"/>
    <topOnto:associatedActionInstance rdf:resource="&action;actionInstance_Z"/>
    <topOnto:previousTransition rdf:resource="&trace;trcX_transition_1"/>
    <topOnto:obsRoot rdf:resource="&trace;trcX_container2.0"/>
  </rdf:Description>

  <rdf:Description rdf:about="&trace;trcX_transition_2">
    <rdf:type rdf:resource="&topOnto;Transition"/>
    <topOnto:targetDossier rdf:resource="&trace;trcX_Capture_4"/>
    <topOnto:targetDocument rdf:resource="&trace;trcX_Capture_5"/>
    <topOnto:obsRoot rdf:resource="&trace;trcX_container_2.1"/>
    <topOnto:previousState rdf:resource="&trace;trcX_state_2"/>
    <topOnto:associatedActionInstance rdf:resource="&action;actionInstance_Y"/>
    <topOnto:hasForTag rdf:resource="&application;tag_gainFocus"/>
  </rdf:Description>

  <rdf:Description rdf:about="&trace;trcX_state_3">
    <rdf:type rdf:resource="&topOnto;State"/>
    <topOnto:associatedActionInstance rdf:resource="&action;actionInstance_Y"/>
    <topOnto:previousTransition rdf:resource="&trace;trcX_transition_2"/>
    <topOnto:obsRoot rdf:resource="&trace;trcX_container_3.0"/>
```

```
</rdf:Description>

<rdf:Description rdf:about="&trace;trcX_transition_3">
  <rdf:type rdf:resource="&topOnto;Transition"/>
  <topOnto:targetDossier rdf:resource="&trace;trcX_Capture_5"/>
  <topOnto:targetDocument rdf:resource="&trace;trcX_Capture_6"/>
  <topOnto:obsRoot rdf:resource="&trace;trcX_container_3.1"/>
  <topOnto:previousState rdf:resource="&trace;trcX_state_3"/>
  <topOnto:associatedActionInstance rdf:resource="&action;actionInstance_Z"/>
  <topOnto:hasForTag rdf:resource="&application;tag_gainFocus"/>
</rdf:Description>

<rdf:Description rdf:about="&trace;trcX_state_4">
  <rdf:type rdf:resource="&topOnto;State"/>
  <topOnto:associatedActionInstance rdf:resource="&action;actionInstance_Z"/>
  <topOnto:previousTransition rdf:resource="&trace;trcX_transition_3"/>
  <topOnto:obsRoot rdf:resource="&trace;trcX_container_4.0"/>
</rdf:Description>

<rdf:Description rdf:about="&trace;trcX_transition_4">
  <rdf:type rdf:resource="&topOnto;Transition"/>
  <topOnto:targetDossier rdf:resource="&trace;trcX_Capture_7"/>
  <topOnto:targetDocument rdf:resource="&trace;trcX_Capture_8"/>
  <topOnto:obsRoot rdf:resource="&trace;trcX_container_4.1"/>
  <topOnto:previousState rdf:resource="&trace;trcX_state_4"/>
  <topOnto:associatedActionInstance rdf:resource="&action;actionInstance_Z"/>
  <topOnto:hasForTag rdf:resource="&application;tag_input"/>
</rdf:Description>

<rdf:Description rdf:about="&trace;trcX_state_5">
  <rdf:type rdf:resource="&topOnto;State"/>
  <topOnto:associatedActionInstance rdf:resource="&action;actionInstance_Z"/>
  <topOnto:previousTransition rdf:resource="&trace;trcX_transition_4"/>
  <topOnto:obsRoot rdf:resource="&trace;trcX_container_5.0"/>
</rdf:Description>

<rdf:Description rdf:about="&trace;trcX_transition_5">
  <rdf:type rdf:resource="&topOnto;Transition"/>
  <topOnto:targetDossier rdf:resource="&trace;trcX_Capture_9"/>
  <topOnto:targetDocument rdf:resource="&trace;trcX_Capture_10"/>
  <topOnto:obsRoot rdf:resource="&trace;trcX_container_5.1"/>
  <topOnto:previousState rdf:resource="&trace;trcX_state_5"/>
  <topOnto:associatedActionInstance rdf:resource="&action;actionInstance_Z"/>
  <topOnto:hasForTag rdf:resource="&application;tag_stopEdition"/>
</rdf:Description>

<rdf:Description rdf:about="&trace;trcX_state_6">
  <rdf:type rdf:resource="&topOnto;State"/>
```

```
    <topOnto:associatedActionInstance rdf:resource="&action;actionInstance_Z"/>
    <topOnto:previousTransition rdf:resource="&trace;trcX_transition_5"/>
    <topOnto:obsRoot rdf:resource="&trace;trcX_container_6.0"/>
  </rdf:Description>

  <rdf:Description rdf:about="&trace;trcX_transition_6">
    <rdf:type rdf:resource="&topOnto;Transition"/>
    <topOnto:targetDossier rdf:resource="&trace;trcX_Capture_11"/>
    <topOnto:targetDocument rdf:resource="&trace;trcX_Capture_12"/>
    <topOnto:obsRoot rdf:resource="&trace;trcX_container_6.1"/>
    <topOnto:previousState rdf:resource="&trace;trcX_state_6"/>
    <topOnto:associatedActionInstance rdf:resource="&action;actionInstance_Y"/>
    <topOnto:hasForTag rdf:resource="&application;tag_gainFocus"/>
  </rdf:Description>

  <rdf:Description rdf:about="&trace;trcX_state_7">
    <rdf:type rdf:resource="&topOnto;State"/>
    <topOnto:associatedActionInstance rdf:resource="&action;actionInstance_Y"/>
    <topOnto:previousTransition rdf:resource="&trace;trcX_transition_6"/>
    <topOnto:obsRoot rdf:resource="&trace;trcX_container_7.0"/>
  </rdf:Description>

  <rdf:Description rdf:about="&trace;trcX_transition_7">
    <rdf:type rdf:resource="&topOnto;Transition"/>
    <topOnto:targetDossier rdf:resource="&trace;trcX_Capture_13"/>
    <topOnto:targetDocument rdf:resource="&trace;trcX_Capture_14"/>
    <topOnto:obsRoot rdf:resource="&trace;trcX_container_7.1"/>
    <topOnto:previousState rdf:resource="&trace;trcX_state_7"/>
    <topOnto:associatedActionInstance rdf:resource="&action;actionInstance_Y"/>
    <topOnto:hasForTag rdf:resource="&application;tag_stopEdition"/>
  </rdf:Description>

  <rdf:Description rdf:about="&trace;trcX_state_8">
    <rdf:type rdf:resource="&topOnto;State"/>
    <topOnto:previousTransition rdf:resource="&trace;trcX_transition_7"/>
    <topOnto:obsRoot rdf:resource="&trace;trcX_container_8.0"/>
  </rdf:Description>



--------------------------------------------------------------------------------

  <rdf:Description rdf:about="&trace;trcX_Capture_0">
    <rdf:type rdf:resource="&topOnto;Capture"/>
    <topOnto:captureOf rdf:resource="http://localhost/dossier/dossier_A"/>
  </rdf:Description>

  <rdf:Description rdf:about="http://localhost/dossier/dossier_A">
    <rdf:type rdf:resource="&topOnto;Dossier"/>
```

```
  </rdf:Description>

  <rdf:Description rdf:about="&trace;trcX_Capture_1">
    <rdf:type rdf:resource="&topOnto;Capture"/>
    <topOnto:captureOf rdf:resource="http://localhost/document/document_B"/>
  </rdf:Description>

  <rdf:Description rdf:about="http://localhost/document/document_B">
    <rdf:type rdf:resource="&topOnto;Ordre"/>
  </rdf:Description>

</rdf:RDF>
```

A.5 Structure de symbole : exemple

```
<rdf:RDF
    xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#"
    xmlns:application="http://localhost/application/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:topOnto="http://localhost/topOnto/"
    xmlns:document="http://localhost/document/"
    xmlns:domain="http://localhost/domain/"
    wmlns:language="http://localhost/language"
    xmlns:community="http://localhost/community/">

  <rdf:Description rdf:about="&language;symbole_X">
    <rdf:type rdf:resource="&topOnto;Symbol"/>
    <topOnto:targetDocument rdf:resource="&language;symbX_Capture_1"/>
    <topOnto:targetDossier rdf:resource="&language;symbX_Capture_0"/>
    <topOnto:hasForTag rdf:resource="&application;tag_input"/>
    <topOnto:initialState rdf:resource="&language;symbX_state_0"/>
    <topOnto:symbolTransition rdf:resource="&language;symbX_transition_1"/>
    <topOnto:finalState rdf:resource="&language;symbX_state_1"/>
    <topOnto:symbolLabel>Label du symbole X</topOnto:symbolLabel>
  </rdf:Description>


  <rdf:Description rdf:about="&language;symbX_state_0">
    <rdf:type rdf:resource="&topOnto;State"/>
    <topOnto:obsRoot rdf:resource="&language;symbX_container_0"/>
  </rdf:Description>

  <rdf:Description rdf:about="&language;symbX_transition_1">
    <rdf:type rdf:resource="&topOnto;Transition"/>
    <topOnto:targetDocument rdf:resource="&language;symbX_Capture_1"/>
    <topOnto:targetDossier rdf:resource="&language;symbX_Capture_0"/>
    <topOnto:hasForTag rdf:resource="&application;tag_input"/>
```

```
    <topOnto:obsRoot rdf:resource="&language;symbX_container_1"/>
    <topOnto:previousState rdf:resource="&language;symbX_state_0"/>
  </rdf:Description>

  <rdf:Description rdf:about="&language;symbX_state_1">
    <topOnto:obsRoot rdf:resource="&language;symbX_container_2"/>
    <topOnto:previousTransition rdf:resource="&language;symbX_transition_1"/>
    <rdf:type rdf:resource="&topOnto;State"/>
  </rdf:Description>

--------------------------------------------------------------------------------

  <rdf:Description rdf:about="&language;symbX_Capture_0">
    <rdf:type rdf:resource="&topOnto;Capture"/>
    <topOnto:captureOf rdf:resource="&document;dossier_A"/>
  </rdf:Description>

  <rdf:Description rdf:about="&document;dossier_A">
    <rdf:type rdf:resource="&topOnto;Dossier"/>
  </rdf:Description>

  <rdf:Description rdf:about="&language;symbX_Capture_1">
    <rdf:type rdf:resource="&topOnto;Capture"/>
    <topOnto:captureOf rdf:resource="&document;document_B"/>
  </rdf:Description>

  <rdf:Description rdf:about="&document;document_B">
    <rdf:type rdf:resource="&topOnto;Ordre"/>
  </rdf:Description>


</rdf:RDF>
```

A.6 Etat de langage : exemple

```
<rdf:RDF
    xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#"
    xmlns:application="http://localhost/application/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:topOnto="http://localhost/topOnto/"
    xmlns:domain="http://localhost/domain/"
    xmlns:language="http://localhost/language"
    xmlns:signature="http://localhost/signature/">

  <rdf:Description rdf:about="&language;language_test">
    <rdf:type rdf:resource="&topOnto;Language"/>
    <topOnto:languageComposition rdf:resource="&language;signature_X"/>
```

```
    <topOnto:languageOrigin></topOnto:languageOrigin>
  </rdf:Description>


  <rdf:Description rdf:about="&language;signature_X">
    <rdf:type rdf:resource="&topOnto;Signature"/>
    <topOnto:signatureComposition rdf:resource="&language;signComp_Y"/>
    <topOnto:signatureLabel>Label de signatureX</topOnto:signatureLabel>
    <topOnto:signaturePonderation>1.0</topOnto:signaturePonderation>
    <topOnto:signatureOrigin></topOnto:signatureOrigin>
  </rdf:Description>


  <rdf:Description rdf:about="&language;signComp_Y">
    <rdf:type rdf:resource="&topOnto;SignatureComponent"/>
    <topOnto:representedSymbol rdf:resource="&language;symbole_Z"/>
    <topOnto:componentPonderation>1.0</topOnto:componentPonderation>
  </rdf:Description>


  <rdf:Description rdf:about="&language;symbole_Z">
    <rdf:type rdf:resource="&topOnto;Symbol"/>
    <topOnto:symbolComposition rdf:resource="&language;symbCont_a"/>
    <topOnto:symbolLabel>Label de symbole_Z</topOnto:symbolLabel>
    <topOnto:symbolPonderation>1.0</topOnto:symbolPonderation>
    <topOnto:symbolTag rdf:resource="http://localhost/application/tag_input"/>
    <topOnto:symbolDocumentType rdf:resource="&topOnto;Ordre"/>
    <topOnto:symbolTargetContainer>container_b</topOnto:symbolTargetContainer>
    <topOnto:symbolOrigin> </topOnto:symbolOrigin>
  </rdf:Description>


  <rdf:Description rdf:about="&language;symbCont_a">
    <rdf:type rdf:resource="&topOnto;SymbolContent"/>
    <topOnto:representedKeyword rdf:resource="&domain;mot_c"/>
  </rdf:Description>


  <rdf:Description rdf:about="&domain;mot_c">
    <rdf:type rdf:resource="&topOnto;Keyword"/>
  </rdf:Description>

</rdf:RDF>
```

Annexe B

Démonstrateur : Diagrammes de classes UML

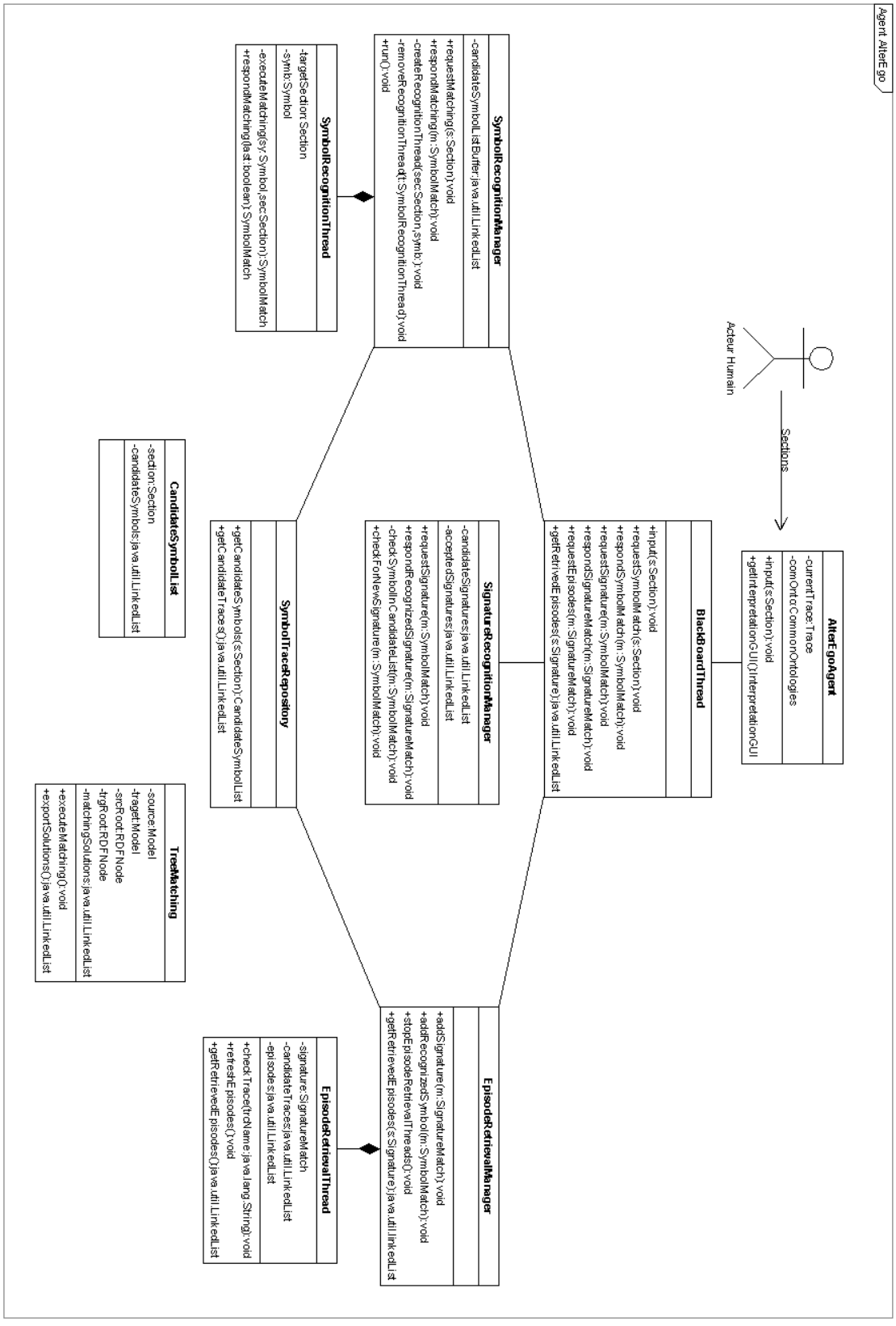


FIG. B.1 – Agent alter ego : diagramme de classes

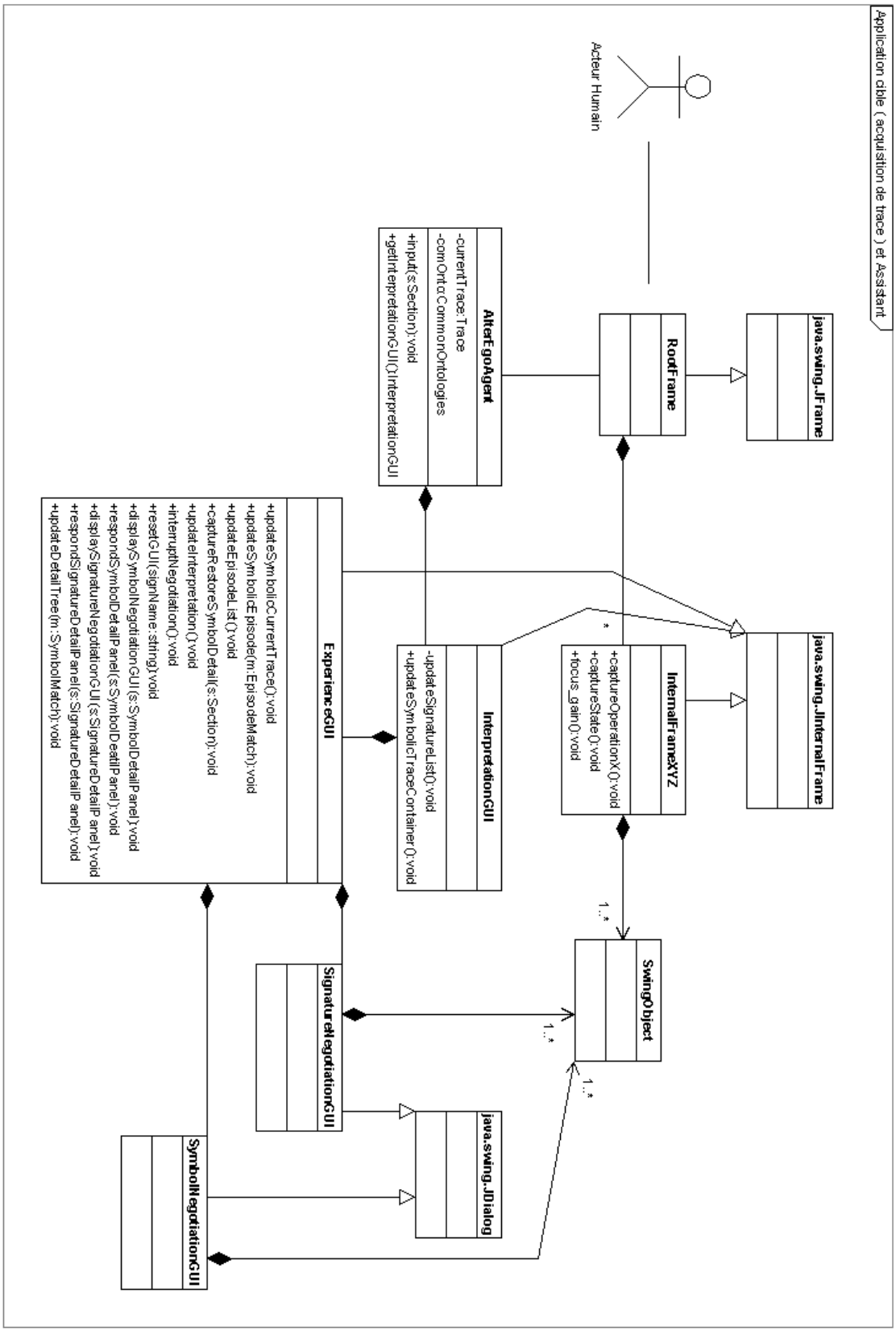


FIG. B.2 – Acquisition de trace : diagramme de classes

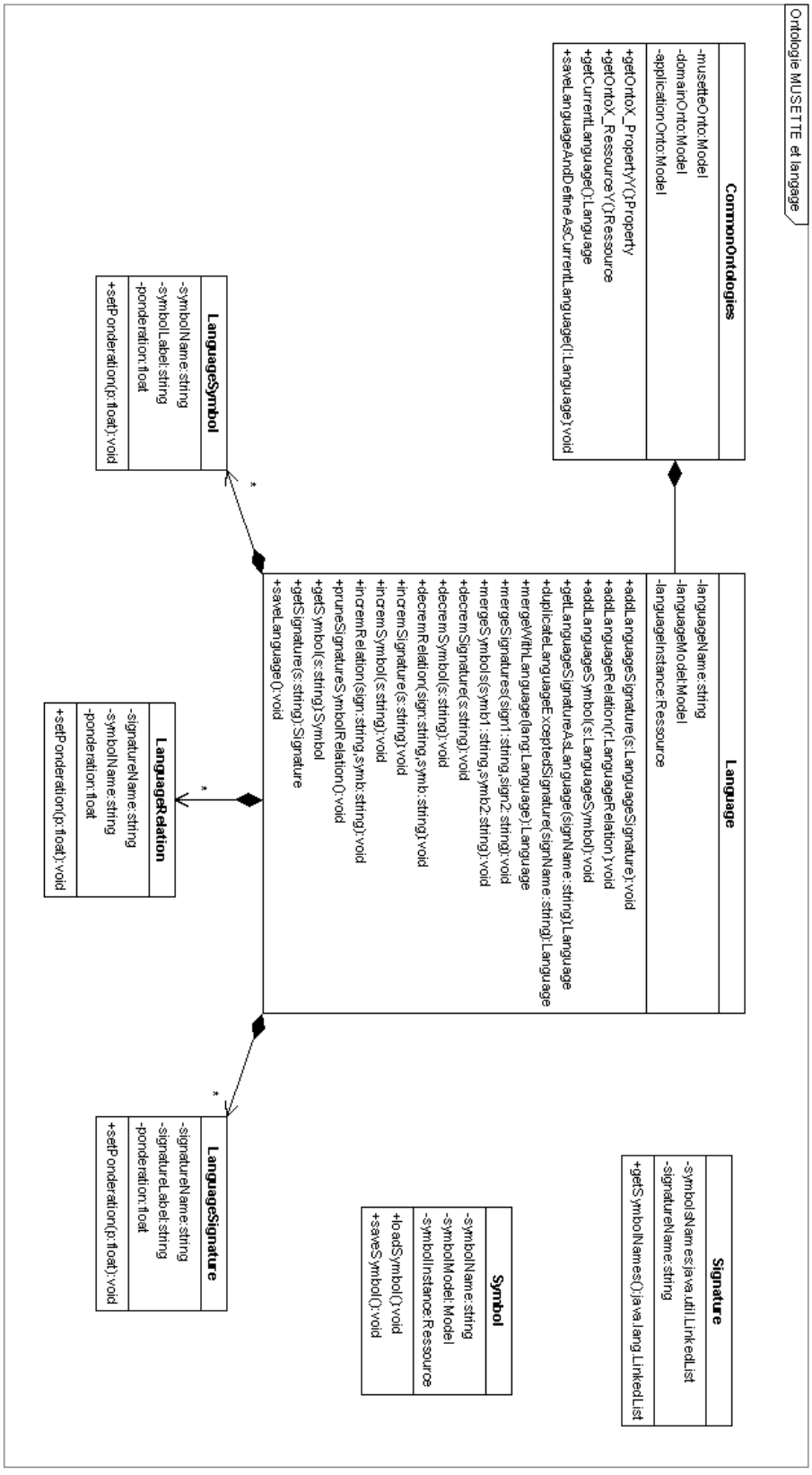


FIG. B.3 – Ontologies et langage : diagramme de classes

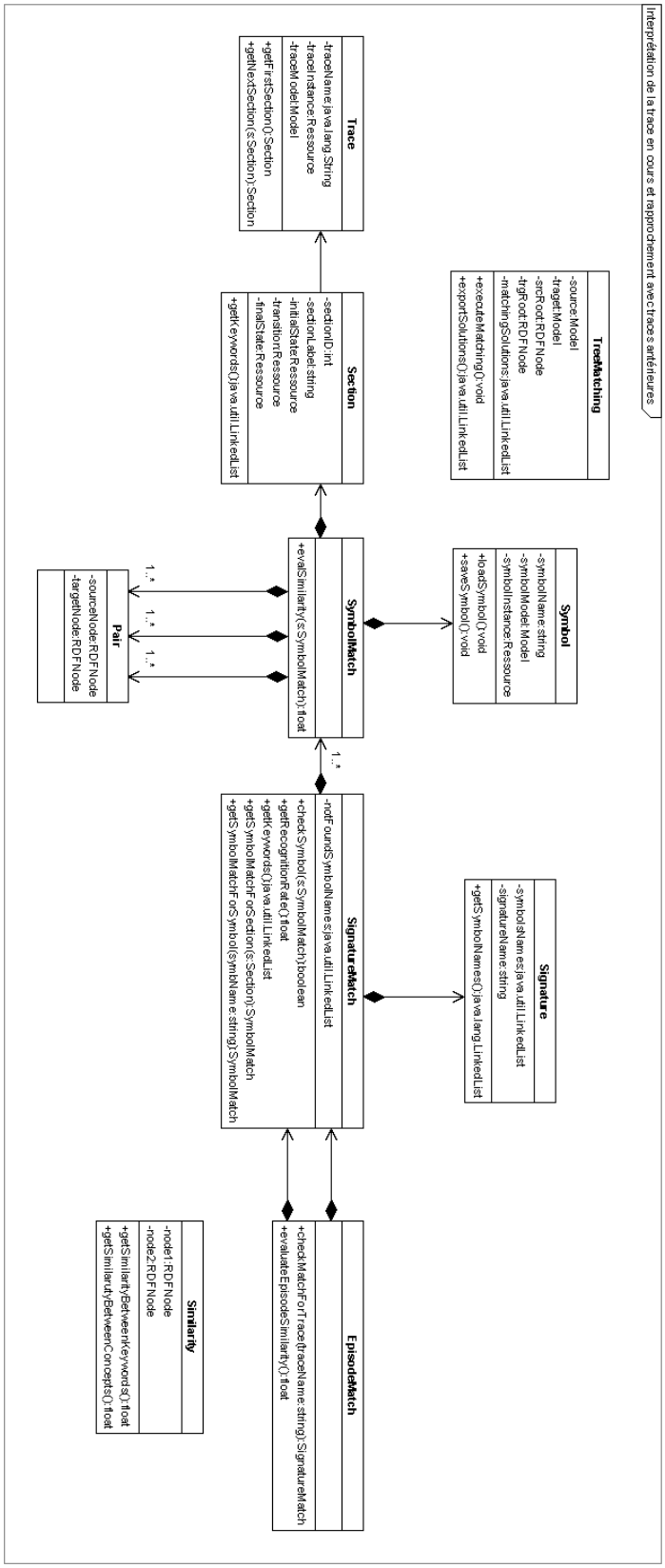


FIG. B.4 – Interprétation et rapprochement : diagramme de classes

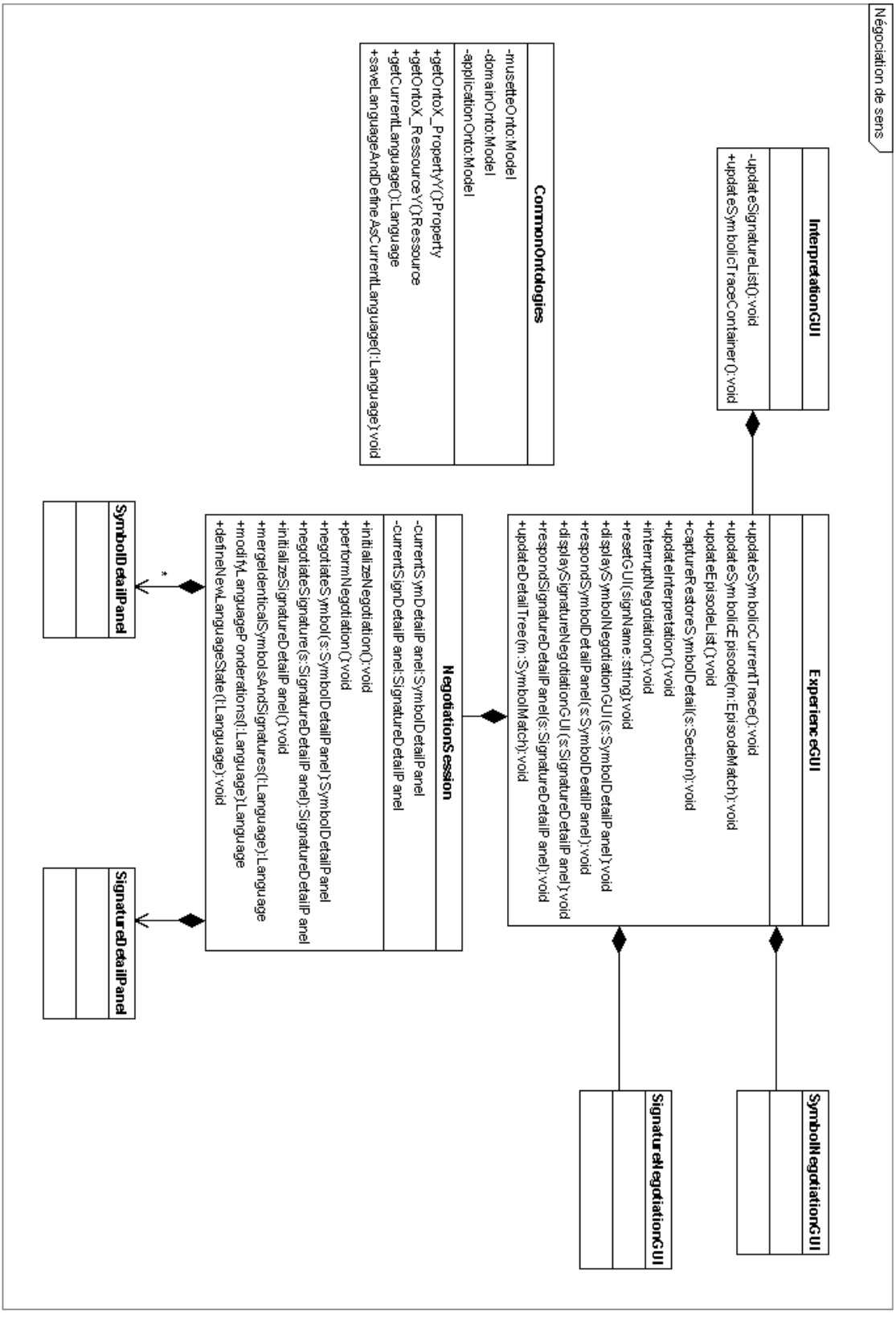


FIG. B.5 – Négociation de sens : diagramme de classes

Table des figures

Table des équations

Résumé

Le travail présenté se place dans la problématique de la capitalisation des connaissances au sein d'une communauté d'acteurs humains interagissant avec un outil informatique commun, support de leur activité. Notre approche est d'associer un agent assistant personnalisé à chaque acteur humain. Un tel agent assistant est chargé d'effectuer des suggestions de traces d'expérience qui sont proches de la situation courante rencontrée par l'acteur humain.

Pour traiter des situations qui ne peuvent être définies a priori, nous nous sommes concentré sur la problématique de co-construction de sens par négociation entre l'acteur humain et son agent assistant, afin d'élaborer des connaissances issues de l'expérience tracée et permettant de la remobiliser. Notre principale contribution se place dans le mécanisme de négociation de sens, celui-ci s'effectue en transposant les principes d'émergence de langage de forme lexicale, tels qu'ils sont présentés par Luc Steels. Cette approche nous a conduit à l'étude d'une reformulation symbolique des traces d'expérience, afin que l'agent assistant puisse établir, en négociant avec l'acteur humain, des expressions symboliques pour désigner les situations d'assistance.

En perspective, nous voyons l'échange d'expérience entre acteurs humains au travers du système se faisant via l'émergence de langages entre agents assistants résultants de la mise en commun de leurs expériences et langages propres. Par ailleurs, notre transposition des mécanismes d'émergence de langage de forme lexical met en perspective la problématique d'émergence de grammaire, en premier lieu pour enrichir l'expressivité du " discours " des agents assistants.

Mots-clés: Co-construction de sens, Expérience tracée, émergence de langage, négociation de sens

Abstract

This work is entering the problematic of Knowledge Management for a community of human actors, interacting with a common computer environment, support of their activity. Our approach is to associate a personal computer assistant to each actor. Such an assistant is charged to retrieve previously traced experiences which are similar to the current one.

In order to deal with situations which can not be defined beforehand, we focused on the problematic of meaning co-construction through negotiation between a human actor and his personal assistant agent. The purpose is to elaborate knowledge from traced experience so that its retrieval would be later convenient for the actor.

Our main contribution relies on a meaning negotiation mechanism, performed by transposition of the principles of emergence of lexical language. This approach led us studying the symbolic reformulation of the traced experiences, so that the assistant agent could establish symbolic expressions allowing to "talk about" the situations of assistance.

In this perspective, we plan to study the exchange of experience between human actors through the common environment, as the emergence of language resulting from the use of individual languages and traced experiences. Moreover, our transposition of the mechanisms of emergence of language, of lexical form, leads us to the problematic of the emergence of grammar, first of all to be able to enhance the expressivity of the "speech" hold by the assistant agents.

Keywords: Meaning co-construction, Traced Experience, Emergence of Language, Meaning Negotiation