Conception d'une chaîne de traitement de la langue naturelle pour un agent conversationnel assistant

François Bouchet

Conception d'une Chaîne de Traitement de la Langue Naturelle pour un Agent Conversationnel Assistant

François BOUCHET

LIMSI-CNRS
École Doctorale d'Informatique
Université Paris-Sud 11 – Orsay

Soutenue le 29 juin 2010 devant le jury composé de :

Rapporteurs :	Guy LAPALME	Professeur – Université de Montréal, RALI
	Sylvie PESTY	Professeur – Université Pierre Mendès-France, LIG
Examinateurs :	Catherine PELACHAUD	Directeur de Recherche – LTCI, Télécom ParisTech
	Anne VILNAT	Professeur – Université Paris-Sud 11, LIMSI-CNRS
Directeur :	Jean-Paul SANSONNET	Directeur de Recherche – LIMSI-CNRS

Remerciements

Bien qu'issu d'une longue phase rédactionnelle en solitaire, ce manuscrit n'aurait jamais pu voir le jour sans les années qui l'ont précédé, et je tiens donc à rendre hommage à toutes les personnes avec qui j'ai pu être amené à interagir au cours de cette période.

Dans un premier temps, mes remerciements vont bien sûr aux personnes ayant pris part à l'évaluation finale de ce travail en me faisant l'honneur d'accepter de prendre part à mon jury de thèse : merci donc à Catherine Pelachaud pour l'intérêt qu'elle a montré envers mon travail en acceptant de présider ce jury, à Sylvie Pesty et Guy Lapalme pour leurs rapports et commentaires avisés m'ayant permis d'améliorer ce manuscrit, ainsi qu'à Anne Vilnat dont le manuscrit annoté m'a permis d'ultimes retouches post-soutenance.
Un grand merci également à celui qui fut bien plus que le cinquième membre de ce jury, mon directeur de thèse, Jean-Paul Sansonnet, pour avoir cru en moi dès le début, puis pour la patience et le dévouement avec lesquelles il m'a supporté (dans tous les sens du terme) tout au long de ces années. Nos discussions, menant souvent bien au-delà de mon sujet de thèse, furent à la fois enrichissantes et motivantes, et ce fut un véritable plaisir de découvrir le monde de la recherche à ses côtés, et d'apprendre (parfois ensemble) à y survivre.

Parmi les travaux présentés dans cette thèse, plusieurs n'auraient été possibles sans collaboration avec d'autres chercheurs. Je souhaite donc remercier Michèle Sebag, pour les nombreuses références pertinentes qui ont rendu possible l'existence de toute une section du chapitre 7 sur la classification automatique, et pour ses conseils sur ma recherche de post-doc. Dans le cadre de mon court mais intense séjour brésilien, je remercie Patricia Jaques, Nicolas Maillard, Aline Villavivencio et Rosa Vicari pour leur accueil chaleureux à Porto Alegre, Evandro Manara Miletto et Marcelo Pimenta pour la collaboration fructueuse que nous avons pu établir et qui m'a donné la possibilité de découvrir un champ de recherche bien différent du mien. Merci également à Mao Xuetao pour sa collaboration sur les expériences DIVA.

Dans un groupe dotés de thématiques aussi variées qu'AMI et au sein d'un laboratoire pluridisciplinaire comme le LIMSI, les collaborations scientifiques sont parfois plus difficiles, mais l'enrichissement humain n'en est lui que plus grand. Merci donc à ceux qui sont devenus au fil des ans de vrais compagnons de route : Rami, Wai Kit, Céline, Matthieu, Issam, Ning, Chahnez, David et Jonathan ; à ceux dont j'ai eu l'occasion de partager le bureau (parfois brièvement, avec mes 3 déménagements) : Jean-Claude, Laurent, Bill, Dominique, Sarkis, Aurélie, François-Xavier, Victoria et Jorge ; à tous ceux que j'ai cotoyé d'un peu plus loin, mais toujours avec plaisir, et sans qui la vie au LIMSI n'aurait pas été tout à fait la même : Mathieu, Sonia, Cédric, Sylvain, François S., Tifanie, Anne et Flavien.

En plus de certains collègues susmentionnés qui sont devenus des amis que j'ai eu plaisir à fréquenter hors du travail, un grand merci aussi aux copains d'avant, qui se sont retrouvés un peu délaissés pendant ces années, mais dont la présence a toujours été extrêmement appréciée, notamment Julien-mon-bi, Antoine et Ludmila. Merci enfin à ma famille pour avoir su faire avec mon emploi du temps et mon humeur parfois chaotiques l'un comme l'autre : Tania, Kyra, papa, maman, tata, cousin, je n'y serai pas arrivé sans vous !

Table des matières

Table des figures

Liste des tableaux

Introduction

Problématique et objectifs

L'interaction avec la machine selon le mode de communication privilégié des humains, la langue naturelle, constitue un objectif de recherche depuis de nombreuses années. On pouvait d'ailleurs penser que ce type d'interaction serait un préalable indispensable pour que le grand public se mette à utiliser les outils informatiques.
Pourtant, force est de constater que grâce au développement des téléphones portables et à l'attrait de nombreux services disponibles sur Internet, les utilisateurs d'applications et services informatiques sont aujourd'hui majoritairement constitués d'un public de non-spécialistes, tandis que le dialogue homme-machine en langue naturelle n'est toujours pas une réalité quotidienne. Faut-il alors penser que cet objectif n'est plus une priorité ?
Dans cette thèse, nous nous intéressons à l'interaction homme-machine dans un cas particulier : l'assistance à des utilisateurs novices. En effet, si le grand public est devenu consommateur des applications et services proposés par les outils informatiques, cela ne signifie pas pour autant qu'ils sont des utilisateurs satisfaits et maîtrisant parfaitement les outils qu'ils utilisent. Au contraire, pour ne rester que dans le cadre du Web, le foisonnement de services sur celui-ci les force à être sans cesse confrontés à de nouvelles interfaces, plus ou moins bien conçues. Et la langue naturelle demeure le moyen instinctivement employé pour rechercher de l'aide dans de telles situations (par exemple via les forums).

De plus, l'utilisation ces deux dernières décennies de personnages virtuels, ou agents conversationnels animés, de plus en plus réalistes tant dans leur représentation graphique que dans la représentation interne qu'ils ont d'eux-mêmes et du monde qui les entoure, a montré que pour peu qu'ils se montrent suffisamment compétents, leur présence était généralement appréciée des utilisateurs, notamment novices. Dès lors, leur utilisation dans le cadre d'un système d'assistance à des utilisateurs novices fait sens, et c'est la raison pour laquelle nous nous proposons de concevoir un agent conversationnel qui soit dédié à la fonction d'assistance.

Dans la première partie de la thèse, nous commencerons dans le **chapitre 1** par réaliser un état de l'art des trois domaines autour desquels s'articulent les travaux présentés : l'assistance à des utilisateurs novices d'applications informatiques, le traitement de requêtes en langue naturelle et les agents conversationnels animés. Nous introduirons ensuite, dans le **chapitre 2**, le

concept d'Agent Conversationnel Assistant (ACA) et le projet DAFT dans lequel prend place les travaux réalisés dans cette thèse.

Dans une deuxième partie, nous décrirons la chaîne de traitement des requêtes d'utilisateurs novices auprès d'un ACA. Tout d'abord, dans le **chapitre 3** nous justifierons le besoin de collecter un corpus spécifique de requêtes (le corpus *Daft*) et détaillerons la méthodologie employée pour le constituer. Nous présenterons ensuite, dans le **chapitre 4**, l'analyseur syntaxico-sémantique `GRASP` défini à partir du corpus précédent, qui analyse les requêtes d'un point de vue grammatical et extrait leur sémantique lexicale. Afin d'obtenir une représentation sémantique plus structurée des requêtes, toujours à partir d'une étude basée sur le corpus *Daft*, nous spécifierons dans le cadre du **chapitre 5**, un langage formel de requêtes : le langage *DAFT*. La production automatique de requêtes exprimée dans ce langage *DAFT* fera l'objet du **chapitre 6**, où l'on exposera le fonctionnement du module `DIG` qui vient après `GRASP` dans la chaîne de traitement.

Dans une troisième partie, nous présenterons des études complémentaires ainsi que des éléments d'évaluation. Dans le **chapitre 7**, nous nous intéresserons plus particulièrement à la nature des requêtes formant le corpus *Daft* ainsi qu'aux possibilités d'analyse de celles-ci d'un point de vue plus conversationnel que ce qu'effectue la chaîne de traitement précédemment décrite. Finalement, le **chapitre 8** sera l'occasion de traiter de travaux connexes réalisés au cours de cette thèse : nous verrons ainsi un exemple concret d'intégration d'un agent conversationnel assistant, et évoquerons l'architecture nécessaire au support de l'agent rationnel $\mathtt{A}_R$ qui sera amené à traiter, dans le futur, les requêtes produites en *DAFT* par `DIG`.

Cette organisation est synthétisée de manière schématique par la figure 1 ci-dessous.

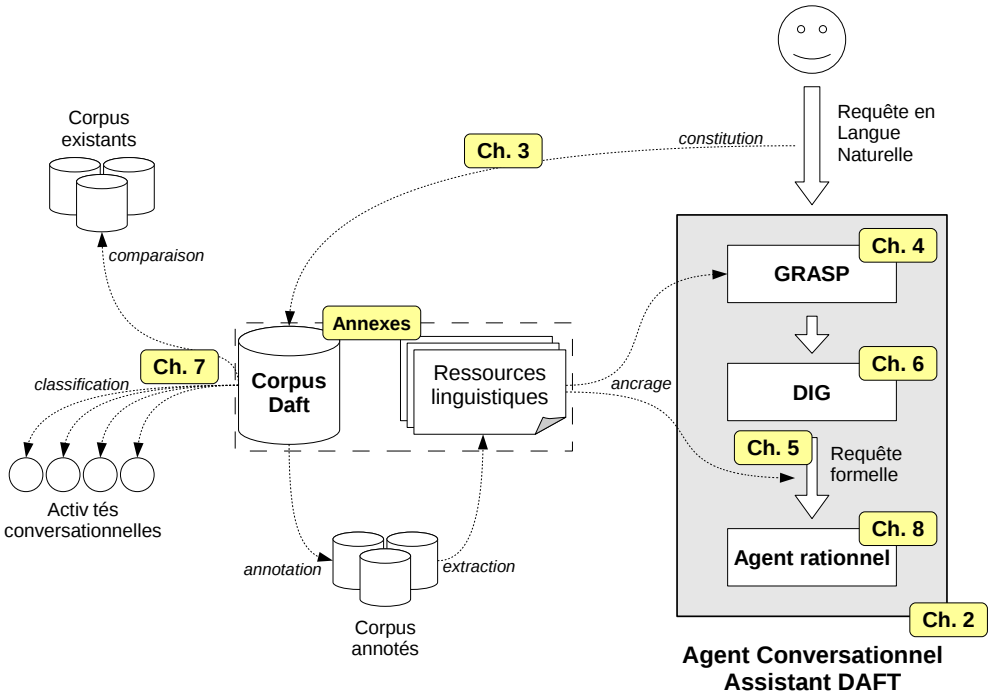


Figure 1 Organisation des chapitres 2 à 8 de la thèse (le chapitre 1 étant un état de l'art)

Notations

Tout au long de ce document, nous emploierons les conventions de notation suivantes :

- Les noms de langages ou d'applications courantes seront écrits en italique.
  Exemple : *Mathematica* ou le langage *DAFT*.

- Les différents sous-systèmes ou modules d'un système seront écrits en type *typewriter*.
  Exemple : le système `DAFT` ou le module `GRASP`.

- Les ensembles de ressources seront représentées par une lettre en alphabet *Blackboard*, éventuellement avec un indice.
  Exemple : l'ensemble des clés sémantiques noté $\mathbb{K}_G$.

- Les opérations sur un ou plusieurs élément(s) sont notées en alphabet *calligraphié*, éventuellement avec un indice lorsqu'il s'agit d'une sous-opération.
  Exemple : l'opération de combinaison de deux éléments x et y notée $\mathcal{M}(x, y)$.

- Les fonctions, au sens informatique du terme, qui implémentent certaines opérations sont écrites en fonte *sans serif*.
  Exemple : la fonction de récupération du plus grand indice écrite $\mathsf{getIndicesMax}()$.

- Les **schèmes** du langage *DAFT* apparaissent en *brun* et en type *typewriter*.
  Exemple : le marqueur `ASK[...]` ou l'action `Click[...]`.

- Lors d'une référence à un élément défini dans le glossaire (à la fin de ce document), celui-ci apparaît en *bleu et gras* à la première occurrence. Il apparaît seulement en bleu lors des références ultérieures.
  Exemple : les schèmes du langage *DAFT*.

- Lors d'une première référence à une expression exprimée par la suite sous forme d'acronyme, celui-ci apparaît en *bleu* en version complète suivi de son acronyme. La liste des acronymes utilisés figure en fin de ce document.
  Exemple : le format de document eXtended Markup Language (XML).

Première partie

État de l'art

Chapitre 1

État de l'art

Sommaire

Notre objectif général repose sur l'utilisation d'agents conversationnels afin de fournir de l'assistance à des utilisateurs **novices** d'applications informatiques. À ce titre, nous nous proposons dans ce chapitre d'effectuer une revue de l'état de l'art dans trois domaines liés au sujet d'étude de cette thèse, à savoir :

- le traitement de l'assistance dans le cadre des applications informatiques ;
- les différentes approches pouvant être employées par un Système de Dialogue Homme-Machine (SDHM), pour la gestion des requêtes langagières ;
- les apports relatifs à l'utilisation d'un Agent Conversationnel Animé (ECA)[1], en particulier vis-à-vis d'utilisateurs novices.

[1]Tout au long de cette thèse, nous utiliserons l'acronyme anglophone ECA pour désigner les Agents Conversationnels Animés, réservant l'usage d'ACA aux Agents Conversationnels Assistants, qui sont au cœur de notre travail.

1.1 Assistance aux utilisateurs d'applications informatiques

1.1.1 Le besoin d'assistance

1.1.1.1 Apports et limites des interfaces graphiques

Pendant les premières décennies de son existence, de par le faible nombre de systèmes disponibles et la complexité des méthodes d'interaction avec ceux-ci (cartes perforées, lignes de commandes...) qui nécessitent une compréhension poussée de la machine, l'informatique demeure un domaine réservé à des **experts**. La volonté d'ouverture à un public plus large apparaît au début des années 1970 avec le concept même d'interface graphique, ou Graphical User Interface (GUI), dans le cadre des travaux menés par les chercheurs du Xerox PARC (Palo Alto Research Center) qui l'implémentent pour la première fois au sein du Xerox Alto [Thacker *et al.* , 1979] en 1973 (prototype de recherche, qui sera tout de même diffusé à plus d'un millier d'exemplaires), puis du Xerox Star en 1981 (premier système commercialisé). Le Star est également le premier système créé dans l'objectif d'être destiné à « des utilisateurs occasionnels plutôt que des gens passant la plupart de leur temps sur la machine » [Johnson *et al.* , 1989]. Déjà dans l'esprit de rendre l'interface des ordinateurs plus accessible à des utilisateurs professionnels mais non experts des systèmes informatiques, Alan Kay propose pour l'Alto de faire usage de la "métaphore du bureau" (desktop metaphor) [Kay, 1977], dont le principe de base consiste à considérer l'environnement affiché à l'écran comme un bureau (physique) sur lequel sont disposés des objets (documents, dossiers...). L'usage d'une métaphore [Ortony, 1993] vise à placer l'utilisateur dans un environnement connu, dans lequel il est intuitivement capable de transposer les actions effectuées sur les objets physiques dans le monde réel aux artefacts contenus dans le monde virtuel représenté par l'interface graphique, selon ce que Carroll & Mack [1984] nomment "l'apprentissage en sachant" ("learning by knowing"). De la même manière, le "principe de la manipulation directe" introduit par Shneiderman [1983] visait à s'affranchir de l'apprentissage du vocabulaire et de la syntaxe d'un langage de commandes complexe pour réaliser des tâches sur la machine. Dans ces deux cas, l'idée sous-jacente est que la réduction de l'effort cognitif nécessaire à l'emploi de l'outil informatique afin de permettre :

- aux utilisateurs de se focaliser sur ce qu'ils veulent faire plutôt que sur la façon de le faire, ce qui rejoint la philosophie même des travaux d'Engelbart [1962], qu'il voyait comme un moyen d'améliorer « les capacités de traitement des informations basiques aux besoins, problèmes et progrès de la société ».
- d'élargir la base même de ces utilisateurs, en réduisant le coût d'apprentissage nécessaire pour employer ces outils, l'interface graphique rendant possible l'apprentissage par autoformation en testant les fonctionnalités du logiciel via la GUI, ce que Carroll & Mack [1984] nomment "l'apprentissage en faisant" ("learning by doing").

Les principes dominant la fin des années 70 (et compilés par exemple dans [Baker, 1976]), étaient donc de tenter d'approcher une GUI parfaitement transparente dans laquelle l'utilisateur pourrait tout voir et contrôler, lui permettant idéalement de se passer de toute information

externe ou apprentissage préalable. Dans ce contexte, l'aide n'est vue que comme un "filet de sécurité" [Kearsley, 1988, p.3] visant à pallier d'éventuelles erreurs de conception.

1.1.1.2 Nouveaux utilisateurs, nouveaux besoins

Toutefois, l'augmentation du nombre d'utilisateurs ainsi rendue possible, a entraîné l'apparition de nouvelles classes d'utilisateurs qui n'ont pas vocation à maîtriser entièrement les logiciels qu'ils utilisent [Shneiderman, 1987]. Shneiderman distingue ainsi désormais trois grandes catégories d'utilisateurs :

- **experts :** ayant des compétences en informatique et utilisant régulièrement le logiciel (ils correspondent donc, historiquement parlant, aux premiers utilisateurs des systèmes informatiques) ;
- **occasionnels :** ayant des compétences en informatique mais n'utilisant pas régulièrement le logiciel ;
- **novices :** n'ayant pas ou peu de compétences en informatique, indépendamment de la fréquence d'utilisation du logiciel.

L'utilisateur novice est rarement défini de manière précise : ainsi Goodwin [1987] se contente de dire qu'un novice est « une personne avec peu d'expérience en informatique (*i.e.* une personne qui n'est pas un expert) » sans plus de précision. Cohill & Williges [1985] tentent de quantifier en précisant qu'il s'agit d'un utilisateur ayant « moins de 10 heures d'expérience avec un système informatique interactif » – une définition qui apparait aujourd'hui extrêmement restrictive, au regard de la diffusion des systèmes informatiques. Fisher [1991] va plus loin en prenant en compte également la fréquence d'utilisation d'un ordinateur, l'expérience avec le logiciel, le type de tâche à accomplir et la formation reçue ou non au préalable. Il distingue notamment les *novices* (nouveaux utilisateurs sans connaissances préalables) des *naïfs*, ces derniers n'étant pas forcément des débutants mais des utilisateurs incapables de réappliquer, dans un contexte différent, des capacités acquises à l'origine pour réaliser certaines tâches dans un contexte bien particulier.
Dans le cadre de cette thèse, nous considèrerons toujours le terme de "novices" sous son acception la plus large possible, c'est-à-dire englobant à la fois les utilisateurs novices, occasionnels ou naïfs que nous venons de mentionner.

Ces nouveaux utilisateurs novices, qui ne représentaient qu'une proportion marginale de l'ensemble des utilisateurs il y a 30 ans, constituent aujourd'hui l'immense majorité des personnes interagissant avec des outils informatiques, remettant donc en cause les principes précédemment énoncés [Carroll & Rosson, 1987]. D'une part, ils ne sont plus forcément familiers de l'environnement du bureau et des métaphores qui en découlent [Carroll & Rosson, 1987, p.10] ; pire, en appliquant parfois trop scrupuleusement les métaphores suggérées, ils en arrivent à attendre certaines opérations non disponibles au regard de l'implémentation interne des entités [Streitz, 1988]. Au minimum, cette évolution doit nous amener à reconsidérer la pertinence

de leur utilisation, voire peut pousser certains à songer à leur abandon, en arguant que « la prochaine génération d'utilisateurs fera son investissement d'apprentissage avec les ordinateurs, et il est contre-productif de leur fournir des interfaces basées sur des imitations maladroites de technologies obsolètes » [Gentner & Nielsen, 1996, p.74]. Un autre argument montrant l'insuffisance des métaphores est apporté par des études montant que, même avec une GUI respectant scrupuleusement les principes énoncés précédemment, les utilisateurs novices n'atteignent toujours qu'un niveau de connaissance de l'application relativement médiocre [Duffy *et al.* , 1989].

Pour toutes ces raisons (voir [Capobianco & Carbonell, 2002] pour une liste plus exhaustive), il est indispensable de procurer aux utilisateurs novices une aide en ligne complémentaire, qui constitue davantage qu'un simple moyen de pallier les défauts d'ergonomie d'une application donnée.

1.1.2 Aide en ligne statique

Une étude assez étendue d'état de l'art sur l'aide en ligne statique a été réalisée par West [1995]. Elle demeure relativement exhaustive même plus de 10 ans après, car comme le soulignent Capobianco & Carbonell [2006], les recherches dans ce domaine se sont raréfiées depuis le début des années 90, plus par impuissance à améliorer l'utilisation des systèmes d'aide en ligne par les utilisateurs que parce qu'une solution réellement satisfaisante aurait été trouvée.

1.1.2.1 Attentes et objectifs

Historiquement, la première aide aux utilisateurs (même non novices) était celle fournie par le manuel papier de l'application. Dans un premier temps, les novices souhaitaient par conséquent disposer, au niveau de l'aide en ligne, d'informations similaires à celles des manuels papiers [Carroll & Aaronson, 1988], de manière à les remplacer en fournissant à la demande des extraits de ceux-ci, sans présuppositions quant à leur connaissance du système [Dzida *et al.* , 1978]. Joseph *et al.* [1989] notent qu'ils apprécient par exemple la présence d'une table des matières et d'un index pour les orienter, quand bien même ils sont relativement peu utilisés en pratique. Toutefois, comme mentionné par Jonassen [1988], il s'agit surtout d'un moyen de transposer des stratégies déjà connues lors d'une prise de contact avec un nouveau système ou logiciel, et il n'est pas évident que les novices des années 2000 soient aussi familiers des manuels papiers que ceux des années 1980. De plus, plusieurs études comme Novick & Ward [2006] montrent clairement que les utilisateurs novices ne lisent pas les manuels papiers, même lorsqu'ils existent.

Au final, selon différentes sources parfois contradictoires, les utilisateurs novices semblent attendre, de manière indépendante ou conjointe, deux rôles distincts de la part de l'aide en ligne :

- **Un rôle de formateur :** pour Kerr [1986], les utilisateurs préfèrent employer l'aide en ligne comme un tutoriel plutôt que comme une référence rapide permettant de retrouver des procédures oubliées.
- **Un rôle d'assistant :** pour Santhanam & Wiedenbeck [1993], les novices ne sont pas intéressés par un vrai apprentissage du système mais veulent y accomplir une tâche bien précise. Duffy *et al.* [1989] affirment d'ailleurs clairement que l'apprentissage est du ressort du tutoriel et non de celui de l'aide en ligne.

Dans le cadre de cette thèse, nous nous intéressons plus à l'assistance qu'à la formation, mais dans tous les cas, il apparaît essentiel que l'aide en ligne se focalise sur les tâches à accomplir dans la mesure où les novices ont une vision orientée tâche de l'application qu'ils utilisent [Carroll & Carrithers, 1984; Horton, 1994]. D'ailleurs, dans le cas de la formation, l'apport du passage au format électronique n'est pas aussi évident. Ainsi, Rathnam [2005] a montré que contrairement à la recherche d'informations spécifiques, pour l'apprentissage de l'utilisation d'un système, tâche nécessitant davantage de temps consacré à la lecture de l'aide, les utilisateurs (professionnels dans cette étude) préfèrent faire usage du manuel papier. Ce phénomène peut s'expliquer par le confort supérieur offert par le support papier par rapport aux écrans classiques (le cas du papier électronique n'étant pas envisagé ici).

1.1.2.2 Apports

Même lorsque le contenu est rigoureusement identique, le passage d'une aide papier à une aide en ligne offre des avantages en termes d'accessibilité à l'information :

- l'aide en ligne est toujours accessible en un nombre restreint et bien défini d'opérations (clics ou appuis sur une touche), contrairement au manuel papier que l'utilisateur n'a pas toujours à proximité et dont la motivation, souvent limitée chez le novice (*cf.* le "paradoxe de la motivation" de Carroll & Rosson [1987], décrit ci-dessous), n'est pas suffisante pour partir à sa recherche ;
- l'implémentation sous forme de base de données autorise une recherche par mots-clés plus rapide et plus efficace que ne l'autorise l'index d'un livre, et on observe des modes de recherche d'information différents [Palmer, 1992] ;
- l'utilisation de liens hypertextes facilite la navigation au sein de l'aide. Bien que Jonassen [1993] observait que de nombreux utilisateurs n'étaient pas tous mentalement à l'aise avec le concept d'hyperlien, les novices actuels, même en n'étant que de simples internautes occasionnels, sont probablement beaucoup moins étonnés par celui-ci que ne l'étaient ceux du début des années 90.

1.1.2.3 Limites

En dépit de progrès par rapport à l'aide papier, l'aide en ligne purement statique reste relativement insatisfaisante, aussi bien en termes d'utilisation effective que d'efficacité dans la

réalisation des tâches lors de son utilisation. Plusieurs problèmes se posent :

1. **Paradoxe de la motivation :** décrit par Carroll & Rosson [1987] et déjà mentionné ci-dessus, le simple fait de disposer d'une aide tout le temps accessible ne résout pas le fait que les utilisateurs préfèrent toujours faire appel à "un ami derrière l'épaule" [Capobianco & Carbonell, 2001].
2. **Intrusivité de l'aide :** l'aide en ligne statique, généralement affichée sous la forme d'une fenêtre secondaire ou d'un volet au sein de la fenêtre de l'application assistée, engendre chez les novices une difficulté cognitive pour alterner entre la tâche en cours de réalisation dans l'application et la méta-tâche de recherche d'information au sujet de la tâche [Kearsley, 1988; Sebillotte & Scapin, 1994].
3. **Manque de proactivité :** pour Cohill & Williges [1985], il est essentiel pour les utilisateurs novices de se sentir guidés, car si l'utilisateur doit rechercher seul et à son initiative une information au sein d'une base de données, l'aide en ligne peut devenir moins efficace que l'aide papier. Il faudrait alors envisager une "aide sur l'aide", qui ne peut malheureusement que renforcer le problème n°2.
4. **Fossé cognitif :** déjà évoqué en 1.1.1.2 dans le cas des métaphores, il s'agit plus généralement de la différence entre d'une part le modèle conçu par le programmeur et qui représente implicitement le modèle cognitif utilisé comme base pour le système d'aide, et le modèle cognitif perçu par l'utilisateur. Dès la prise en main de l'application, une divergence apparaît de part le manque de connaissances informatiques des utilisateurs novices, mais elle peut en outre se creuser suite à leur tendance à abuser du raisonnement empirique pour généraliser des résultats, entraînant la conception d'un modèle causal erroné [Mack *et al.* , 1983]. Pire, le paradoxe de la motivation fait qu'ils ne cherchent pas ou peu à réduire ce fossé en se défaisant d'un modèle cognitif trop complexe ou erroné du moment que celui-ci leur permet de réaliser les tâches voulues et d'obtenir le résultat espéré. Pour Maïs [1989], le concepteur d'applications (expert) cherche à fournir une aide optimale (principe d'optimisation) à un utilisateur occasionnel qui cherche juste à atteindre une solution satisfaisante (principe d'opérationalisation).

1.1.3 Systèmes d'aide contextuelle

Simonin & Carbonell [2007] présente un état de l'art récent sur l'adaptation, statique ou dynamique, de l'Interface Homme-Machine (IHM) en général en fonction à la fois du contexte d'utilisation et de l'utilisateur, et a par conséquent constitué l'une de nos principales sources pour cette section.

1.1.3.1 Attentes et objectifs

Pour tenter de résoudre les problèmes 3 et 4 mentionnés en 1.1.2.3, à savoir le manque de proactivité mais surtout le fossé cognitif, de nombreux travaux ont visé à permettre le

passage d'un système d'aide statique à une aide dynamique et contextuelle : il s'agit alors d'un Contextual Help System (CHS).

1.1.3.2 Apports

Lorsque l'on parle d'aide contextuelle, la notion de contexte est triple, puisqu'il s'agit de prendre en compte, de manière indépendante ou conjointe, trois types différents de contextes :

Le contexte lié à l'utilisateur : tous les utilisateurs n'ont pas les mêmes besoins et les différences inter-utilisateurs sont parfois très importantes. Au minimum, cette adaptation peut se faire de manière statique par l'utilisation de stéréotypes (ou classes d'utilisateurs), c'est-à-dire considérer que l'utilisateur dispose d'un ensemble fixe de connaissances prédéterminé sur lequel on se basera pour savoir ce pour quoi il est utile ou non de l'assister. C'est déjà ce que suggérait Shneiderman [1987] lorsqu'il proposait de fournir deux interfaces graphiques différentes selon que l'utilisateur était considéré comme "novice" ou "expert", en demandant à celui-ci de choisir la catégorie lui correspondant le mieux. Il est évidemment possible de continuer sur cette idée en augmentant le nombre de classes afin d'avoir une granularité plus fine dans l'adaptation, mais surtout en considérant que l'usager peut passer de l'une à l'autre à la manière de Trumbly *et al.* [1993], encore une fois dans le cadre des interfaces graphiques adaptatives.

Le profilage dynamique est toutefois une tâche délicate, qui requiert de modéliser et de suivre l'évolution des connaissances de l'utilisateur [Brajnik & Tasso, 1992; Paiva & Self, 1994], notamment à partir des traces d'interaction de celui-ci avec le système Fink *et al.* [1998].

Le contexte lié à la tâche : la tâche en cours d'accomplissement peut présenter certaines spécificités, et en étant capable de déterminer le but de l'utilisateur et l'étape où il se trouve dans la procédure menant à celui-ci, il devient possible de rendre le système plus proactif [Dzida *et al.* , 1987]. La définition de plans sous-optimaux est aussi une approche utile pour ne pas forcer l'utilisateur à suivre une procédure stricte qu'il n'est pas forcément en mesure de comprendre, faute de disposer du modèle cognitif adéquat [Maïs, 1989].

Le contexte lié à l'application : ou plus précisément, au modèle cognitif que l'utilisateur a de l'application dans laquelle s'accomplit la tâche. De manière plus ambitieuse, on peut tenter de déterminer le modèle mental que l'utilisateur se fait de l'application [Wærn, 1990], de manière à le comparer au modèle réel de celle-ci et donc à déceler le fossé cognitif avant qu'il ne se creuse [Leray & Sansonnet, 2007].

1.1.3.3 Limites

Si le profilage des utilisateurs peut être assez bien réalisé, la détermination de ses buts et intentions reste un problème relativement ouvert, avec des approches généralement fondées

soit sur la détection automatique de motifs particuliers dans les traces d'interaction [Horvitz *et al.* , 1998; Jameson *et al.* , 2000], soit sur l'étude d'interactions issues de sessions de type **Magicien d'Oz** où les utilisateurs sont amenés à éliciter leurs objectifs au fur et à mesure [Capobianco & Carbonell, 2001].
De plus, des évaluations comme Capobianco [2002] où des humains experts suivent des stratégies d'assistance avec ou sans prise en compte du contexte tendent à montrer que l'apport de la contextualité n'augmente que très peu le recours à l'aide.

1.1.4 Aides multimodales et dialogiques

Dans cette section, nous considérons qu'une modalité est définie à la manière de Bouchet *et al.* [2004]; Nigay & Coutaz [1995], c'est-à-dire comme un couple <d, L> constitué par un périphérique physique (physical device) d doté d'un langage d'interaction L. Il convient par ailleurs de distinguer les modalités en entrée (la façon dont l'utilisateur fait appel à l'assistance) des modalités en sortie (la façon dont le système d'assistance se manifeste auprès de l'utilisateur).

1.1.4.1 Attentes et objectifs

Si la contextualité vise à résoudre le problème du fossé cognitif, les problèmes 1 et 2 de la section 1.1.2.3 demeurent entiers. Ils sont pourtant primordiaux, car même un CHS idéal n'aurait que peu d'utilité si les utilisateurs ne veulent pas l'utiliser et/ou sont perturbés par sa présence.
Capobianco [2002], déjà mentionné dans 1.1.3.3, a également permis d'observer que lorsque les utilisateurs interagissaient avec des experts humains, quand bien même en termes de performances ceux-ci simulaient celles du système, ils faisaient appel à leur aide dans environ deux tiers des tâches, ce qui est significativement plus que ce que l'on observe avec une aide logicielle. Ils tendent également à s'exprimer de manière différente lorsqu'ils s'adressent aux experts humains que lorsqu'ils s'adressent au système [Amalberti *et al.* , 1993]. Cette observation démontre que bien plus que la compétence réelle, le paradoxe de la motivation est aussi lié à un besoin d'interaction "naturelle" : un système d'assistance traditionnel présente de nombreuses restrictions d'utilisation. Une solution intéressante consiste à faire appel à des modalités différentes, et en particulier à la langue naturelle via le dialogue.

1.1.4.2 Apports et limites de la multimodalité en sortie

Parmi l'ensemble des modalités en sortie recensées [Bernsen, 1994], sur un système informatique classique comme celui utilisé par la plupart des novices doté essentiellement de deux périphériques en sortie (écran et haut-parleurs/casque), on peut en lister au moins cinq :

Les instructions écrites <écran, langue naturelle écrite> **:** qui est celle par défaut des systèmes d'aide, aussi bien statiques que dynamiques.

Les instructions orales <haut-parleurs, langue naturelle parlée> **:** la synthèse de parole a été assez peu étudiée dans le cas de l'assistance. Dans le cas d'instructions à exécuter, il semble que les utilisateurs soient plus rapides pour effectuer les tâches demandées (éventuellement réalisables au fur et à mesure de l'écoute), mais préfèrent toutefois disposer du texte [Harrison, 1995]. Il semble en outre probable que comme dans le cas des animations, faute de temps pris pour analyser les actions effectuées, la capacité à transposer le savoir ainsi acquis soit réduite. En outre, cette modalité n'est réellement utile que si les instructions sont courtes, car dans le cas contraire, l'utilisateur ne parvient pas à les mémoriser [Carbonell & Kieffer, 2005].

L'illustration <écran, image statique> **:** combinée avec la modalité textuelle, par exemple en ajoutant des icônes au texte [Morrell & Park, 1993] ou des illustrations de l'état du système [Harrison, 1995], elle permet une amélioration des performances observée dans la réalisation de tâches.

L'animation <écran, image animée> **:** l'intérêt de graphiques animés complémentaires est plus controversée : pour Palmiter & Elkerton [1991], elles augmentent la mémorisation à long terme, mais cela se fait au détriment de la lecture du texte et conduit à une simple imitation plutôt qu'à une compréhension permettant de transposer les acquis. Harrison [1995] trouve des résultats comparables, mais montre que l'apport de l'animation est peu significatif lorsqu'il s'agit uniquement de montrer une succession d'étapes ou des mouvements du curseur. Tversky *et al.* [2002] est plus critique et pense que l'animation n'apporte rien qu'un ou plusieurs schémas comportant la même quantité d'informations ne pourraient apporter – pire, l'animation pourrait être contre-productive en étant difficile à percevoir et interprétée de manière erronée comme une succession d'éléments discrets.

La déictique <écran, mise en évidence d'une zone> **:** l'utilisation d'animations pour aider à focaliser l'attention est en revanche en règle générale clairement efficace, que celle-ci se fasse par l'utilisation de la surbrillance, d'un pointeur, de zooms [Bederson *et al.* , 1996] ou de vue en œil-de-poisson [Furnas, 1986], même si ces différentes techniques ne semblent pas avoir été étudiées spécialement dans le contexte de l'assistance.

Bien que les études sur la modalité graphique en sortie soient nombreuses, dans le cadre particulier de l'assistance, son utilisation est problématique puisqu'elle fait d'autant plus empiéter le système d'aide dans l'espace visuel bien souvent occupé par l'interface de l'application assistée. L'ajout d'un écran secondaire est un palliatif possible [Capobianco, 2002] mais cela demeure une solution coûteuse et pas forcément toujours applicable.

1.1.4.3 Apports et limites de la multimodalité en entrée

En ce qui concerne les modalités employées en entrée pour l'accès à l'aide, on trouve essentiellement :

La navigation classique <souris, pointer-et-cliquer> **:** il s'agit du mode d'interaction traditionnel avec les systèmes d'aide dotés d'hyperliens, en utilisant la souris comme périphérique d'entrée pour le pointage. Face au développement des interfaces tactiles, notamment pour les téléphones portables, on pourrait aussi envisager l'utilisation de la modalité <écran tactile, toucher> de manière relativement similaire.

La recherche écrite <clavier, mots-clés> **:** la recherche par mots-clés est l'autre mode classique de navigation au sein d'un système d'assistance. Il suppose de connaître le vocabulaire technique employé par le concepteur de l'aide et le principe de recherche par mots-clés (pas forcément intuitif pour des novices), mais aussi d'avoir une certaine familiarité avec le périphérique de saisie employé (clavier) pour que son utilisation ne perturbe pas davantage un usager déjà en situation de détresse cognitive.

Les requêtes écrites <clavier, langue naturelle écrite> **:** l'accès aux informations n'est plus perçue comme de la navigation dans une base de connaissances mais comme un dialogue [Amalberti, 1996]. La demande d'assistance se fait selon le même langage utilisé pour demander de l'aide dans des contextes différents (par exemple demander son chemin dans la rue). À défaut de faciliter l'interaction pour les utilisateurs peu efficaces dans l'utilisation du clavier, le passage d'un système de mots-clés à la langue non contrainte libère l'utilisateur d'un apprentissage préalable (se faisant souvent par tâtonnement) des limites du langage. Visuellement, l'ajout d'un champ texte en bas de la fenêtre est relativement peu intrusif.

Les requêtes orales <microphone, langue naturelle parlée> **:** l'utilisation de la langue naturelle orale et non contrainte n'a que peu d'impact sur l'efficacité de la réalisation d'une tâche : c'est d'ailleurs ce qui permet d'employer la méthodologie du "thinking aloud" pour l'étude de l'interaction homme-machine [Ummelen & Neutelings, 2000]. En outre, il s'agit d'une modalité particulièrement intuitive, et même souvent spontanée chez les novices. Cette modalité n'a aucune intrusivité visuellement, mais la reconnaissance vocale requiert souvent l'usage d'un dispositif de type micro-casque, généralement pas utilisé le reste du temps, et dont le bon fonctionnement nécessite une configuration (par lecture de texte à voix haute) qui peut prendre un certain temps.

La déictique <souris, pointer> ou <écran tactile, toucher> **:** de manière complémentaire à la langue naturelle parlée, il est possible à l'utilisateur de désambigüiser les références faites à des éléments de l'interface en les pointant (avec le doigt ou un stylet dans une interface tactile, ou avec le pointeur de la souris dans une interface plus classique).

L'intérêt essentiel de l'utilisation de la multimodalité pour l'assistance consiste à faire employer à l'utilisateur une autre modalité que celle(s) utilisée(s) pour l'interaction avec le système lorsqu'il souhaite avoir recours au système d'aide. Cette séparation en termes de

modalités permet de renforcer mentalement la distinction entre la tâche principale à réaliser et la sous-tâche “recherche d’assistance”, et donc de réduire l’intrusivité de l’aide en ligne (problème 2). À ce titre, l’utilisation de requêtes écrites ou orales semble être ce qui permet le mieux cette distinction [Capobianco & Carbonell, 2006]. Leur utilisation induit par ailleurs la possibilité de modifier le fonctionnement des modalités <écran, langue naturelle écrite> et <haut-parleurs, langue naturelle parlée> qui peuvent dès lors être utilisées pour transmettre les réponses sous forme de phrases et ainsi simuler un véritable dialogue avec le système.

1.2 Systèmes de Dialogue Homme-Machine

Bien que paraissant idéale a priori, le problème de l’assistance par le dialogue vient de la difficulté généralement constatée pour mettre en œuvre un SDHM qui soit à la fois :

- **efficace :** c’est-à-dire capable dans un premier temps d’analyser correctement et précisément les demandes des utilisateurs, afin de leur fournir dans un second temps une réponse appropriée.
- **robuste :** c’est-à-dire acceptant toutes les variabilités de la langue naturelle exprimée de manière non contrainte, par opposition aux systèmes fonctionnant avec un langage de contrôle rigoureusement défini.

Nous nous proposons donc d’analyser maintenant les différentes recherches menées en Traitement Automatique de la Langue Naturelle (TALN) dans le domaine des SDHM. Dans un premier temps, nous allons détailler les principes généraux de fonctionnement des différents modules d’un système type de dialogue homme-machine, en présentant les principes de fonctionnement théoriques généraux de ceux-ci. Puis, dans un second temps, nous présenterons quelques exemples de réalisations de systèmes de dialogues fonctionnels (avec ou sans agent virtuel animé - ce point fera l’objet de la dernière section de cet état de l’art).

1.2.1 Structure générale d’un système de dialogue

La gestion d’un dialogue est une tâche complexe mettant en jeu des techniques issues de nombreux sous-domaines du TALN. En effet, un système de dialogue est constitué de plusieurs modules, dont l’agencement communément admis (voir par exemple à cet effet Jurafsky *et al.* [2008]) est représenté sur la figure 1.1. Nous allons donc les aborder par ordre d’apparition dans la chaîne de traitement.

un module d’acquisition : il s’agit dans un premier temps de capturer la requête langagière émise oralement par l’utilisateur du système en faisant appel à un système de reconnaissance de la parole. Dans ce domaine, les approches les plus efficaces aujourd’hui sont

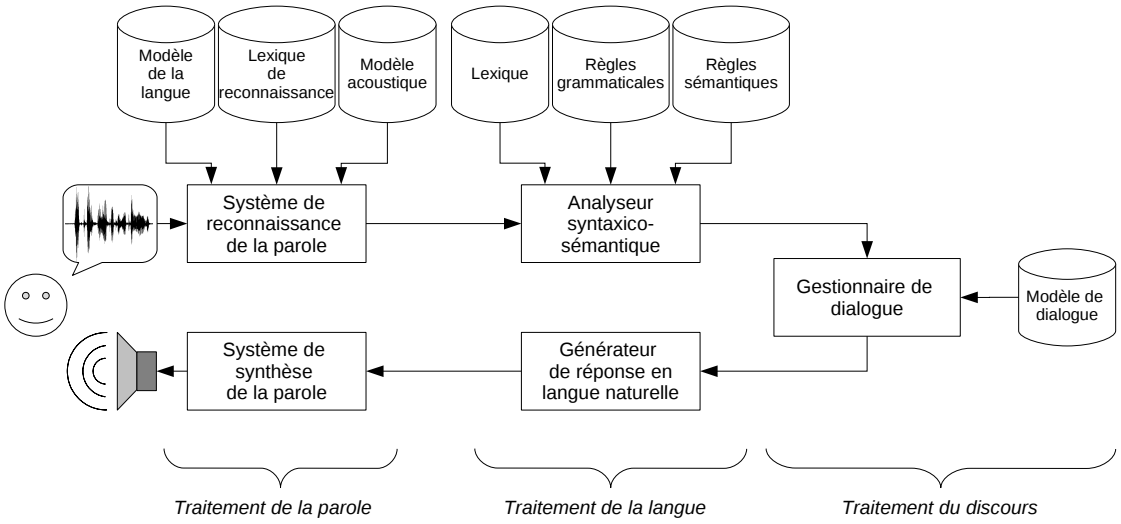


Figure 1.1 Architecture classique d'un système de dialogue homme-machine

fondées sur des méthodes probabilistes [Lamel & Gauvain, 2003], qui se fondent sur un apprentissage à partir de corpus[2] :

- un corpus de textes : le plus représentatif possible des entrées du système, visant à estimer la fréquence des différents **N-grammes** (soit ici les séquences de N mots). Il permet de disposer d'un *modèle de la langue*.
- un corpus de parole : dont la transcription manuelle permet de l'associer aux fréquences observées sur le corpus de textes, et où l'extraction d'éléments acoustiques spécifiques suite à l'analyse du signal sert d'entrée à un système d'apprentissage, classiquement à base de modèle de Markov caché (ou Hidden Markov Model (HMM)). On obtient ainsi un *modèle acoustique*.

Le principe de décodage consiste alors à confronter le signal analysé en entrée aux deux modèles ainsi construits et au lexique faisant le lien entre les deux, afin de déterminer la séquence de mots correspondante la plus probable. La difficulté principale est alors liée à l'optimisation de la recherche dans le large espace formé par les deux modèles.

un module d'analyse syntaxique et sémantique : en se basant sur diverses ressources linguistiques (lexique de mots utilisés, règles pour l'analyse grammaticale de la phrase, règles de désambiguisations pour associer le sens exact à chaque mot ou locution) qui peuvent être génériques ou adaptées aux particularités du système de dialogue, ce module produit une représentation structurée de l'énoncé fourni en entrée. Les techniques mises en œuvre ici seront détaillées dans la section 1.2.2.

un module de gestion du dialogue : en resituant l'énoncé dans le contexte des échanges précédents, éventuellement par rapport à la tâche en cours d'accomplissement (s'il s'agit d'un dialogue orienté tâche), et en utilisant un certain nombre de règles conversationnelles plus ou moins générales, le système prend une décision et produit un énoncé formel

[2] Les intérêts de l'approche corpus seront discutés en détail dans le cadre du chapitre 3, lorsque nous y aurons nous-même recours.

de réponse définissant le contenu sémantique devant être exprimé dans celui-ci (*i.e.* ce qu'on veut dire) : on parle de macroplanification. Ce point sera abordé de manière plus complète dans la section 1.2.3.

un module de génération de réponse : à partir de la représentation formelle et structurée de la réponse (généralement sous forme d'arbre), trois autres étapes, représentées sur la figure 1.2, sont communément admises comme nécessaires pour produire automatiquement un énoncé correct en langue naturelle [Bateman & Zock, 2003] :

1. la microplanification : alors que la macroplanification a déterminé préalablement les actes de dialogue (ce qu'il faut dire), il s'agit ici de choisir les éléments détaillant l'organisation interne de la phrase (comment le dire) [Levelt, 1989]. Cette étape s'intéresse donc à la gestion des références, à l'agrégation des éléments redondants en leur substituant des anaphores, et à la lexicalisation en choisissant les mots appropriés pour exprimer chaque notion.
2. la réalisation de surface : qui gère l'application des règles de grammaire en termes de choix de constructions, mais aussi d'accords des différents éléments entre eux et s'occupe par ailleurs de l'ajout de prépositions.
3. la présentation physique : finalement en charge de la coarticulation des phrases et de l'ajout de la ponctuation ; dans le cas d'un système de dialogue oral, cette étape peut aussi servir à l'ajout d'indicateurs prosodiques, la prosodie pouvant aussi être porteuse de sens, en insistant par exemple sur la prononciation de certains mots que l'on souhaite souligner [Walker & Rambow, 2002]. Cette étape n'est toutefois pas indispensable dans un système de dialogue où les réponses demeurent bien souvent relativement courtes, et n'est en outre pas considérée comme indépendante par tous les auteurs [Hovy, 2000].

Notons que cette tâche de génération de réponse peut être substantiellement simplifiée lorsque l'on travaille dans un domaine restreint permettant l'usage d'un **langage contrôlé**, comme l'ont montré Danlos *et al.* [2000].

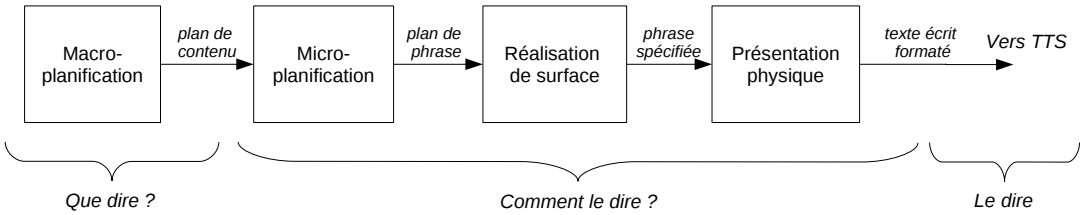


Figure 1.2 Architecture classique d'un système de génération de langue naturelle

un module de synthèse de réponse : ou Text-To-Speech (TTS), reçoit en entrée une ou plusieurs phrases, correctes d'un point de vue sémantique (si le gestionnaire de dialogue a bien fonctionné) et syntaxique (si le générateur de réponse a fait de même) qui doivent être vocalisées. Typiquement, d'après Dutoit & Stylianou [2003], ce module est lui-même constitué par :

1. un préprocesseur : pour transformer tout nombre ou symbole dans le texte équivalent et déterminer les limites des phrases. Par exemple, "Le prix est de 42 €" deviendra "Le prix est de quarante-deux euros".
2. un **analyseur morphosyntaxique** : dans la mesure où lorsqu'un mot a plusieurs **natures** possibles, celle-ci peut avoir un impact sur sa prononciation. Ainsi, le nom "couvent" (/ku.vɑ̃/) ne se prononce pas comme le verbe "couver" conjugué à la troisième personne du pluriel (/kuv/).
3. un phonétiseur : chargé d'associer à chaque morphème le phonème correspondant (par exemple /k/, /u/, /v/ ou /ɑ̃/), généralement en utilisant conjointement des lexiques de morphèmes voire de mots et des règles de transcription graphème vers phonème, en donnant la priorité à l'un ou à l'autre.
4. un générateur de prosodie : qui modifie l'ensemble de phonèmes produit précédemment de manière à adapter l'accentuation, la hauteur ou la longueur de certaines syllabes en fonction de l'utilisation conjointe de certains mots ou de la ponctuation (le ton montant par exemple à la fin d'une question).
5. un synthétiseur vocal : générant le signal audio à proprement parler, soit en utilisant un certain nombre de règles pour simuler la formation de sons dans les cavités vocales ou produites par l'articulation, soit en concaténant de manière particulière des unités acoustiques enregistrées auprès d'un locuteur humain (ce qui procure généralement un caractère moins mécanique au rendu de la voix ainsi produit).

Le premier étage du système n'est utile que si l'on s'intéresse à un dialogue oral (à la fois en entrée et en sortie), ce qui est le cas de la plupart des exemples de systèmes de SDHM. Si l'on considère une interaction langagière de type dialogue mais par écrit, cette étape devient inutile : en entrée, la requête est recueillie via un simple champ texte où l'utilisateur tape sa requête au clavier et il faut donc lui substituer un système de correction orthographique qui fournira idéalement, comme le module d'acquisition, une phrase correcte en langue naturelle. En sortie, il suffit d'afficher à l'écran la requête en langue naturelle normalement transmise au système de synthèse vocale.

Dans le cadre de cette thèse, nous nous sommes restreints à la modalité écrite en entrée et en sortie ; même si l'on ne s'interdit pas d'utiliser un outil de synthèse vocale en complément de la réponse écrite en sortie, il s'agit ici d'un simple aspect technologique et non pas d'une question scientifique. À ce titre, nous nous proposons de développer ici uniquement l'étude du fonctionnement des modules d'analyse grammaticale (*cf.* section 1.2.2) et de gestion du dialogue (*cf.* section 1.2.3). La problématique de la génération de réponse ne sera donc traitée qu'au niveau de la macroplanification, sans se préoccuper particulièrement de la génération automatique de phrases grammaticalement correctes. Cette focalisation sur ces deux modules nous servira également de grille d'analyse pour l'étude d'exemples de systèmes existants menée dans la section 1.2.4.

1.2.2 Analyse syntaxico-sémantique

L'analyse syntaxico-sémantique vise à permettre le passage d'une forme textuelle à une représentation formelle et structurée de la requête de manière à la rendre traitable par le système de dialogue. Elle se fait en trois grandes étapes successives, généralement bien distinctes :

- **l'analyse lexicale**, visant d'une part à prétraiter la phrase en ramenant ses constituants à une forme canonique, et d'autre part à identifier la nature de ses constituants (*e.g.* noms, verbes, adjectifs...).
- **l'analyse syntaxique**, qui produit une représentation des interdépendances entre les constituants de la phrase (*e.g.* syntagme nominal regroupant un déterminant, un nom et un adjectif, proposition subordonnée à une autre...) en fonction des règles grammaticales de la langue (le français dans notre cas).
- **l'analyse sémantique**, qui rattache un sens exploitable par le moteur de dialogue aux mots et structures identifiées précédemment. Ce sens est défini non pas de manière absolue, mais par rapport aux relations liant les différents concepts du domaine au sein d'une ontologie (plus ou moins spécifique au contexte dans lequel est employé le système de dialogue).

La représentation structurée ainsi obtenue se fait donc selon deux contraintes complémentaires : l'une venant de la langue (une requête doit faire sens *syntaxiquement*, c'est-à-dire être grammaticalement correcte), l'autre venant des objectifs pratiques que l'on souhaite que le système puisse atteindre (une requête doit faire sens *sémantiquement*, c'est-à-dire être pertinente en contexte).

1.2.2.1 Analyse lexicale

Lemmatisation La première étape d'une analyse syntaxique passe par une phase de lemmatisation des différents mots la constituant, c'est-à-dire le passage d'une forme fléchie à une forme canonique appelée le **lemme**[3], qui correspond typiquement à l'entrée associée dans un dictionnaire : pour les adjectifs, il s'agit de la forme au masculin singulier, pour les verbes de l'infinitif, *etc.*)
Cette phase se fait généralement en utilisant un lexique associant les formes fléchies à leurs lemmes, avec des entrées du type de celles représentées sur la figure 1.3.

Cette méthode est toutefois un peu lourde car relativement redondante : même si l'espace de stockage nécessaire pour un lexique complet d'une langue n'est plus un problème, l'ajout d'un nouveau verbe au lexique nécessite d'encoder manuellement toutes ses formes fléchies, nombreuses dans des langues comme le français. Une variante consiste donc à éviter d'encoder de manière individuelle les flexions propres à chaque mot, et à définir plutôt une liste annexe

[3]Cette définition du lemme ne fait pas consensus, et correspond plutôt pour certains auteurs comme Rastier [1994] à un lexème – nous adopterons toutefois à cette définition "classique" et nous ne discuterons pas davantage ce point qui déborde largement du cadre de cette thèse.

bleu	←	bleus, bleue, bleues
cliquer	←	clique, cliques, cliquons, cliquez, cliquent, cliquera, cliqueras, cliquerons, cliquerez, cliqueront, cliquais, cliquait, cliquions, cliquiez, cliquaient, cliquai, cliquas, cliqua, cliquâmes, cliquâtes, cliquèrent, cliquasse, cliquasses, cliquât, cliquassions, cliquassiez, cliquassent, cliquerais, cliquerait, cliquerions, cliqueriez, cliqueraient, cliqué, cliqués, cliquée, cliquées, cliquant

Figure 1.3 Extrait de lexique de formes complètes utilisé par un lemmatiseur

contenant un ensemble de règles de conjugaisons types et de terminaisons possibles. Les lemmes sont alors associés à leur règle d'accord ou de conjugaison [Pitrat, 1981], comme représenté sur la figure 1.4.

bleu	←	ADJ_REG
cliquer	←	VGRP1_REG
ADJ_REG	←	R+s, R+e, R+es
VG1_REG	←	R+e, R+es, R+ons, R+ez, R+ent, R+era, R+eras, R+erons, R+erez, R+eront, R+ais, R+ait, R+ions, R+iez, R+aient, R+ai, R+as, R+a, R+âmes, R+âtes, R+èrent, R+asse, R+asses, R+ât, R+assions, R+assiez, R+assent, R+erais, R+erait, R+erions, R+eriez, R+eraient, R+é, R+és, R+ée, R+ées, R+ant

Figure 1.4 Extrait de lexique de lemmatiseur avec règles de conjugaison types

Les lemmatiseurs gèrent donc au minimum l'**inflexion**, association d'une forme fléchie à sa racine, mais ils peuvent également prendre en compte d'autres combinaisons de morphèmes :

- les **dérivations** : qui changent la nature d'un mot. Par exemple, le suffixe dérivationnel "-ation" appliqué au verbe "modifier" donne le nom "modification". Ces informations supplémentaires peuvent se révéler utiles dans la suite de l'analyse, par exemple pour établir des règles d'équivalence du type : "faire une [R]-ation" ⇔ "[R]", qui permet de traiter exactement de la même manière deux requêtes sémantiquement identiques comme : "Je veux faire une modification de la page" et "Je veux modifier la page".
- les **compositions** : qui associent deux mots pour en former un. Ainsi, les noms "clic" et "droit" s'associent pour former le nom composé "clic-droit". Ce type de combinaison est relativement complexe à exploiter dans la suite de l'analyse dans la mesure où le sens de la composition n'est pas systématiquement en rapport direct avec celui des mots ainsi composés (par exemple on peut citer le cas classique de "pomme de terre" dont le sens n'est pas une simple composition des sens de "pomme" et "terre").
- les **élisions** et **clitisations** : l'élision, liée à l'amuïssement de la voyelle finale d'un mot lorsque celui qui le suit commence par une voyelle (*e.g.* "l'application") entraîne également une modification de la graphie du premier mot, tandis que la clitisation revient à associer un morphème, représentant une forme réduite d'un mot, à un second mot (*e.g.* "je" dans "puis-je"). Dans les deux cas, généralement, le lemmatiseur ramène le clitique à

sa forme usuelle et supprime les élisions, ce qui donnerait dans les exemples précédents les formes “la application” et “je pouvoir (+ interrogation)”, grammaticalement incorrectes mais sémantiquement identiques pour la suite de l'analyse.

Étiquetage morphosyntaxique La lemmatisation se traduit toutefois par une perte d'information, qui peut être préjudiciable dans l'analyse sémantique qui suit, comme le montre l'exemple de la figure 1.5. En paralllèle de la lemmatisation, on utiliser donc un analyseur morphosyntaxique (ou **POS Tagger**) pour associer aux locutions ou mots lemmatisés un certain nombre d'étiquettes liées à leur nature (ou **Part Of Speech (POS)**). Ceci permet de conserver les informations portées par la forme fléchie tout en les rendant plus accessibles (encodage explicite) et en permettant malgré tout au reste du système d'analyse de bénéficier de la réduction de complexité apportée par la lemmatisation.
Le principal problème provient de la difficulté à lever l'ambiguité liée à un certain nombre de mots pour lesquels à une graphie unique peuvent être associées plusieurs natures selon le contexte. Ainsi, “passe” peut être le verbe “passer” conjugué à la troisième personne du singulier au présent de l'indicatif, le même verbe à la second personne du singulier de l'impératif ou un nom masculin singulier.

Phrase 1 :	“J'ai voulu aller à la page d'accueil. . . ”
Interprétation 1 :	“. . . mais ça n'a pas marché : pourquoi ?”
Lemmatisation 1 :	Je vouloir aller à la page de accueil
Phrase 2 :	“Je veux aller à la page d'accueil. . . ”
Interprétation 2 :	“. . . dis moi comment faire !”
Lemmatisation 2 :	Je vouloir aller à la page de accueil

Figure 1.5 Exemple de deux phrases sémantiquement différentes rendues indistinctes par la lemmatisation

Pour traiter ce problème, on peut distinguer trois grands types d'approches qui se sont développées [Voutilainen, 2003] :

règles linguistiques : développée entre la fin des années 1950 (date des premiers travaux dans le domaine de l'étiquetage morphosyntaxique de manière automatique) et le milieu des années 1970, cette approche se fonde sur l'utilisation de règles d'analyse définies manuellement par des linguistes. D'abord gérées par des expressions régulières implémentées comme des machines à états finis (une analyse de phrase doit être lue correctement par tous les automates pour être acceptée, *cf.* figure 1.6), elles évoluent pour être gérées sous forme de règles fondées sur des schémas contextuels. Un des systèmes les plus performants parmi ceux couvrant de manière assez large la langue (anglaise) est TAGGIT [Greene & Rubin, 1971], qui est parvenu à étiqueter correctement 77% des mots du corpus Brown [Francis, 1964] – les 23% restants doivent être traités par des linguistes humains mais comme ils restent a priori à déterminer, les linguistes doivent encore vérifier tout le corpus.

Cette approche sera ensuite délaissée pendant près de 20 ans, jusqu'à ce que des systèmes développés à partir de la Grammaire Contrainte au milieu des années 90 permettent de nouveau de rivaliser en performance avec les systèmes utilisant une approche statistique [Samuelsson & Voutilainen, 1997].

approches statistiques : le principe repose sur l'analyse d'un corpus de langue suffisamment significatif pour pouvoir estimer la nature d'un mot en fonction de ceux qui l'entourent. Dans les années 1980, ce sont essentiellement des analyseurs faisant leur apprentissage sur un corpus annoté, en analysant le voisinage des mots sous forme de N-grammes. Sur l'exemple précédent, en plus de connaître la fréquence f_1 à laquelle "passe" est plutôt un verbe qu'un nom, on peut aussi disposer de la fréquence f_2 à laquelle le mot "passe" apparaissant dans le **bigramme** "passe moi" est un verbe.
Dans un second temps, dans les années 1990, l'analyse des N-grammes est remplacée par l'utilisation de HMM qui ont l'avantage de permettre de travailler sur des corpus non annotés et se basent sur des voisinages de tailles variables, pas forcément directement contigus au mot considéré.
Dans les deux cas, l'approche statistique offre des résultats nettement supérieurs à ce que permettait l'usage de règles. Ainsi CLAWS1 (utilisant des N-grammes) étiquette correctement plus de 96% des mots en anglais [Marshall, 1987]. En langue française, le système développé par Chanod & Tapanainen [1995] au Xerox Research Centre Europe (XRCE) à base de HMM demeure une référence.

approches hybrides : en dépit de leur bonne performance, les systèmes à base de HMM ont l'inconvénient d'être des boîtes noires puisqu'il est extrêmement difficile pour des humains d'analyser leur fonctionnement. Plusieurs recherches dont [Brill, 1995] se concentrent donc sur l'utilisation des techniques statistiques pour la génération de règles du type de celles que peuvent définir manuellement les linguistes, pour permettre éventuellement une post-édition de celles-ci.
Depuis les années 2000, ce sont d'autres types d'approches hybrides visant à combiner des règles éprouvées écrites manuellement à des systèmes d'apprentissage statistiques pour les exceptions qui retiennent le plus l'attention de la communauté scientifique dans ce domaine [Padro, 1998].

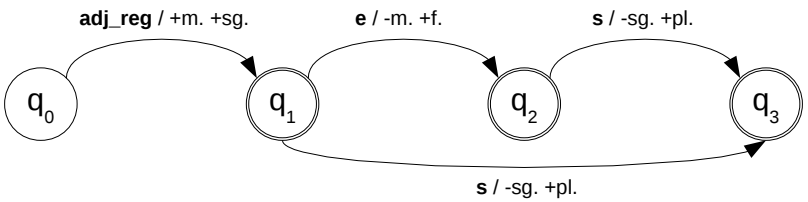


Figure 1.6 Exemple d'automate à états finis étiquetant en genre et en nombre un adjectif régulier en français

1.2.2.2 Analyse syntaxique

Afin de structurer, généralement sous forme d'un arbre, les différents composants préalablement étiquetés de la phrase, il est nécessaire de disposer d'une grammaire de la langue et d'un outil permettant de l'exploiter pour assembler les éléments selon les règles fixées par la grammaire : c'est ce que l'on nomme analyseur syntaxique ou parseur.

Grammaires Cette approche est fondée sur le principe chomskyen de grammaire générative, selon lequel il est possible de définir l'ensemble virtuellement infini des expressions bien formées d'une langue à l'aide d'un ensemble fini de règles. Dans le cadre de la langue naturelle, les grammaires les plus couramment utilisées pour l'analyse syntaxique sont les grammaires dites hors contexte, ou Context-Free Grammar (CFG), correspondant au type 2 dans la hiérarchie de Chomsky [1956]. Une telle grammaire G est définie par un quadruplet $G = (N, \Sigma, R, S)$, où :

- N constitue l'ensemble des non-terminaux, c'est-à-dire les éléments pouvant faire l'objet d'une règle de réécriture ;
- Σ constitue l'ensemble des terminaux, c'est-à-dire les éléments formant les mots de la phrase ou feuilles de l'arbre de d'analyse ;
- R est l'ensemble des règles de réécriture de la forme $A \to \alpha$ où $A \in N$ et $\alpha \in (N \cup \Sigma)^*$, et $\exists \omega \in (N \cup \Sigma)^* : S \to \omega$;
- S représente l'axiome du langage à partir duquel s'effectuent les dérivations, tel que $S \in N$.

Dans les règles de la forme $A \to s$ où $A \in N$ et $s \in \Sigma$, A correspond en fait à la nature du mot s, à la manière de ce qui a été vu en 1.2.2.1.
Si on considère la mini-grammaire $\mathcal{G}_{ex}$ donnée dans la figure 1.7, capable de représenter la phrase "je veux cliquer sur le bouton bleu" et utilisant les abréviations classiques de POS du Penn Treebank [Marcus *et al.* , 1994], l'arbre d'analyse correspondant est donné par la figure 1.8. Ici, on voit par exemple qu'une phrase n'est considérée valide dans cette grammaire que si elle est constituée d'un groupe nominal (NP) suivi d'un groupe verbal (VP). Pour que l'exemple soit complet, on donne le lexique correspondant, ce qui signifie que $\mathcal{G}_{ex}$ gère donc également l'étape d'analyse morphosyntaxique dont nous avons parlé précédemment.

À noter que l'on ne se préoccupe pas à ce niveau de l'analyse de savoir si la phrase ainsi analysée a effectivement un sens. Ainsi, la phrase "le page cliquèrent modifiant sur la bouton sûr" est parfaitement analysable par la mini-grammaire de la figure 1.7. Il est donc souvent préférable de réserver ce type d'approche aux situations dans lesquelles on sait, par hypothèse, que les phrases en entrée du système seront bien construites.

Parseur On voit que disposant d'une grammaire et d'une phrase, deux approches sont possibles pour parvenir à obtenir l'arbre de la figure 1.8 :

N = { S, DT, IN, JJ, NN, PRP, VB, PP, NP, VP }
Σ = { je, vouloir, cliquer, modifier, dans, sur, le, la, page, bouton, bleu, ouvert, sûr }
R = {

S	→	NP VP
NP	→	PRP | DT NN | DT NN JJ
PP	→	IN NP
VP	→	VB NP | VB VP | VB PP
DT	→	le | la
IN	→	sur | dans
JJ	→	bleu | ouvert | sûr
NN	→	page | bouton
PRP	→	je
VB	→	cliquer | modifier | vouloir

}

Figure 1.7 Exemple de grammaire $\mathcal{G}_{ex}$

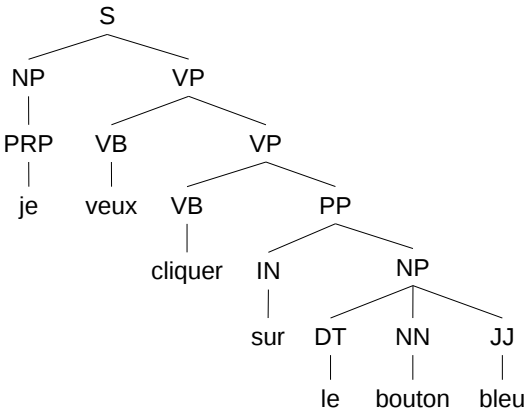


Figure 1.8 Arbre d'analyse syntaxique pour la phrase "je veux cliquer sur le bouton bleu" avec la grammaire $\mathcal{G}_{ex}$

– une analyse descendante (ou top-down) : dans laquelle on part de l'axiome S de la grammaire et on applique l'ensemble des règles disponibles jusqu'à ce qu'on parvienne à obtenir un arbre dont les feuilles correspondent à la nature des constituants de la phrase à analyser (*cf.* figure 1.9).
– une analyse montante (ou bottom-up) : dans laquelle on considère les règles de production de la grammaire dans l'ordre inverse pour tenter de remonter jusqu'à l'axiome S (*cf.* figure 1.10) de la grammaire.

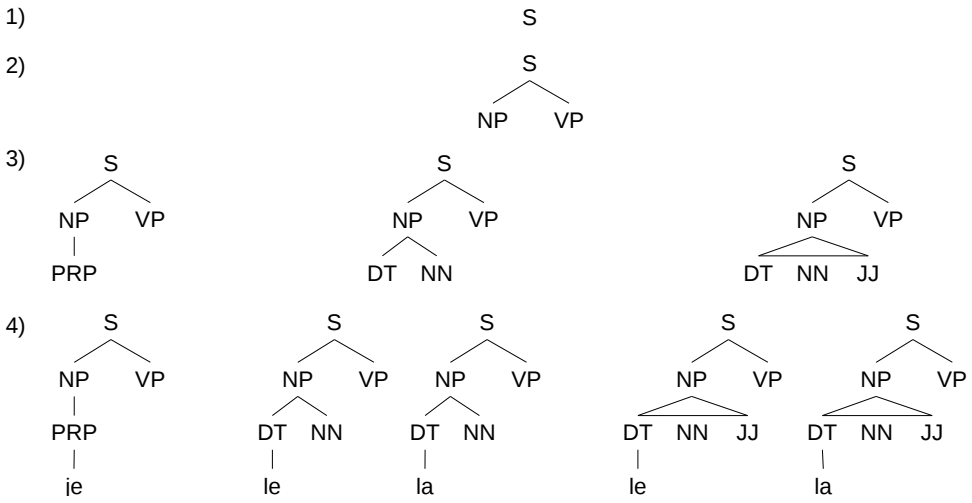


Figure 1.9 Quatre premières étapes de l'analyse descendante de la phrase "je veux cliquer sur le bouton bleu" avec la grammaire $\mathcal{G}_{ex}$: cinq arbres sont présents lors de la quatrième phase ; le plus à gauche correspond à l'analyse correcte de la phrase.

Toutefois, en supposant que l'on dispose d'une grammaire modélisant parfaitement la langue étudiée, elle ne pourrait être utilisée efficacement que sur des entrées respectant elles-mêmes exactement les règles de la langue. Si cette hypothèse est envisageable pour des articles de journaux ou des romans (encore que des problèmes ne sont parviennent pas à être correctement traités), dans le cas d'un système de dialogue homme-machine, il est très courant de disposer en entrée de requêtes fortement agrammaticales (*cf.* figure 1.11) : les approches descendantes ne sont alors pas du tout appropriées.

Une solution consiste donc à effectuer uniquement une analyse de surface (shallow parsing) sur des "morceaux de phrases" (chunks) sans recouvrement. Ainsi, dans le cas de l'exemple de la figure 1.8, une analyse de surface donnerait simplement :

$[_{NP}$ Je] $[_{VP}$ veux] $[_{VP}$ cliquer] $[_{PP}$ sur] $[_{NP}$ le bouton bleu]

En supposant que l'analyse soit totale et parfaite. Ce type d'approche permet en effet de produire un résultat partiel si la phrase n'est pas complètement analysée, par exemple en n'obtenant que certains éléments :

$[_{NP}$ Je] veux $[_{VP}$ cliquer] sur $[_{NP}$ le bouton bleu]

En outre, en disposant d'un corpus suffisant, on peut aussi envisager de combiner une approche statistique Probablistic Context-Free Grammar (PCFG) en associant des coefficients

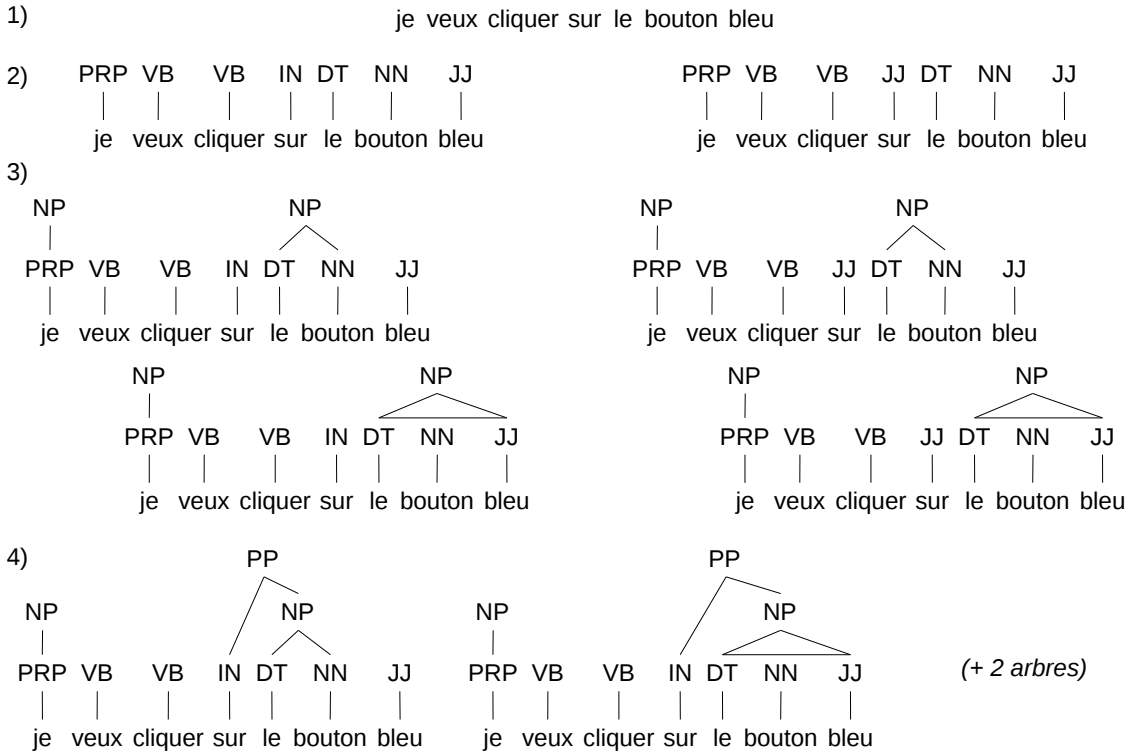


Figure 1.10 Quatre premières étapes de l'analyse montante de la phrase "je veux cliquer sur le bouton bleu" avec la grammaire $\mathcal{G}_{ex}$: cinq arbres sont présents lors de la quatrième phase ; le second en partant de la gauche correspond à l'analyse correcte de la phrase.

à la frontière y a un trait en gras pourquoi ?je doute je doute	
bin j'ai dit RAZ pas RARE	je trouve pas les autres ?
changement de bureau de Sansonnet	l'assistance elle inclu quoi exactement ?
cher usager :-) c'est bizarre ..	léa c'est ton nom ?
c qui qui lit cette barre ?	marcher
c'est le % de quoi	Page de Vitesse
heu, moyen milieu	si c une blouse, c pour faire scientifique
j'aime pas du tout que tu fais comme ça	ya des retsrictions ?

Figure 1.11 Exemples de requêtes agrammaticales recueillies dans le corpus *Daft* (*cf.* chapitre 3)

de probabilité à chaque règle.

1.2.2.3 Analyse sémantique

La dernière partie de l'analyse linguistique consiste à rattacher du sens aux différents éléments constituant la requête, ce qui se fait ici encore en deux temps, en donnant du sens :

1. aux mots individuellement : à la manière d'un dictionnaire, qui permet de définir un mot de manière relationnelle, par rapport aux autres mots présents dans ce même dictionnaire.
2. aux mots au sein de la requête : en s'intéressant aux rôles sémantiques joués par les différents constituants de la phrase.

Sens des mots À un mot[4] donné peuvent correspondre plusieurs sens. Ainsi, dans l'exemple de phrase vu dans la sous-section précédente, pour au moins deux d'entre eux ("bouton" et "bleu"), il existe plusieurs significations possibles, dont quelques-unes sont présentées dans l'encadré de la figure 1.12.

bouton	1. petite pièce généralement ronde permettant de retenir ensemble les deux parties d'un vêtement ; 2. corps rond ou allongé poussant sur les arbres ou arbustes d'où naissent les feuilles et branches ; 3. petite tumeur arrondie pouvant se former sur la peau ; 4. pièce de cuivre ou de fer, ronde ou ovale, permettant de tirer à soi une porte ou à l'ouvrir ; 5. commande manuelle d'un interrupteur ; ...
bleu	1. de la couleur du ciel dégagé en plein jour (longueur d'onde de 470 nm) ; 2. peu cuit ; 3. vivement surpris de quelque chose ; 4. complètement ivre ; ...

Figure 1.12 Quelques sens possibles pour les lemmes "bouton" et "bleu" (source : Wiktionnaire – voir aussi WordNet [Fellbaum, 1998] pour d'autres sens)

Il s'agit le plus souvent d'**homonymie** (ou plus exactement d'homographie) si les sens n'ont pas de rapport direct entre eux, comme dans les exemples de la figure 1.12. Ce peut être également de la **polysémie** si les mots sont sémantiquement liés, voire de la **métonymie** si l'usage d'un sens sert à faire référence aux autres également (par exemple "Paris" pour faire référence au gouvernement français, lieu où il siège).

[4]Nous considérons ici qu'un "mot" correspond en fait à un lemme, puisqu'on considère que l'étape de lemmatisation a été effectuée.

Définition d'une ontologie La première étape pour attacher le sens d'un mot à celui-ci est donc de disposer d'une ontologie répertoriant les différents sens possibles. À ce titre, une des références (en anglais) reste WordNet [Fellbaum, 1998], qui dans sa version actuelle (3.0) recense plus de 110 000 noms, 10 000 verbes et 20 000 adjectifs avec en moyenne 1,23 sens par nom et 2,16 sens par verbe. En langue française, il existe une ontologie similaire à WordNet, développée dans le cadre du projet EuroWordNet, qui visait à établir des ontologies similaires à WordNet dans différents langages (et de les lier au WordNet d'origine de Princeton) [Vossen, 1998].
Toutefois, dans la mesure où ces ontologies sont généralistes en visant à couvrir tous les domaines de la langue, les systèmes de dialogue utilisent le plus souvent une ontologie propre au système, qui permet d'exclure d'office un certain nombre de sens extrêmement improbables dans le contexte d'utilisation du système quand celui-ci est dédié à une tâche précise (ce qui est généralement le cas, comme on le verra dans les exemples de la section 1.2.4). Ce principe de "one sense per discourse" [Gale *et al.* , 1992] constitue donc une simplification (puisque cela revient à couper dans l'ontologie générale de la langue pour n'en conserver qu'une partie) pourrait être interprétée comme un recours à la pragmatique, c'est-à-dire à prendre en compte le contexte dans lequel est énoncé la phrase reçue en entrée.

Choix du sens approprié dans l'ontologie Une fois que l'on dispose d'une ontologie adaptée au domaine, la seconde étape consiste à attribuer le sens aux mots ou locutions de la phrase. Si cette tâche est triviale pour les mots ou locutions qui ne correspondent qu'à une unique entrée sémantique dans le dictionnaire, il n'en est pas de même lorsque plusieurs sens sont possibles (*cf.* figure 1.12) : on doit alors avoir recours à une méthode de désambigüisation de sens ou Word Sense Disambiguation. Le principe général de la Word Sense Disambiguation (WSD) consiste à considérer que lorsque plusieurs sens s_i (avec $i \in \mathbb{N} \setminus \{0, 1\}$) sont possibles pour un mot m dans une phrase, le sens correct s peut être déterminé en regardant le contexte dans lequel il occurre, c'est-à-dire les mots qui l'entourent [Ide & Véronis, 1998].
Si l'on dispose d'un corpus de taille significative de phrases étiquetées (manuellement) avec l'ontologie de sens choisie, il est possible d'extraire pour chaque sens d'un mot m :

- les mots cooccurrents les plus fréquents au sein du corpus ;
- un N-gramme formé d'une fenêtre de (N-1)/2 mots pris avant et après m.

Il suffit alors d'appliquer un algorithme d'apprentissage supervisé, par un exemple un classifieur bayesien naïf. Dans la mesure où l'annotation manuelle entière du corpus (ou d'une part suffisamment significative de celui-ci pour que les méthodes d'apprentissage soient efficaces) est parfois difficile, d'autres méthodes consistent à utiliser comme sources complémentaires la **glose** fournie par les dictionnaires [Lesk, 1986]. Ainsi, chaque sens de WordNet est associé à une glose du type de celles données en exemple dans la figure 1.12.
Il est également possible d'augmenter le nombre de phrases de référence disponibles en étendant ce procédé aux sens identifiés comme proches dans l'ontologie, par exemple les sens liés par une

relation d'hyponymie ("bleu clair" relié à "bleu"), d'hyperonymie ("couleur" relié à "bleu") ou de méronymie ("le bouton" relié à "la fenêtre de l'application"). Ces informations peuvent être directement disponibles au sein du dictionnaire utilisé (c'est le cas de WordNet), ou déterminées par une analyse au sein du corpus, en partant du principe que les mots m_j qui cooccurrent souvent avec les mêmes mots que ceux trouvés dans le voisinage de m lorsqu'il a le sens s_i ont très certainement un sens $s_{j,k}$ lié à celui de s_i.

Sens des phrases La seconde manière de donner du sens à la phrase est de considérer l'organisation des constituants entre eux, non pas en termes de rôles grammaticaux comme lors de l'analyse syntaxique, mais en termes de rôles sémantiques. En effet, d'après les travaux de Fillmore [1982] sur les Grammaires de Construction, le sens d'une phrase est plus que la somme des sens des éléments qui la constitue : les constructions grammaticales complexes sont donc également des formes ayant un sens, au même titre que les mots.
Encore une fois, il faut donc disposer de deux éléments : une ressource définissant la liste des structures sémantiques définissant les relations possibles entre les différents constituants d'une phrase, et un mécanisme permettant d'établir la liaison entre les éléments individuels d'une phrase donnée et les structures sémantiques les englobant.

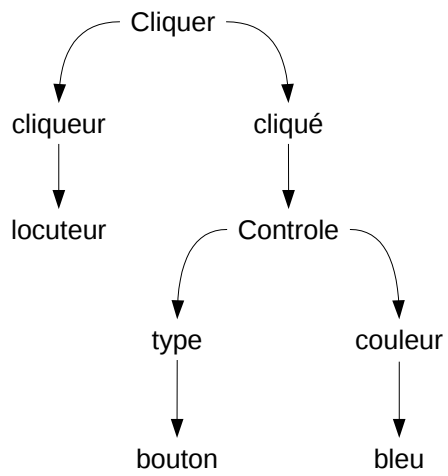
Structures sémantiques Pour être traitable par le système, la représentation du sens de la phrase doit être structurée. Elle peut se faire selon différents formalismes, et parmi les plus courants, on peut mentionner :

- les diagrammes conceptuels [Schank & Abelson, 1977] : essentiellement étudiés dans les années 1970 et 1980 ;
- la logique du premier ordre : qui est en formalisme éprouvé et permet donc de raisonner symboliquement sur les phrases ;
- les "cadres" (ou **frames**, selon le terme consacré par Minsky [1974] en intelligence artificielle) : dont le principe se base sur la représentation de stéréotypes de situations où certains éléments sont typiquement attendus pour la décrire. Ils permettent de représenter toute expression logique pouvant être exprimée dans le cadre de la logique du premier ordre.
- les réseaux sémantiques : qui sont équivalents, en termes de capacité de représentation, aux frames, mais permettent une meilleure visualisation des dépendances entre les constituants.

Un exemple de représentation de la même phrase selon ces trois derniers formalismes est donné sur la figure 1.13.

Les frames peuvent être définies a priori de manière totalement arbitraire, puisque c'est la réaction qu'elles déclenchent qui compte vraiment. Toutefois, pour formaliser leur définition, on fait généralement appel à la logique de description Donini *et al.* [1996]; Baader *et al.* [2007], qui permet de représenter la base de connaissances du système, et distingue en particulier :

$$\exists a, e\, Cliquer(a) \wedge cliqueur(a, locuteur) \wedge clique(a, e) \wedge Controle(e) \wedge type(e, bouton) \wedge couleur(e, bleu)$$




```
Cliquer                          Controle
   cliqueur : locuteur              type : bouton
   cliqué : entité                  couleur : bleu
```

Figure 1.13 La phrase "Je clique le bouton bleu" selon trois représentations sémantiques différentes : logique du premier ordre, réseaux sémantiques, frames.

- les catégories et concepts du domaine, formant une ontologie appelée la terminologie du domaine, et représentés par une *T-Box*.
- les individus et leurs attributs propres, qui sont représentés dans une *A-Box*.

Ainsi, pour donner du sens à l'exemple de phrase de la figure 1.13, il faut supposer que l'on dispose d'un certain nombre de connaissances du domaine :

- Cliquer est une relation qui lie deux éléments : un cliqueur (sujet) et un cliqué (objet)
- Un Contrôle est un élément qui a deux paramètres : un type et une couleur
- Un "locuteur" peut être un "cliqueur"
- Un Contrôle peut être "cliqué"
- Un "bouton" est un "type"
- Le "bleu" est une couleur

L'utilisation la plus répandue à ce jour d'une utilisation à large échelle de la logique de description est celle réalisée dans le cadre du Web sémantique avec le langage Web Ontology Language (OWL) [McGuinness & van Harmelen, 2003].

L'application de la logique de description pour la représentation de la sémantique a donné lieu à différentes initiatives, les deux plus notables étant la Proposition Bank (ou PropBank) [Kingsbury & Palmer, 2002] et FrameNet [Baker *et al.* , 1998].
PropBank est le plus grand corpus dont les phrases ont vu leurs constituants annotés en fonction de leurs rôles sémantiques, en s'appuyant uniquement sur la structure des verbes

employés. Ce corpus se base sur les phrases figurant dans le corpus TreeBank Marcus *et al.* [1994], qui était à l'origine annoté de manière syntaxique uniquement. Dans un tel contexte, notre phrase d'exemple pourrait y être annotée comme suit :

[Arg_0 Je] *clique* [Arg_1 le bouton bleu]

FrameNet, pour sa part, se concentre davantage sur la définition de frames décrivant des situations, et qui ne sont donc pas uniquement centrées sur le verbe. Un exemple est donné dans la figure 1.14 pour modéliser le fait de quitter un lieu.

> Un *Self_mover* quitte un emplacement d'*Origine* initial. Le *Chemin* que le *Self_mover* suit, et le *But_poursuivi* peuvent aussi être mentionnés. Bien souvent, lorsque cette frame est évoquée, cela implique que le *Self_mover* est mécontent de l'emplacement d'*Origine*.

Figure 1.14 Glose de la frame de FrameNet

Comme on le voit dans cet exemple, FrameNet s'attache également à définir des implications logiques possibles qui pourraient autoriser un système de dialogue les exploitant d'avoir des réactions de haut niveau (ici, en déduire que l'action qui amène à quitter l'application suppose un mécontentement de l'utilisateur lorsqu'il énonce cette requête).

Bien que potentiellement adaptables, PropBank et FrameNet sont en langue anglaise. En français, les trois ressources les plus proches librement disponibles sont, le LExique des Formes Fléchies du Français (LEFFF) [Clement *et al.* , 2004], les tables du LADL fondées sur les travaux de Gross [1975] et Volem [Fernandez *et al.* , 2002]. Aucune de ces ressources n'est toutefois pleinement satisfaisante dans notre contexte : le LEFFF, pose potentiellement des problèmes d'exactitude[5]. Les tables du LADL qui regroupent des verbes ayant des constructions similaires sont clairement plus orientées syntaxe que sémantique (même si les deux ne sont pas décorellés), comme le montre l'extrait donné sur la figure 1.15. Enfin, Volem a malheureusement une couverture relativement faible (seulement 1 700 verbes pour la langue prise dans son ensemble) et n'a pas de description accessible des schémas syntaxiques utilisés (*cf.* figure 1.16).

N0 =: Nhum	N0 =: Nnr	N0 =: le fait Qu P	N0 = V1 W	[extrap]		8	N0 est V-ant	N0 V	N0 est Vpp W	N1 =: Qu P	N1 =: Qu Psubj	[pc z.]	N1 =: si P ou si P	N1 = V0 W	Tc =: futur	Tc =: passé	Vc =: devoir	Vc =: pouvoir	Vc =: savoir	N1 =: ce(ci+la)	N1 =: Ppv	de N1 = de là	N1 =: Nhum	N1 =: N-hum	N1 =: le fait Qu P	Prép Nhum = Ppv	[extrap][passif]	de N1 V N0	N0 V contre Nhum
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
+	-	-	-	-	s'	abstenir	+	+	-	-	+	-	-	+	-	-	-	-	-	+	+	-	-	+	-	+	-	-	-
+	-	-	-	-		abuser	-	+	-	+	+	-	-	+	+	-	-	+	+	+	+	-	+	+	+	+	+	-	-

Figure 1.15 Extrait des tables du LADL, issu de [Gardent *et al.* , 2006]

[5] "accuracy" dans l'opposition "precision *vs* accuracy" : degré de conformité entre la valeur mesurée et sa valeur réelle – ici, la validité d'une frame produite semi-automatiquement par rapport à celle qui pourrait idéalement décrire la sémantique de la requête.

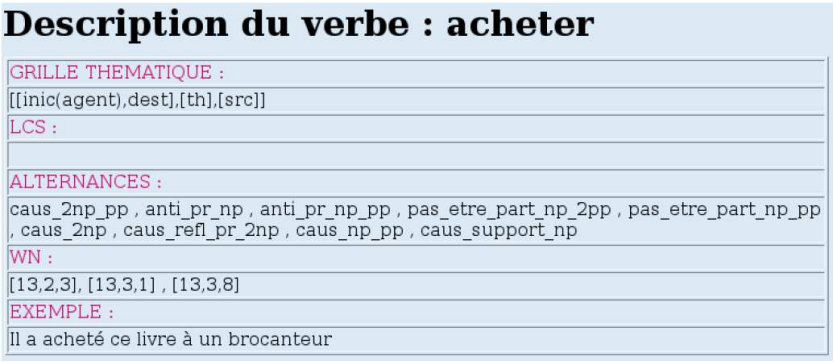


Figure 1.16 Représentation du verbe "acheter" dans Volem, issu de [Bossard, 2007]

Étiquetage de rôles sémantiques Cette dernière étape de l'analyse sémantique consiste à avoir recours à l'ontologie du domaine et/ou aux frames pour les appliquer à la phrase en cours d'analyse : c'est donc l'équivalent de ce qu'est la WSD pour les mots. Ici également, les algorithmes d'apprentissage supervisé permettent de choisir la frame la plus susceptible de correspondre à la phrase en tentant d'appliquer chacune d'elles. Pour trouver les entités correspondantes aux arguments des frames, on peut exploiter toutes les informations fournies par les précédentes étapes de l'analyse : lemme, nature grammaticale de l'entité, genre, nombre, sens, *etc.*

1.2.3 Gestion du dialogue

De la même manière que lors de l'analyse sémantique, nous avons pu voir que le sens de mots pris de manière individuelle pouvait être profondément modifié lorsqu'on les considérait en tant que groupements dotés d'un rôle sémantique au sein de la phrase, cette même phrase est à resituer dans un double contexte :

- le contexte général dans lequel a lieu l'interaction : en termes de medium d'interaction (*e.g.* le téléphone), du rôle général du système (*e.g.* fournir des informations sur des horaires de trains) ou selon des informations propres à l'utilisateur (*e.g.* pressé, habitué ou novice, *etc.*).
- le contexte du dialogue : c'est-à-dire en prenant en compte l'ensemble des interactions passées ayant déjà eu lieu entre le système et cet interlocuteur. Il est alors éventuellement possible de comparer ce dit dialogue à un modèle prototypique de dialogue, si celui-ci ne varie que peu d'un interlocuteur à l'autre (*i.e.* il est lié à une tâche avec des contraintes fortes et connues).

C'est ce contexte qui permet d'interpréter en termes de pragmatique la requête, c'est-à-dire de définir les actions à effectuer (communication avec l'application) et la réponse à fournir (communication avec l'utilisateur).
La prise en compte du contexte peut se faire avec différents niveaux de précision et de généricité. Nous allons donc distinguer par ordre de sophistication trois manières de gérer le

fonctionnement du gestionnaire de dialogue : avec un système à états finis, en modélisant les requêtes en termes d'**actes de dialogue** dotés d'une sémantique opérationnelle, et finalement en liant le système de dialogue à un véritable système de planification[6].

1.2.3.1 Système à états finis

Il s'agit du système de dialogue le plus simple, dans lequel le dialogue est modélisé sous la forme d'un arbre dont la racine est la phrase d'introduction et qui à chaque nœud se divise en autant de branches que l'utilisateur peut faire de réponses [Nicolle *et al.* , 1998]. Toutefois, il est clair que cette approche n'est appropriée que lorsque le système peut se permettre de prendre l'initiative et que le nombre de réponses possibles à chaque étape est relativement restreint, ce qui n'est pas valable le cas de la situation de l'assistance qui nous concerne.

1.2.3.2 Actes de dialogues

Théorie et taxonomies Les principes à la base de la théorie des actes de dialogue trouvent leur origine dans les travaux d'Austin [1962]. Poussant l'analogie qu'établissait Wittgenstein [1953] entre un discours et la pratique d'un jeu comme les échecs où une partie est constituée d'une suite d'échanges, Austin considère que tout énoncé de dialogue est assimilable à une action réalisé par son locuteur. Ainsi, tout énoncé peut selon lui être analysé selon trois aspects :

l'aspect locutoire : qui correspond à l'énonciation d'une phrase (comme combinaison de sons ou de mots) ;

l'aspect illocutoire : qui est l'acte réalisé par le fait même d'énoncer cette phrase (*e.g.* promesse, menace, conseil, avertissement, ordre...) ;

l'aspect perlocutoire : c'est-à-dire les effets produits par l'acte sur l'interlocuteur, généralement de manière intentionnelle (*e.g.* engagement, peur, action à réaliser...).

Les aspects illocutoires et perlocutoires sont entièrement liés au contexte d'énonciation. Ainsi, si on reprend notre exemple de la phrase "Je veux cliquer sur le bouton bleu", dans un contexte d'assistance, la force illocutoire correspondrait à l'expression d'une volonté, avec comme effet perlocutoire attendu d'obtenir une aide sur la manière de procéder.

Searle [1969] a repris les travaux d'Austin et proposé une taxonomie des actes illocutoires, qu'il rebaptise actes de dialogue, formée de cinq classes :

les assertifs : le locuteur s'engage sur la vérité de ce qu'il énonce (*e.g.* affirmer, nier, informer...) ;

les promissifs : le locuteur s'engage personnellement à faire ce qu'il énonce (*e.g.* promesse, offrir, garantir...) ;

[6] Notons qu'il est également possible d'avoir un système de dialogue homme-machine sans système de gestion du dialogue, en ne réagissant qu'à l'aide de règles de réécriture : c'est ainsi que l'on distingue les chatbots, successeurs d'ELIZA [Weizenbaum, 1966] (*cf.* chapitre 8).

les directifs : le locuteur veut que son allocutaire fasse ce qu'il énonce (*e.g.* ordonner, interdire, suggérer...) ;

les expressifs : le locuteur exprime son état d'esprit au sujet de ce qu'il énonce (*e.g.* remercier, se plaindre, s'excuser...) ;

les déclaratifs : le locuteur modifie l'état du monde par la simple énonciation (*e.g.* approuver, annuler, démissionner...).

D'un point de vue théorique, de nombreuses alternatives visant à affiner cette taxonomie ont été proposées, à l'image de Caelen [2003] qui distingue ainsi :

faire F^{A} **:** faire une action ($\Leftrightarrow$ déclaratifs) ;

faire-faire F^{F} **:** demander de faire une action ($\in$ directifs) ;

faire-savoir F^{S} **:** communiquer une information ($\Leftrightarrow$ promissifs $\cup$ expressifs) ;

faire faire-savoir F^{FS} **:** demander une information ($\in$ directifs) ;

faire pouvoir F^{P} **:** donner un choix ($\Leftrightarrow$ promissifs) ;

faire devoir F^{D} **:** obliger sans donner d'alternative ($\in$ directifs).

Mais comme on le voit ici, ces taxonomies alternatives peuvent généralement toujours être ramenées à celle proposée par Searle.

Dans le cadre plus précis de certaines applications, des ontologies d'actes de dialogue plus spécifiques sont parfois définies, ce qui peut poser problème pour les comparer (comme nous verrons dans le chapitre 7). Core & Allen [1997] ont toutefois tenté avec Dialog Act Markup in Several Layers (DAMSL) de proposer une liste étendue de 42 actes censés être indépendants d'un domaine particulier.

Sémantique opérationnelle : théorie L'intérêt de cette analyse en termes d'actes de dialogue est de pouvoir identifier le type de réponse attendue par le locuteur, d'un point de vue purement conversationnel. Pour cela, il faut définir une sémantique opérationnelle attachée aux actes de dialogue, c'est-à-dire des règles logiques à appliquer. Ainsi, Vanderveken [1990b] a travaillé à définir les réactions en définissant pour chacun des actes searliens des conditions de succès (*i.e.* les conditions du contexte d'énonciation qui doivent être remplies pour que le locuteur accomplisse un acte) et des conditions de satisfaction (*i.e.* les conditions qui doivent être remplies ensuite dans le monde pour que l'acte soit satisfait). Par exemple, un assertif du type "Je t'informe que je clique le bouton bleu" est un succès si le locuteur croyait qu'il allait cliquer sur le bouton bleu et que le système ne le savait pas avant. Il est également satisfait si après cette interaction, l'utilisateur a effectivement cliqué sur le bouton bleu et que le système le croit à cause de l'énoncé précédemment reçu. On voit donc que la prise en compte des actes dialogues d'un point de vue fonctionnel suppose que le système ait un certain nombre de croyances au sujet du monde et de l'utilisateur, et que celles-ci doivent être au moins partiellement modifiables par le simple fait de l'interaction avec un tiers : nous allons revenir sur ce point dans la section 1.2.3.4.

Sémantique opérationnelle : implémentation Berger [2006] a été amenée à définir la sémantique opérationnelle de 32 verbes (parmi la liste des verbes performatifs du français établie par Vanderveken [1990a]). Leur utilisabilité a pu être évaluée à partir d'un agent conversationnel pour le commerce électronique, disponible dans le cadre de l'application du *Deuxième Monde* [Chicoisne & Pesty, 1999, 2003].

Dans le domaine des systèmes multi-agents, la communication ne se fait pas en termes de langue naturelle mais des représentations formelles d'actes de dialogue peuvent être échangées entre au moins deux agents. Les travaux menés autours de KQML [Finin *et al.* , 1994] puis de ACL-FIPA [O'Brien & Nicol, 1998] représentent deux tentatives de définition de langages standards de communication basés sur les travaux de Searle [1969]. ACL-FIPA repose ainsi sur la définition de la sémantique opérationnelle de 22 actes de communication (`accept-proposal`, `agree`, `cancel`, `call-for-proposal`, `confirm`, `disconfirm`, `failure`, `inform`, `inform-if`, `inform-ref`, `not-understood`, `propagate`, `propose`, `proxy`, `query-if`, `query-ref`, `refuse`, `reject-proposal`, `request`, `request-when`, `request-whenever`, `subscribe`).

1.2.3.3 Prise en compte de la pragmatique du dialogue

Au-delà du fait qu'une réponse à une question au cours d'un dialogue doit être en rapport direct avec celle-ci, il existe des règles d'ordre pragmatique s'appliquant à toute interaction dialogique qui, si elles ne sont pas respectées, peuvent nuire fortement à la qualité de l'interaction et donc à l'acceptabilité du système. C'est ce que Grice [1975] résume sous le terme de "principe de coopération", et pour lequel il prodigue les quatre maximes suivantes :

1. Maxime de qualité : ne pas donner une réponse que l'on sait fausse ou pour laquelle on n'a pas d'information.
2. Maxime de quantité : fournir ni plus ni moins que la quantité d'information attendue d'après le contexte de l'échange.
3. Maxime de relation : être pertinent, *i.e.* ne pas fournir une réponse sans lien évident avec la question.
4. Maxime de manière : formuler une réponse claire, brève, ordonnée et non ambigüe.

Néanmoins, si ces maximes peuvent guider les concepteurs d'un système de dialogue dans le type de réponse à formuler à un énoncé, elles ne sont pas directement exploitables d'un point de vue opérationnel.

1.2.3.4 Modélisation des objectifs du système

Il apparaît que pour qu'un système soit réellement capable de dialogue avancé, il doit disposer non seulement d'un modèle de connaissances, mais aussi d'états internes. Dans ce domaine, qui sort du TALN à proprement parler, on peut notamment se reporter encore une fois aux travaux menés dans la communauté agents, notamment autour du modèle des

agents Beliefs Desires Intentions (BDI), proposé par Rao & Georgeff [1991]. Les réponses du système ne dépendent alors plus seulement de ses "connaissances" (beliefs) mais aussi de ses "intentions" propres qui lui fournissent des objectifs à atteindre. Ces objectifs, sélectionnés parmi sa liste de désirs, peuvent eux-mêmes évoluer dans le temps.

1.2.4 Exemples de SDHM

Après avoir vu le fonctionnement général d'un système de dialogue, nous nous proposons de lister ici quelques systèmes significatifs développés depuis les années 1990, en nous détaillant plus particulièrement ceux qui semblent les plus proches de notre problématique, notamment en ce qui concerne les modules d'analyse de requêtes langagières (requêtes écrites dans notre cas) et de gestion du dialogue. Nous ignorons ici la présence ou non d'un agent animé.

1.2.4.1 Quelques systèmes simples pour tâches fortement contraintes

Les premiers systèmes de dialogue homme-machine réalisés et exploités commercialement l'ont été pour la consultation d'horaires et la réservation de billets de train ou d'avion par téléphone, dans la mesure où il s'agit d'une interaction suivant un schéma particulièrement contraint (ordre défini des questions, variabilité faible). Leur gestionnaire de dialogue était donc du type "système à états finis" (*cf.* section 1.2.3.1). Dans le domaine de la consultation d'information comme les horaires, on peut citer les exemples de GALAXY [Goddeau *et al.* , 1994] ou PHILIPS [Aust *et al.* , 1995] pour les voyages, ou bien de MIMIC [Chu-Carroll, 2000] pour le cinéma. Des systèmes comme celui de l'université du Colorado [Pellom *et al.* , 2001] ou le système MERCURY [Seneff & Polifroni, 2000] permettent en outre d'effectuer des réservations de billet. Dans tous ces exemples, le dialogue est facilement modélisable par un automate à états finis.
Une autre technique peut consister à utiliser des structures permettant de recueillir les éléments attendus aux différents moments de la conversation [Roeck *et al.* , 2000], à la manière de ce que font les frames à l'échelle d'une phrase.

Néanmoins aucune de ces approches n'est réellement applicable lorsque le déroulement de la session dialogique n'est plus prédictible a priori, ce qui est le cas dès qu'on considère des applications plus complexes, qui ne peuvent plus être représentées facilement par un automate à états finis. En outre, même si on connait l'état de l'application et que cela restreint le champ des requêtes probables, nous ne considérons pas qu'à un état donné du système ne correspond pas forcément un ensemble de questions prédéterminables. Ce problème est essentiel dans notre cas, dans la mesure où les applications requérant un système d'assistance sont celles qui présentent un certain degré de complexité.

1.2.4.2 Systèmes de dialogues pour applications complexes

Une solution consiste donc à ne pas se contenter d'une analyse fondée sur la structure des interactions, mais à adapter la gestion du dialogue en fonction d'un modèle des buts et donc des états mentaux qui seraient ceux d'un humain placé dans le rôle du système de dialogue. Cette approche suppose donc la création d'un agent rationnel, c'est-à-dire une entité régie par le "principe de rationalité" [Newell, 1982] selon lequel il doit utiliser ses connaissances pour déterminer les actions à entreprendre de manière à atteindre ses buts. En fonction de ses connaissances du monde et de ses propres objectifs, cet agent réagit de manière différenciée à la requête d'un utilisateur reçue en entrée. C'est cette approche que nous utilisons dans cette thèse, et qui est également employée dans les systèmes listés ci-dessous.

SPA Le système Smart Personal Assistant (SPA) vise à offrir une interface dialogique pour la gestion de courriers et d'agenda électroniques sur diverses plate-formes [Wobcke *et al.* , 2006]. Le système, essentiellement réactif, suit des plans structurés suite à une demande initiale de l'utilisateur. Son architecture est donnée sur la figure 1.17, et on voit que le système de dialogue du système repose sur des agents de type BDI [Rao & Georgeff, 1991] implémentés avec le système d'agents JACK, développé par [Howden *et al.* , 2001]. L'analyse linguistique des requêtes repose sur une analyse partielle de surface cherchant à recueillir 6 types d'information différents définis en fonction des besoins du système :

- le type de la requête : question oui/non, wh- question[7], déclaration, ordre ou salutation ;
- le sujet de la requête ;
- le prédicat : en fonction d'une liste prédéfinie de verbes d'actions reconnus ;
- l'objet direct : l'élément principal rattaché au prédicat (ex : ce qu'il faut envoyer par courriel) ;
- l'objet indirect : élément secondaire en fonction du prédicat ;
- complément : de type spatial ou temporel.

TRAINS & TRIPS Dans le cadre des projets TRAINS [Allen, 1995] puis TRIPS [Ferguson & Allen, 1998], le dialogue est la clé de voûte d'un processus collaboratif au sein duquel le système et l'utilisateur sont amenés à travailler de manière conjointe pour résoudre un problème. Dans ces projets, il s'agit de gérer une situation de crise, par exemple dans un scénario visant à envoyer un véhicule de secours en un lieu donné.

TRINDI Le projet TRINDI [Larsson & Traum, 2001] s'intéresse à la définition d'un environnement de développement permettant d'expérimenter différentes implémentations d'un moteur de dialogue basé sur le concept d'état informationnel. L'état informationnel permet de conserver à la fois des informations sur l'état mental interne du système, et les différentes

[7] Questions faisant appel à des pronoms ou adverbes contenant les lettres W et H en anglais : what, when, where, which, who, whom, whose, why et how.

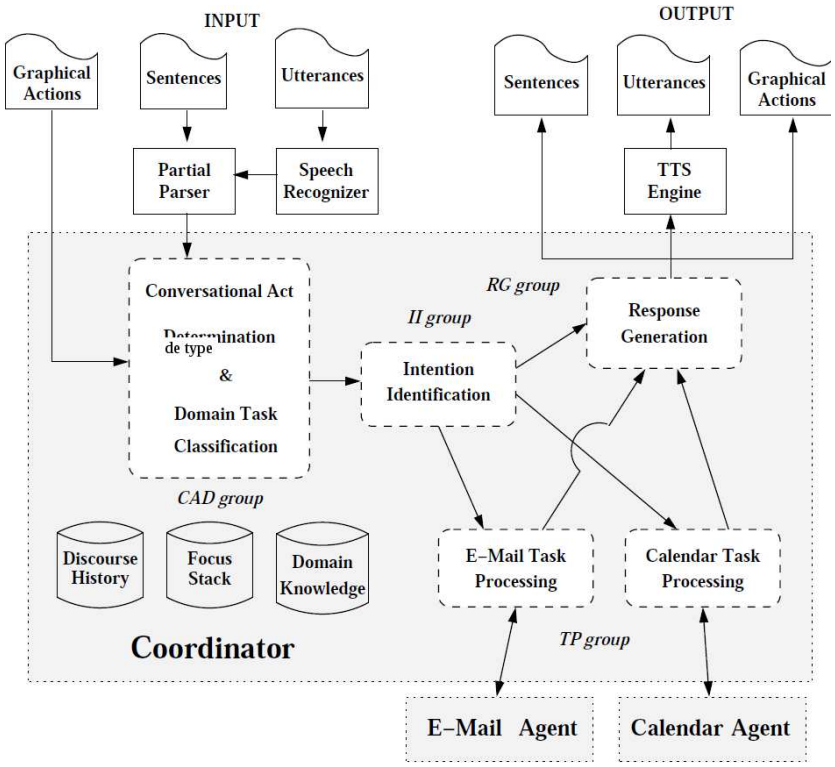


Figure 1.17 Architecture du système de dialogue de SPA, extrait de Wobcke *et al.* [2006]

informations recueillies lors des **tours de parole** précédents. Il peut aussi contenir des informations permettant d'implémenter (de manière statique ou dynamique) les règles de dialogues classiques.

Les bases de ce système de dialogue ont par la suite été reprises dans les travaux de Traum, par exemple pour le développement d'"humains virtuels" au sein d'un environnement de réalité virtuelle pour l'entraînement de soldats [Traum *et al.* , 2002].

1.2.4.3 Systèmes de dialogues multi-applications

OZONE Le système de dialogue développé dans le cadre du projet européen Offering an Open and Optimal roadmap towards consumer oriented ambient intelligence (OZONE) [Gaiffe *et al.* , 2004] s'intéresse en particulier au développement d'un système de dialogue doté d'une certaine généricité puisqu'il vise à permettre le contrôle de différentes applications (pas forcément connues a priori) dans un contexte d'interactions multimodales (*cf.* figure 1.18). Techniquement, plutôt que d'utiliser une modélisation de type BDI, l'approche choisie se concentre davantage sur une approche linguistique proche des travaux dans le domaine de l'analyse du discours, selon une version allégée de la Structured Discourse Representation Theory (SDRT) de Asher & Lascarides [2003]. La notion essentielle est celle de *domaines de référence* [Landragin, 2004] : par exemple, "le triangle rouge" suppose d'avoir un domaine qui contient des triangles et des éléments rouges, et prise en compte d'informations potentiellement issues d'un contexte multimodal.

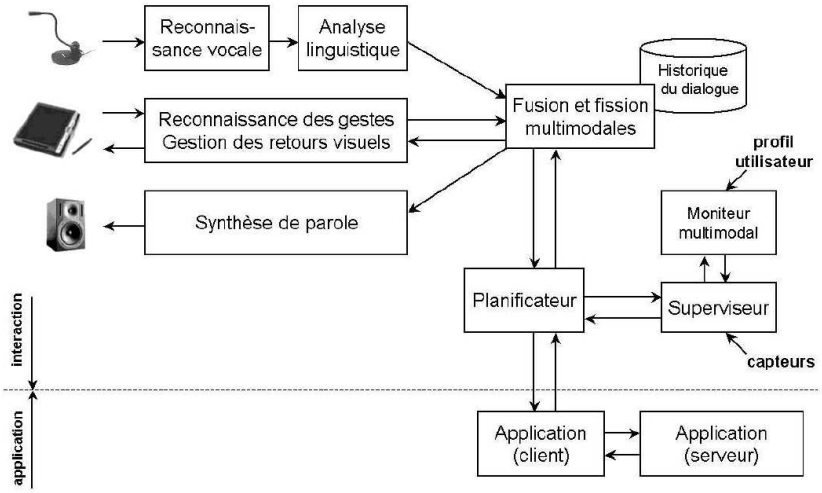


Figure 1.18 Architecture du démonstrateur OZONE, extrait de Gaiffe *et al.* [2004]

ICAI Le système proposé par Paraiso & Barthès [2004] est assez proche de notre problématique, puisqu'il s'intéresse également à une interface conversationnelle employée pour fournir de l'assistance. Une Interface Conversationnelle pour une Aide Intelligente (ICAI) est définie comme étant « le résultat de la conjonction d'un mécanisme conversationnel en langage naturel parlé et de la gestion intelligente de ce mécanisme, permettant le déroulement d'un dialogue coopératif et capable de gérer le déclenchement de plusieurs tâches à la demande de l'utilisateur, avec un minimum d'effort de la part de ce dernier ». Les auteurs, qui s'intéressent notamment à la définition d'une politique de présentation et d'affichage des informations lorsque plusieurs sources sont disponibles, et emploient une alternative à l'approche BDI classique de manière à pouvoir intégrer des mécanismes d'apprentissage et d'adaptation au comportement de l'utilisateur.

1.2.5 Utilisabilité du dialogue pour l'assistance

Contrairement aux utilisateurs experts qui s'intéressent à la manière dont fonctionne le système, les questions les plus fréquentes chez les novices semblent être liées à :

1. la prise de contact : "comment commencer ?" [Horton, 1994]
2. les prérequis d'une action : "que dois-je faire afin de... ?" [Sebrechts & Swartz, 1991; Jackson *et al.* , 1992]
3. les possibilités offertes : "où suis-je ?" et "que puis-je faire à partir de là ?" [Trenner, 1989]

Si elles font ressortir le besoin d'une modélisation précise de l'application assistée et des actions qui peuvent y être accomplies, ce type de questions, relativement indépendantes les unes des autres, montre également que les besoins en termes de modélisation de la session dialogique ne sont pas très forts. Nous confirmerons cette impression lors de l'étude du corpus *Daft* aux chapitres 3 et 7.

1.3 Agents Conversationnels Animés (ECA)

1.3.1 Les besoins psychologiques des novices

Nous avons vu dans la section 1.1.4 que l'usage de la multimodalité était un moyen efficace de diminuer l'intrusivité des systèmes d'aide habituels tout en luttant contre le paradoxe de la motivation en offrant un mode d'interaction (le dialogue) plus proche de celui employé lorsque l'utilisateur fait appel à "un ami derrière l'épaule" [Capobianco & Carbonell, 2001]. Cette expression n'est pas anodine, et révèle un autre aspect essentiel des attentes du novice en quête d'assistance : un véritable besoin de soutien psychologique. En effet, les utilisateurs novices sont bien souvent anxieux lorsqu'ils doivent accomplir une tâche [Blignaut *et al.* , 2000], ce qui les amène à douter du système informatique qu'ils utilisent de manière générale [Negron, 1995]. Dans ces conditions, leur demander d'utiliser un autre outil informatique pour comprendre le premier est donc assez paradoxal, et n'a finalement souvent comme conséquence que d'augmenter le stress déjà présent.
En outre, Shneiderman [1980] a montré que cet état les rend moins à même de comprendre correctement et ralentit donc leur apprentissage de l'utilisation du système. Cette situation n'est toutefois pas irrémédiable : ainsi, il peut suffire d'un certain nombre d'expériences "positives" avec le système pour contre-balancer cette tendance naturelle [Cambre & Cook, 1987].

1.3.2 Apports des Agents Conversationnels Animés à l'apprentissage

1.3.2.1 De nombreux avantages

Or le développement d'interfaces homme-machine intégrant un Agent Conversationnel Animé a montré qu'ils constituaient une alternative intéressante, notamment pour des utilisateurs peu habitués aux technologies informatiques (jeunes enfants, personnes âgées, personnes souffrant d'un handicap...). C'est ainsi que Lester *et al.* [1997] a montré que la simple présence d'un agent animé au sein d'un environnement d'apprentissage permettait de rendre cet environnement plus crédible et donc d'améliorer l'expérience qu'en retire les sujets : cet effet positif a été baptisé le "Persona Effect".

Depuis lors, cet effet a été confirmé dans diverses études. Moreno & Mayer [2000] a montré que les étudiants recevant des informations dans le cadre d'un dialogue avec un agent conversationnel étaient non seulement plus motivés par l'apprentissage, mais également plus performants dans la mémorisation des informations transmises, par rapport à d'autres étudiants à qui ces mêmes informations étaient fournies par le biais un simple texte. Les auteurs ont par ailleurs déterminé que c'était l'utilisation d'une modalité supplémentaire (la synthèse vocale réalisée par l'agent) et l'utilisation du dialogue plutôt que d'un monologue qui justifient cette meilleure performance, plus que la présence d'un agent animé à proprement parler.
Toutefois un agent animé peut aider à améliorer les performances à partir du moment où il

emploie le regard et des gestes déictiques pour désigner les zones de l'écran liées à l'explication donnée. Ainsi, Atkinson [2002] a montré que même dans le cadre d'un monologue, les sujets apprenant avec un agent animé retenaient davantage les informations transmises que ceux confrontés au même monologue donné sous forme d'un simple texte, même si ce texte était en plus lu par un système de synthèse vocale mais sans agent animé présent. L'ajout d'un personnage virtuel permet d'ajouter de la multimodalité en sortie et donc de renforcer la naturalité de l'interaction déjà permise par l'utilisation du dialogue.

1.3.2.2 Des avantages parfois discutés

Dans d'autres études en revanche, le rôle positif des ECA est apparu comme plus modeste. Par exemple, il semble que lorsqu'ils agissent comme de simples guides de navigation, leur apport ne soit pas significatif en termes d'apprentissage, comme ont pu le montrer Graesser *et al.* [2003] dans le cas d'un agent visant à indiquer les possibilités de navigation dans un site Web. Ce résultat est toutefois à considérer avec précaution, dans la mesure où aucune différence n'a été observée entre les quatre modes testés (avec tête parlante animée et dotée de possibilité de pointage, avec une simple voix, avec du texte écrit, sans aucun guide) : ce n'est donc pas forcément tant l'agent que l'aide même à la navigation qui serait en cause dans l'exemple testé.
D'autre part, Moreno *et al.* [2002] ont montré qu'il n'existait pas forcément de corrélation entre l'efficacité réelle d'un apprentissage (en termes de mémorisation ou de capacité à accomplir la tâche apprise par la suite) et le plaisir pris par l'utilisateur lors de l'emploi du système d'aide à l'apprentissage. Mais même si un agent ne permet que de rendre l'apprentissage plus agréable et n'améliore pas directement la performance, l'utilisateur sera plus enclin à interagir de nouveau avec un système qu'il a évalué positivement à l'avenir : indirectement, l'apprentissage serait donc tout de même meilleur sur le long terme [Beun *et al.* , 2003].
Krämer *et al.* [2003] fait également part de résultats contrastés lorsque l'on compare l'utilisation d'un agent virtuel animé avec un système d'affichage du texte doublé d'une synthèse vocale. Dans cette expérience, visant à apprendre le fonctionnement d'un système d'enregistrement vidéo, l'agent n'a pas été évalué plus positivement que le système {texte + voix}, sauf dans les situations où l'agent reportait une information négative, auquel cas il permettait d'atténuer la frustration de l'utilisateur. Ce genre d'expérience négative étant assez caractéristique de la situation d'utilisateurs faisant appel à un outil d'assistance, on peut donc estimer que cela renforce l'intérêt que les ECA peuvent avoir dans le cadre de l'assistance.

1.3.2.3 Un contre-exemple notoire

Enfin, dans le domaine des ECA utilisés pour l'assistance à la réalisation d'une tâche, il est impossible de ne pas citer le contre-exemple de l'agent Clippy de Microsoft (*cf.* figure 1.19) [Horvitz *et al.* , 2001]. Implémenté au sein de la suite *Microsoft Office* à partir

d'Office 97, il a définitivement disparu dans Office 2007, après avoir fait face à de nombreuses critiques de la part des utilisateurs qui le considéraient davantage comme une source de frustration supplémentaire dans leur utilisation de l'aide. Ce rejet est toutefois à considérer avec précaution, dans la mesure où d'autres facteurs entrent en jeu :

- l'intrusivité : l'agent apparaissant sans être sollicité pour pointer l'existence de procédures assistées pour la réalisation de la tâche en cours d'accomplissement (*cf.* la bulle de la figure 1.19) ;
- le manque de prise en compte du niveau de l'utilisateur : les mêmes conseils sont donnés à la personne lançant Office la première fois qu'à un utilisateur chevronné ;
- le manque de multimodalité : Clippy apparaît en surimpression du document en cours d'édition, mais n'est pas en mesure d'interagir avec celui-ci, par exemple en pointant un élément du document en rapport avec la réponse apportée ;
- le manque d'efficacité : le système d'aide auquel est lié Clippy est relativement basique (recherche par mots-clés) et peu performant ;
- le manque de dialogisme : même lorsqu'il comprend correctement une question, Clippy se contente d'ouvrir une fenêtre contenant la procédure à appliquer, et n'exploite donc pas la possibilité d'interagir davantage avec l'utilisateur en langue naturelle.

Clippy n'employant aucun des éléments mentionnés dans ce qui peut faire le succès d'un agent conversationnel animé, son rejet ne remet donc pas en cause les observations précédentes. Néanmoins, il illustre le fait que la présence d'un ECA sous-entend implicitement pour l'utilisateur que le système est dotée de capacités plus proches de celle d'un humain qu'un système classique : par conséquent, la déception est donc encore plus forte lorsque le système échoue sur des tâches simples [Xiao *et al.* , 2004].

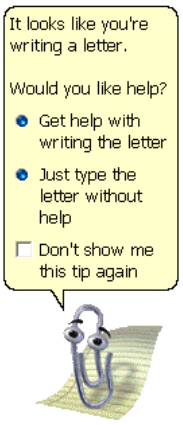


Figure 1.19 L'agent animé de Microsoft Office : Clippy

Globalement, il semble que dans le contexte de l'assistance à des utilisateurs novices, l'utilisation d'un ECA ait toutes les chances de se révéler positive, au moins en termes d'incitation à utiliser le système d'aide. Les ECA peuvent donc être un moyen efficace de lutte contre le paradoxe de la motivation.

1.4 Résumé

À l'issue de cette analyse de l'état de l'art, il apparaît que l'assistance à des utilisateurs novices est d'une part nécessaire, et d'autre part demeure un champ de recherche ouvert car les diverses solutions proposées depuis près d'une quarantaine d'années ne sont pas satisfaisantes. Dans ce contexte, différentes études ont montré que l'assistance par le biais du dialogue (oral ou écrit) semble être particulièrement prometteuse car employant le mode d'expression naturel des utilisateurs en situation de détresse cognitive. Par ailleurs, le recours au dialogue dans le contexte particulier de l'assistance semble envisageable de par les restrictions linguistiques que cela entraîne. Enfin, on a vu que notamment grâce au "Persona Effect", l'utilisation d'une interface dotée d'un Agent Conversationnel Animé permet d'améliorer l'acceptabilité du système, pour peu que le système soit suffisamment performant. L'utilisation d'un ECA pour l'assistance permet donc d'espérer augmenter l'utilisation effective du système d'assistance par des novices.

Chapitre 2

Architecture d'un Agent Conversationnel Assistant

Sommaire

Le projet DAFT au sein duquel ont été réalisé les travaux présentés dans cette thèse vise à développer des agents conversationnels animés assistants pour venir en aide à des utilisateurs novices. Il est le fruit de plusieurs initiatives successives menées au LIMSI, au sein du groupe Architectures et Modèles pour l'Interaction (AMI), depuis 1999. Plusieurs modules, indépendants ou complémentaires, ont été réalisés pour répondre à cet objectif général, et afin de comprendre les choix architecturaux effectués, il est utile de retracer l'évolution du projet avant d'analyser ceux-ci plus en détail.

2.1 Aux origines : le projet Interviews (1999–2003)

2.1.1 Description générale

Le projet Interviews, initié en 1999 par Jean-Paul Sansonnet [Sansonnet, 1999], s'intéresse à la notion d'agents dialogiques, c'est-à-dire des composants logiciels dans lesquels l'agent

conversationnel "se confond" avec l'application elle-même. La problématique consiste alors à doter ces agents de capacités d'introspection, de façon à ce qu'ils soient capables de représenter leur propre comportement interne et de raisonner sur celui-ci. L'idée-clé dans cette approche est donc de ne pas définir de modèle séparé de l'agent mais au contraire de raisonner sur la représentation symbolique définissant son fonctionnement, en se basant sur la notion de vues définies dans un langage spécifique, le View Design Language (VDL). Les demandes d'introspection à l'agent sont générées par des requêtes des utilisateurs émises en langue naturelle : pour chaque agent de ce type, un mini-corpus de quelques dizaines de requêtes a été défini et leur formalisation se fait donc de manière relativement ad hoc.
Un exemple d'agent Interviews est représenté sur la figure 2.1.

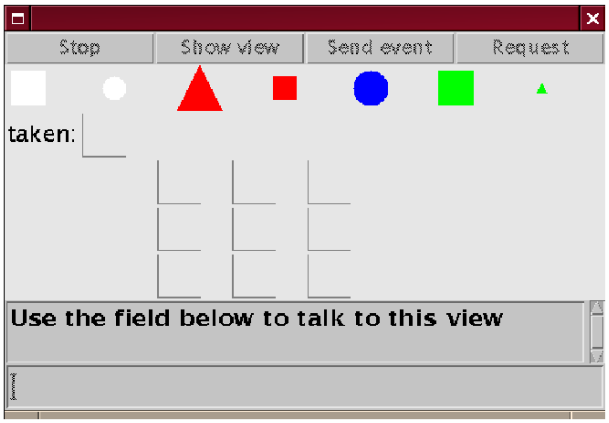


Figure 2.1 Exemple d'agent Interviews : Jojo, issu de Sabouret [2002], qui est capable de se représenter son fonctionnement, raisonner dessus et répondre à des questions sur celui-ci

2.1.2 Critique

Si l'idée d'assistance à des novices était déjà présente, il s'agissait davantage d'une perspective d'utilisation pour les agents Interviews afin de pouvoir les déployer sur le Web (développés en Java, ils peuvent en effet être mis en ligne sous forme d'applet). S'il y avait d'ores et déjà une volonté de travailler à partir de requêtes en langue naturelle, la transformation d'une requête langagière en requête formelle n'était pas au cœur du projet, avec deux conséquences principales :

un manque de représentativité des requêtes : les requêtes étaient recueillies dans des conditions pas forcément réalistes (par exemple entrées par les développeurs eux-mêmes) et elles ne pouvaient dès lors pas être considérées comme réellement représentatives des besoins de vrais utilisateurs novices.

un manque de généricité du langage : on redéfinissait pour chaque application un langage ad hoc à partir des requêtes langagières, sans exploiter les similitudes entre les requêtes recueillies dans le cadre de différentes applications.

Enfin, comme les mini-applications étudiées étaient elles-mêmes considérées comme étant des agents dialogiques, il n'y avait par définition pas de distinction faite entre l'agent assistant

et l'application assistée, et par conséquent la question ne se posait pas de disposer d'une personnification de l'agent assistant (par exemple sous la forme d'un ECA).

2.2 DAFT (2004–2009)

2.2.1 Description générale

Le projet Dialogical Agent Formal Talk (DAFT) a été lancé en 2004, en prolongement d'Interviews. Dans ce projet, la *Fonction d'Assistance* devient le sujet d'étude principal. S'il est toujours question de poursuivre les travaux initiés par Sabouret [2002] dans Interviews en développant des outils et méthodes de génération et d'*analyse d'un modèle dynamique*, le projet DAFT se distingue par une volonté de généricité, tant au niveau des méthodes utilisées pour l'introspection du fonctionnement de l'application que pour la formalisation des requêtes langagières. Pour cela, le projet DAFT requiert d'étudier une large palette d'applications différentes, utilisées dans des conditions réalistes par des utilisateurs novices. Idéalement, l'objectif de ces travaux est de pouvoir fournir à terme aux concepteurs d'applications un environnement leur permettant de déployer, pour un investissement minimal en temps, un agent assistant capable de s'interconnecter avec leurs propres applications.

2.2.2 Objectifs

Cette approche centrée sur les notions d'assistance et de généricité permet ainsi de formuler trois nouveaux besoins propres au projet DAFT :

séparer l'application assistée de l'agent assistant : dans Interviews, l'application était elle-même l'agent dialogique. Au contraire, ici, pour obtenir un système générique, il faut que l'environnement d'assistance soit capable de s'adapter pour raisonner à partir de différents modèles d'applications, qui peuvent être synthétisés de deux manières :

- fournis par les concepteurs d'application dans un format prédéfini (par exemple en XML) de manière à être exploitables par l'agent ;
- extraits de manière automatique à partir d'une introspection du code source par l'agent lui-même.

traiter la diversité des requêtes utilisateurs : de manière à rendre le système le plus facilement adaptable à de nouvelles applications, il n'est plus possible de se contenter de traiter quelques exemples de phrases prototypiques. Par conséquent, le système doit être conçu autour d'une chaîne de traitement de la langue naturelle et elle-même doit être fondée sur un corpus représentatif de requêtes.

intégrer un agent virtuel : pour les raisons discutées dans le chapitre 1, en particulier le "Persona Effect" [Lester *et al.* , 1997], il est utile de disposer d'un ECA qui vise à :

- stimuler par sa présence visuelle l'utilisateur à faire appel au système d'assistance en cas de besoin ;

– agir comme un médiateur entre la chaîne de traitement de la langue naturelle et l'utilisateur ;
– exprimer la réponse du système de manière multimodale (langue naturelle, expression d'émotions, déictique. . .).

2.2.3 DAFT 1.0 (2004–2006)

Une première phase du projet entre 2004 et 2006 a permis de concevoir une version fonctionnelle du système DAFT, que nous appelerons `DAFT 1.0`, et dont nous allons analyser ici successivement l'architecture, les apports pratiques et les limitations.

2.2.3.1 Architecture

Son architecture est fondée sur les idées énoncées précédemment en 2.2.2, et est représentée sur la figure 2.2. Elle se compose de quatre modules principaux dont nous allons ici décrire le fonctionnement général.

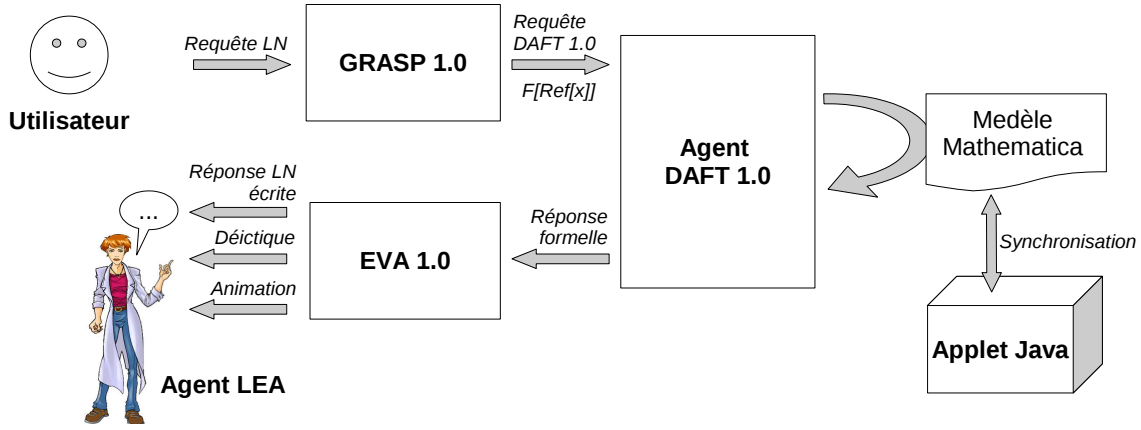


Figure 2.2 Architecture du système `DAFT 1.0`

L'analyseur sémantique GRASP 1.0. Le Générateur de Requêtes par Analyse Sémantique Partielle (GRASP) est implémenté en *Mathematica*[1]. Ce module est constitué d'un ensemble d'outils de traitement de la langue naturelle réalisant successivement trois opérations principales :

1. la lemmatisation des mots de la requête langagière ;
2. l'identification des références, en exploitant les travaux de [Gérard, 2003] sur la référence extensionnelle associative ;
3. l'identification d'un **acte de langage** principal, déterminé parmi une liste de 23 classes sémantiques (*cf.* table 2.1), à la manière de Core & Allen [1997].

[1] De Wolfram Research, en version 5.0

Il fournit alors en sortie une requête dans un langage nommé *DAFT*. Cette requête est semi-formelle, usant de représentations formelles pour identifier les actes de langage, mais conservant la représentation langagière pour les références : objets (*e.g.* “le bouton”), propriétés (*e.g.* “la vitesse”), événements (*e.g.* “le clic”), *etc.*

Catégories	Actes
Désirs	WANT, NEED, CAN
Commandes	EXEC, UNDO
Questions	ASK, EXIST, CHECK, WHYEXIST, WHYNOTEXIST, WHYIF, WHYINSTD, HOW
Argumentation	REPLY, ACK, NACK, AGREE, DISAGREE
Opinion	TELL, LIKE, DISLIKE
Ouverture de session	HELLO

Tableau 2.1 Classification des actes de langage utilisés par `GRASP 1.0` pour former une requête *DAFT*

L’agent rationnel DAFT 1.0. Le module de raisonnement du système implémente la sémantique opérationnelle des actes de langage que l’on retrouve dans la requête formelle en *DAFT* transmise par `GRASP`. Le processus de choix de la réaction en fonction de la requête *DAFT 1.0* reçue est réalisé par une imbrication de structures de contrôle d’alternatives, comparant de manière itérative la requête reçue à l’ensemble des structures de requêtes connues jusqu’à trouver une correspondance pour déclencher la réaction correspondante. Réactions et requêtes ont donc ici une relation de type “un à plusieurs”, chaque requête ne déclenchant qu’une et une seule réaction (*cf.* figure 2.3).
Cette réaction prend la forme d’un ensemble de règles à appliquer, définissant ce que doit faire l’agent de manière à construire une réponse en langue naturelle à la requête de l’utilisateur. Elle implique généralement une consultation du modèle de l’application, et donc au préalable l’établissement de la correspondance entre les références langagières et les entités contenues dans le modèle de l’application. Une fois cette réponse construite, elle est transmise au module EVA qui se charge de sa mise en forme et de son affichage.
Notons que la réponse fournie par le module DAFT n’est pas forcément textuelle uniquement : des informations supplémentaires sous forme de balises à destination de EVA (non visibles dans la réponse affichée à l’écran) peuvent en effet être ajoutées pour déclencher certaines expressions ou mouvements de la part de l’agent animé. Cet agent, appelé LIMSI Embodied Agent (LEA), est décrit ci-dessous.

L’application et son modèle. Le modèle de l’application est construit manuellement dans une représentation utilisant des expressions symboliques *Mathematica*, puis lié aux différentes fonctions de l’application de manière à ce qu’une action sur la GUI se traduise par un changement au niveau du modèle et que réciproquement, une action effectuée sur le modèle par l’agent (suite à une requête utilisateur) modifie l’état réel de l’application. Le maintien du lien

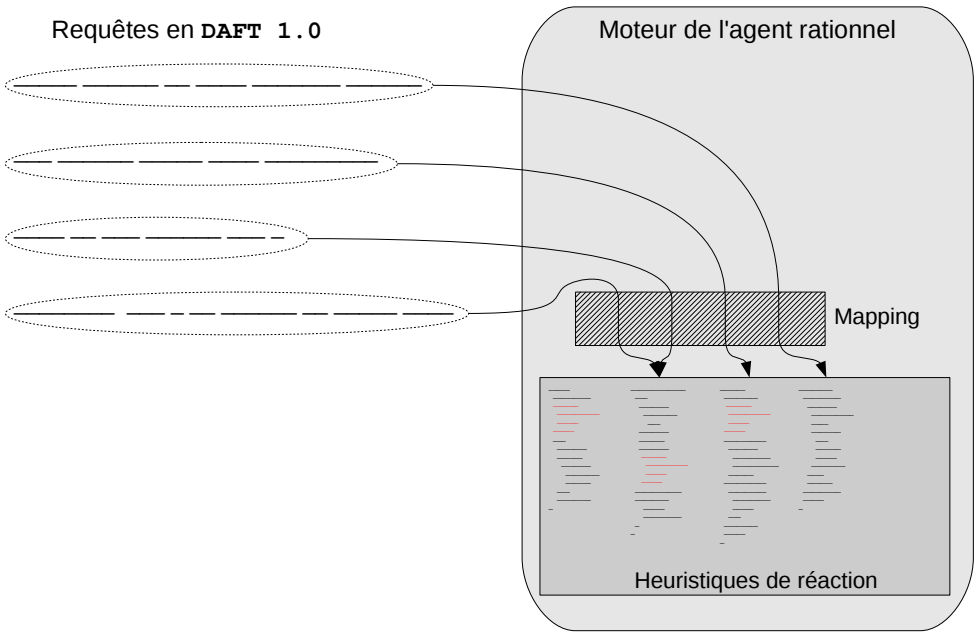


Figure 2.3 Agent rationnel DAFT 1.0 assurant le lien entre une requête formelle et une heuristique de réaction

modèle-application pose donc une contrainte forte sur le langage utilisé pour coder l'application (Java dans le cas présent).

Le contrôleur d'animation EVA. L'agent animé est contrôlé par le module EVA en charge de la production d'une réponse multimodale et de la synchronisation des éléments qui peuvent la constituer, à savoir :

- l'expression de la réponse en langue naturelle (via un phylactère),
- l'expression émotionelle de l'agent (animation des éléments du visage et du corps),
- la déictique de l'agent animé (en le déplaçant à l'écran et en utilisant les mouvements des bras pour pointer vers un élément).

L'agent animé LEA. Les agents LEA sont des personnages de style "2D cartoon"[2] (4 personnages sont disponibles), issus de travaux menés au sein du projet européen Natural Interactive Communication for Edutainment (NICE) [Buisine & Martin, 2005], entre 2002 et 2005. Ils apparaissent sur le côté de l'application qu'ils assistent (*cf.* figure 2.4), et sont constitués par un assemblage d'images au format GIF (avec transparence) représentant les différentes parties du personnage (corps, bras droit et gauche, tête, bouche, yeux et regard) qui permettent en les combinant de créer différentes poses s'enchaînant pour former des séquences expressives.

[2]C'est-à-dire issus de la composition de dessins réalisés par un infographiste. Ils sont volontairement "non réalistes", à la fois dans leurs proportions et dans les moyens d'expression émotionnelle utilisés (simples emotes).

Pour plus de détails techniques sur l'implémentation de ces différents modules, on peut se reporter à Le Guern [2004].

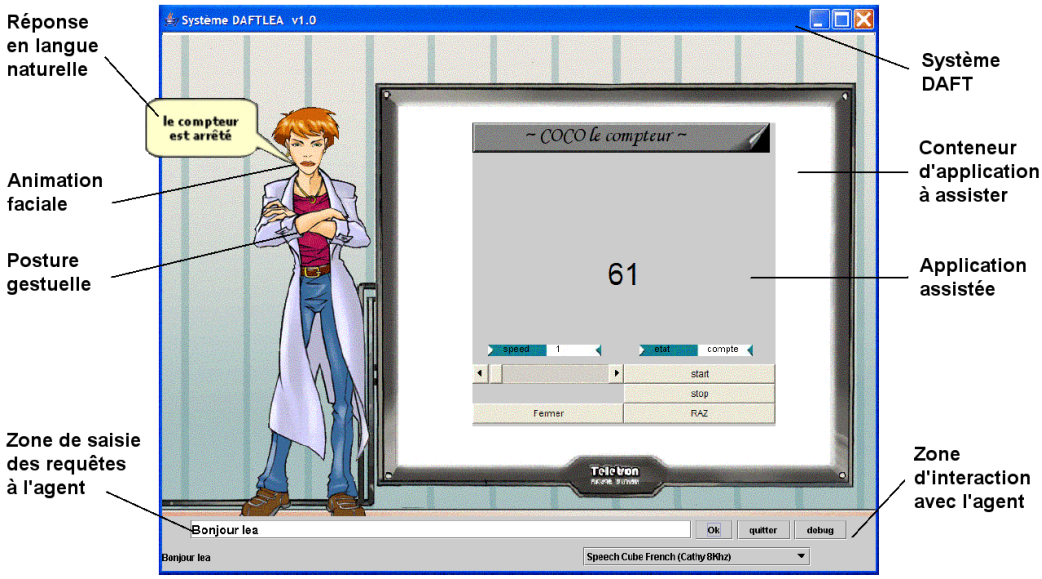


Figure 2.4 Exemple d'agent assistant DAFT 1.0 pour l'application Coco, un simple compteur

2.2.3.2 Apports pratiques

Le système `DAFT 1.0` a permis de disposer d'une base fonctionnelle à partir de laquelle il était possible de travailler pour améliorer, de manière relativement indépendante, les différents modules la constituant. De plus, en dépit de performances limitées, il a offert la possibilité de faire interagir les utilisateurs avec un système fonctionnel permettant de recueillir leur réactions en situation (et plus particulièrement, des requêtes en langue naturelle).

2.2.3.3 Critique et limitations

Analyse sémantique. GRASP a été construit à partir de quelques phrases d'exemples choisies arbitrairement et non forcément représentatives des requêtes réelles des utilisateurs, d'où un certain manque de robustesse dans le traitement de celles-ci. Les différents modules le composant étant interdépendants, la prise en compte de nouvelles phrases est d'autant plus difficile. Enfin, la modélisation formelle fournie n'est pas non plus forcément idéale pour l'assistance, et des éléments demeurent peu ou pas traités (comme les références).

Agent rationnel. Une limitation de `DAFT 1.0` provient de l'obligation d'associer à une requête donnée une unique réaction de nature monolithique. Cela signifie que pour introduire un nouveau comportement particulier, le concepteur de l'agent doit passer en revue toutes les heuristiques déjà existantes et éventuellement insérer le code correspondant à ce comportement. Le code inséré est ainsi dupliqué et se retrouve dépendant des heuristiques de comportement

déjà existantes. De plus, tout ajout ultérieur d'heuristique suppose de se poser de nouveau la question de l'insertion de ce comportement dans celle-ci, ce qui rend presque impossible la définition de nouvelles heuristiques par plus d'une seule personne.

Application et modèle. `DAFT 1.0` a été conçu pour s'interfacer avec des applications Java, technologie disponible sur de nombreux systèmes, y compris sur le Web par le biais des applets. Or l'évolution d'Internet depuis le début du projet DAFT s'est accompagnée d'une disparition progressive des applets Java, relativement lourdes à cause de la machine virtuelle, rendant partiellement obsolète l'approche choisie si l'on souhaite que le système puisse être exploité par les concepteurs d'applications actuelles. D'autre part, le modèle défini pour chaque application l'est sans prendre en compte la perception réelle qu'en ont ses utilisateurs, ce qui augmente les chances de non résolution des requêtes (puisque l'utilisateur réfère à des éléments qui peuvent ne pas exister du point de vue du modèle) : là aussi, une étude basée sur les retours de `DAFT 1.0` à ce niveau a permis une amélioration des performances.

Agent animé. Bien que permettant la déictique et l'expression de certaines émotions, les agents LEA sont de type "2D cartoon", ce qui peut avoir un impact direct sur les performances de l'utilisateur. Baylor & Kim [2004] ont notamment montré, dans le cadre d'agents pédagogiques, qu'un réalisme visuel accru de l'agent (passage d'une représentation de type "2D cartoon" à une représentation "3D réaliste"[3]) permettait une amélioration des performances en termes d'apprentissage, essentiellement chez les utilisateurs masculins. Par ailleurs, le mode de présentation actuel est peu flexible : il contraint à inclure l'application à assister dans l'environnement Java de `DAFT 1.0`, ce qui a pour effet de réduire l'espace disponible pour l'application et de générer une rupture visuelle avec le style graphique voulu par le concepteur d'application.

2.2.4 DAFT 2.0 (2006–2010)

D'après les critiques faites à la version `DAFT 1.0`, plusieurs objectifs ont donc été fixés pour établir une nouvelle version du système, ce qui a des conséquences directes sur l'architecture même du système, comme on peut le voir sur la figure 2.5. Nous nous contentons dans cette section de lister ces objectifs et les pistes suivies pour les atteindre, leur mise en application faisant l'objet des chapitres suivants.

2.2.4.1 Objectifs

L'analyseur sémantique GRASP 2.0. À ce niveau, nous avons identifié trois points nécessitant une amélioration par rapport à la version 1.0 :

[3]Plus exactement "2,5D", dans la mesure où si un modèle 3D de l'agent a été réalisé, son implémentation se fait à l'aide d'images le représentant sous un nombre limité d'angles de vue (pas de rotation à 360°).

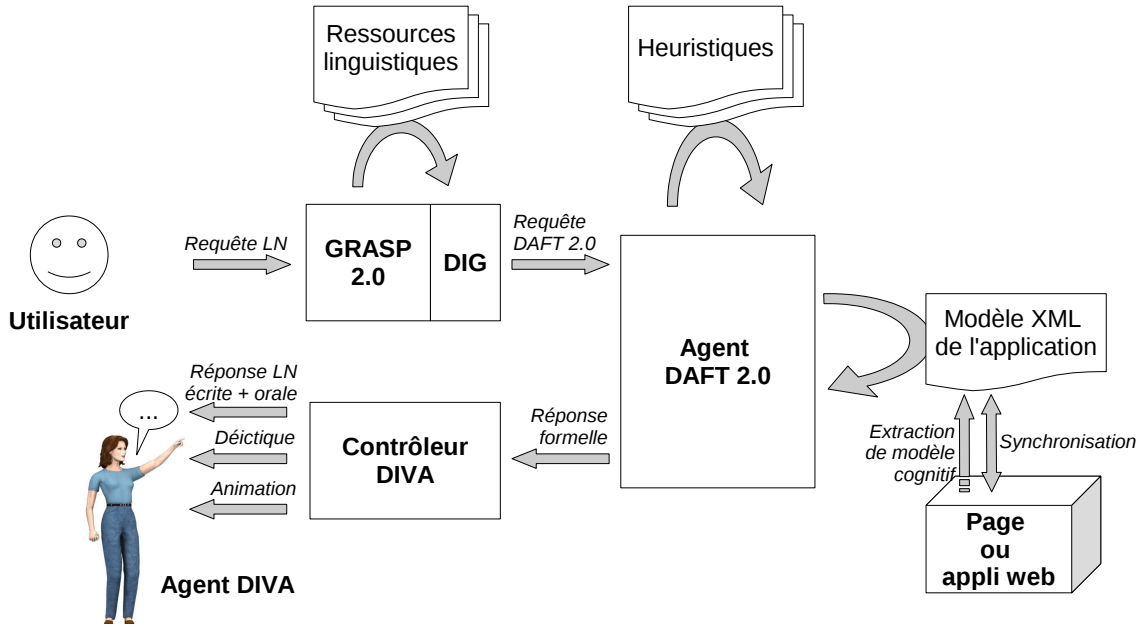


Figure 2.5 Architecture du système `DAFT 2.0`

Amélioration de la robustesse : afin d'améliorer les performances générales du système, il est essentiel de se baser sur des requêtes représentatives des besoins réels des utilisateurs novices afin de diminuer le nombre de mots ou expressions non reconnues en entrée par le système. L'idée-clé est donc de procéder dans un premier temps à un *recueil de corpus* dont l'analyse doit permettre de réaliser une adaptation des différents outils de la chaîne de TALN, ce qui est particulièrement utile pour les modules ayant recours à la fréquence d'apparition de différents phénomènes linguistiques, comme pour la désambigüisation sémantique.

Possibilité d'améliorations incrémentales : l'augmentation de la robustesse apportée par l'approche corpus est fortement dépendante de la qualité de celui-ci. Dans la mesure où de nouvelles requêtes d'utilisateurs peuvent être recueillies tout au long de l'utilisation du système, le corpus est amené à croître constamment, tant en quantité qu'en qualité, la seconde découlant de la première. Il est alors essentiel de pouvoir disposer d'un outil d'analyse flexible, c'est-à-dire capable de s'adapter pour que la qualité de l'analyse fournie suive les améliorations successives de la qualité du corpus. Cela requiert donc une séparation stricte entre les données et les processus d'analyse fondés sur celles-ci, soit une *externalisation des ressources linguistiques*.

Modélisation plus fine des requêtes : à supposer que tous les mots et expressions soient reconnus en entrée, encore faut-il que le système soit capable de capturer les nuances sémantiques entre eux. Cela requiert donc une amélioration sensible du modèle fondé sur les 23 classes d'actes de dialogue du tableau 2.1 associés à des références, en définissant un nouveau langage de requêtes formel, *DAFT 2.0*, capable de modéliser plus précisément les intentions réelles des utilisateurs et plus adapté aux requêtes émises dans le contexte de l'assistance.

La justification du besoin d'un corpus spécifique pour le système DAFT constitue le sujet du chapitre 3 tandis que sa prise en compte pour l'amélioration du module GRASP sera présentée dans les chapitres 4 et 6, dans lequel on gardera également à l'esprit le besoin d'externalisation des ressources linguistiques. La question de l'amélioration de la précision du langage sera quant à elle discutée dans le chapitre 5.

L'agent rationnel DAFT 2.0 : $\mathtt{A}_R$ Pour pallier les problèmes mentionnés en 2.2.3.3 et permettre une complexification des comportements potentiels de l'agent tout en facilitant la définition de ceux-ci, nous souhaitons que l'agent rationnel (que nous nommerons $\mathtt{A}_R$ dans DAFT 2.0) prenne en compte les trois caractéristiques suivantes (illustrées sur la figure 2.6) :

Factorisation des réactions : afin de simplifier l'écriture d'heuristiques de réaction, nous nous proposons de factoriser leur code en permettant la définition de routines correspondant à des situations classiques, auxquels les heuristiques de plus haut niveau peuvent faire appel. En définissant de tels éléments, le concepteur de réactions peut s'abstraire d'une rationalité "de bas niveau" (par exemple la vérification de l'existence d'une référence) et se concentrer sur des heuristiques plus génériques.

Parallélisation des réactions : si la factorisation permet à une réaction d'être éventuellement composée d'un ensemble de sous-réactions, encore faut-il que le concepteur connaisse les sous-réactions existantes. Il faut également que tout ajout d'une nouvelle sous-réaction passe par une analyse des réactions de "haut niveau" existantes pour voir si elle doit y être intégrée. Par conséquent, on peut pousser plus loin la logique en passant à une relation de type "plusieurs à plusieurs" dans laquelle une requête peut être associée à plusieurs réactions, par exemple en associant les réactions non pas à des requêtes complètes, mais à des sous-parties de la requête formelle.

Externalisation des réactions : l'objectif du système étant d'être utilisé par des concepteurs d'applications pour les doter d'un agent assistant, la logique de *modularisation* dans laquelle s'incrivent les deux points précédents peut être prolongée en séparant totalement le moteur de l'agent rationnel des réactions utilisées par celui-ci, de manière à pouvoir les externaliser. L'avantage d'une telle approche est double : d'une part il devient possible de définir des ensembles de réactions qui peuvent ou non être chargées par l'agent (pour étudier par exemple l'impact concret d'une heuristique donnée), d'autre part il est possible pour les concepteurs de "scripter" facilement leurs propres règles sans avoir à éditer le moteur de l'agent. On se dirige ainsi vers une approche collaborative où chaque concepteur d'applications peut (s'il le souhaite) passer du statut de consommateur d'une solution d'assistance clé en main à celui de contributeur.

Le fonctionnement de l'agent rationnel étant clairement affecté par la redéfinition de la structure des requêtes formelles *DAFT* qu'il reçoit en entrée, présentée dans le chapitre 5, l'évaluation de cette redéfinition de l'architecture d'association requête-réaction de l'agent rationnel pour prendre en compte ces trois objectifs sera abordée au niveau du chapitre 8.

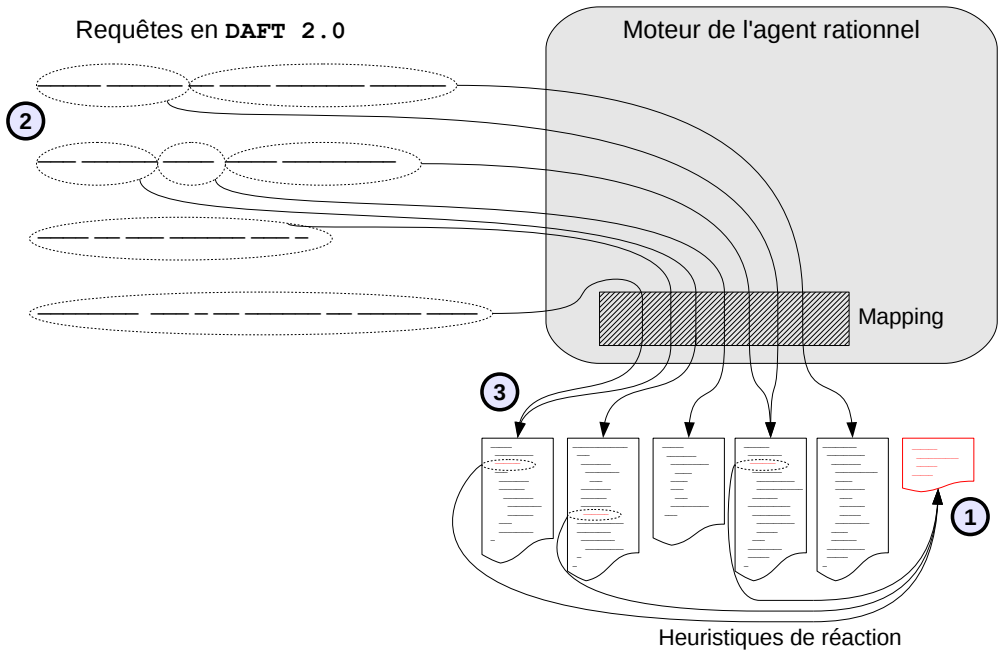


Figure 2.6 Agent rationnel $\mathtt{A}_R$ assurant le lien entre une requête formelle et des heuristiques de réaction, avec factorisation (1), parallélisation (2) et externalisation (3) des réactions, par rapport à la figure 2.3

L'application et son modèle. En vue de fournir une assistance plus précise et de rendre le système utilisable par le plus grand nombre de développeurs d'applications, deux points ont retenu notre attention au niveau du modèle d'application :

Extension de la portée des applications : le Web constitue une plate-forme idéale pour toucher un grand nombre d'utilisateurs, notamment novices. De plus, grâce au développement récent d'applications Web dynamiques basées sur le langage JavaScript en conjonction avec XML pour la représentation des données, l'analyse du code de pages Web permettrait donc de fournir des assistants à la fois pour la navigation et pour l'utilisation d'applications. Pour cette raison, `DAFT 2.0` a été envisagé comme résolument orienté vers les pages Web et les applications ou services développés en JavaScript.

Prise en compte des biais utilisateurs : pour pouvoir apporter une assistance efficace, il est utile de comprendre la manière dont les utilisateurs perçoivent l'application qu'ils utilisent, et notamment la façon dont ils se réfèrent aux éléments qui la compose en termes d'aggrégats perceptifs, qui peut sensiblement différer de l'implémentation effective sous-jacente (*cf.* figure 2.7). De plus, les études complémentaires menées par Hasson [2007] dans le cadre du projet ont montré que la représentation mentale que les utilisateurs ont de la tâche est au moins tout aussi importante.

L'étude et l'implémentation des améliorations décrites ci-dessus ont été le sujet des travaux de David Leray de 2005 à 2009 [Leray, 2009].

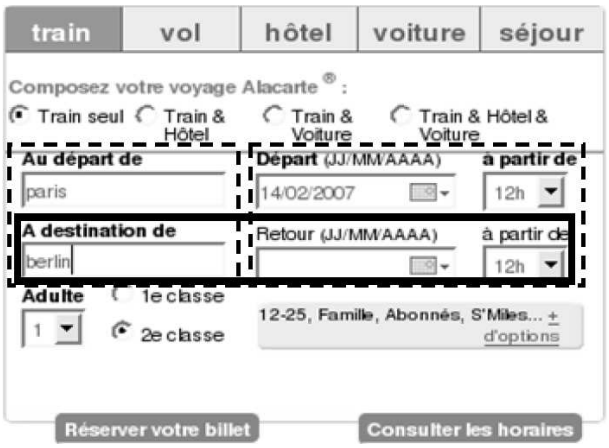


Figure 2.7 Exemple de différence entre la perception de l'utilisateur (encadrée en gras) et l'implémentation réelle (encadrée en pointillés) d'un formulaire sur un site Web permettant la réservation de billets de train [Sansonnet & Leray, 2007]

L'agent virtuel animé.

Réalisme accru : bien que Norman [1994] souligne qu'un agent trop réaliste visuellement génère des attentes plus grandes en termes de performances, et donc un risque de déception plus conséquent si elles ne sont pas à la hauteur de ces attentes, on a vu dans les critiques de `DAFT 1.0` qu'il y avait de réels avantages à disposer d'un avatar plus réaliste visuellement, à la fois en termes de représentation (personnages humanoïdes générés en 3D avec des textures plus détaillées) et d'animation (fluidité accrue dans les mouvements avec une interpolation au lieu de passer sans transition d'une animation à l'autre).

Intégration Web : pour les raisons invoquées précédemment, il est également nécessaire d'adapter les agents animés utilisés dans `DAFT 1.0` en vue d'un déploiement sur le Web, c'est-à-dire les rendre capables de s'insérer de manière la moins intrusive possible sur une page Web (page classique ou bien contenant une application ou service). Techniquement, deux solutions sont possibles a priori : l'ajout en surimpression de la page ou l'ajout dans une frame[4] supplémentaire. Si cette dernière option est plus simple à implémenter, elle a comme inconvénient, outre celui de faire appel à une méthode aujourd'hui peu recommandée, de réduire la zone d'affichage normalement dévolue à l'application. La première option semble donc préférable, mais afin de pouvoir s'adapter à tout type de page sans forcer le développeur d'application à gérer lui-même le positionnement de l'agent sur celle-ci, elle induit que l'agent animé doit pouvoir être déplacé (par glisser-déposer) et réduit (ou masqué) lorsque l'utilisateur ne souhaite pas faire appel à lui.

Afin de répondre à ces deux objectifs, un nouveau système d'agents visuellement réalistes et intégrés au Web a été développé, les agents DOM Integrated Virtual Agents (DIVA). Comme leur nom l'indique, ils sont complètement intégrés au Document Object Model (DOM) de la page Web dans la mesure où ils sont implémentés par le biais d'une imbrication de balises HTML de type `<div>` contenant des images correspondant aux différentes parties

[4] Espace séparé (visuellement et conceptuellement) du reste de la page Web dans laquelle il est inclus et où d'autres contenus peuvent prendre place – sans rapport avec la frame au sens du glossaire.

du corps de l'agent[5]. Cette intégration au DOM permet de gérer leur animation par le biais de scripts (écrits ici en JavaScript).

2.2.4.2 Notations

Nous avons vu dans les sections précédentes que l'acronyme DAFT pouvait être utilisé pour faire référence à différents éléments. Pour clarifier les choses, nous distinguerons donc dans les chapitres suivants :

- le "projet DAFT" : cadre de l'ensemble de ces travaux ;
- le système `DAFT` (majuscules, police romaine à chasse fixe) : ensemble de la chaîne de traitement ;
- le module DAFT (majuscules, police romaine) : correspondant au module de l'agent rationnel au sein du système `DAFT` ;
- le corpus *Daft* (minuscules, police italique) : recueilli à l'aide du système `DAFT 1.0`, utilisé pour la conception de `DAFT 2.0` ;
- le langage *DAFT* (majuscules, police italique) : utilisé pour la représentation formelle des requêtes des utilisateurs. On distinguera la version de `DAFT 1.0` (notée *DAFT 1.0*), résumée dans la table 2.1, de la version à concevoir pour `DAFT 2.0` (notée *DAFT 2.0*).

2.3 Conclusion

Nous avons présenté dans ce chapitre la première version du système `DAFT` et proposé un certain nombre d'améliorations relatives à l'agent animé utilisé (passage à un mode "3D réaliste" et intégration aux pages Web), à l'accès au modèle de l'application assistée (rendu possible par l'intégration au DOM), à l'agent rationnel (modularisation des heuristiques de réaction) et à la chaîne de traitement des requêtes en langue naturelle (modélisation plus fine et système plus robuste avec possibilité d'améliorations incrémentales). Nous allons dans la partie suivante de cette thèse nous concentrer sur ce dernier point.

[5]Notons que pour prendre en compte les problèmes de ralentissement qui peuvent apparaître avec une connection bas débit pour télécharger toutes les animations des agents réalistes DIVA, une version Web des agents "2D cartoon" LEA a également été réalisée.

Deuxième partie

Chaîne de traitement

Chapitre 3

Constitution d'un corpus de requêtes d'assistance : le corpus *Daft*

Sommaire

Ce chapitre introduit un des points essentiels faisant l'originalité de l'approche suivie dans cette thèse, à savoir le fondement de la chaîne de traitement linguistique sur un corpus de requêtes d'assistance. Nous discuterons tout d'abord de l'intérêt de l'approche corpus de manière générale, puis plus particulièrement dans notre cas, afin de démontrer notre besoin d'un corpus spécifique par rapport aux corpus de dialogue déjà existants. Puis, nous présenterons et justifierons les différentes étapes suivies pour constituer ce corpus, que nous nommerons le "corpus *Daft*" dans ce qui suit.

3.1 L'approche corpus pour le dialogue

3.1.1 Origines et intérêt

La linguistique traditionnelle est très attachée à l'étude de la langue comme structure, en faisant grandement usage de phrases construites pour les besoins d'illustration des phénomènes étudiés, à l'image des "donkey sentences" proposées par Geach [1962]. Même dans le cas où la langue est étudiée sous l'angle des interactions inter-individuelles, les premiers modèles de représentation du discours comme la Discourse Representation Theory (DRT) [Kamp, 1981; Kamp & Reyle, 1993] s'intéressent finalement plus à la modélisation de phénomènes comme les "donkey sentences" susmentionnées, sans se soucier réellement de savoir s'ils apparaissent fréquemment dans l'usage.
Toutefois, dans les années 1990, suite aux succès des méthodes fondées sur l'utilisation de **corpus**, notamment dans le domaine de la traduction automatique, ce type d'approches est devenu de plus en plus courant. Depuis lors, le TALN tend de plus en plus à chercher un ancrage dans la réalité en fondant ses études et travaux sur des corpus de phrases propres aux domaines étudiés. Cette approche permet à la fois de quantifier et de mieux qualifier les phénomènes linguistiques étudiés. En outre, dans le cadre du développement d'outils visant à traiter des phénomènes particuliers (résolution d'anaphores, correction orthographique, systèmes de questions-réponses (QR), ...), l'utilisation d'un corpus fournit une mesure relativement objective permettant :

- d'évaluer la performance de l'outil réalisé et notamment de quantifier les apports des améliorations apportées à celui-ci ;
- de comparer entre eux différents outils similaires appliqués à un même corpus servant de référence.

Bien entendu, la performance réelle obtenue par l'outil dans son cadre d'utilisation effectif est totalement dépendante de la représentativité du corpus, c'est-à-dire du degré de proximité entre son contenu et les textes ou phrases auxquels sera confronté l'outil dans le contexte où il doit être réellement appliqué. À partir de là, la constitution d'un corpus est loin d'être une simple contrainte technique dont on pourrait s'acquitter promptement pour se focaliser sur la suite. Au contraire, il s'agit d'une étape extrêmement importante qui conditionne totalement la réussite future de l'outil développé. En effet, même si on imagine un cycle de développement réalisé selon un modèle en spirale [Boehm, 1988], l'utilisation d'un corpus adéquat permet au minimum d'éviter un certain nombre d'itérations généralement nécessaires pour adapter le système en fonction des retours des tests (par exemple, s'il s'agit simplement de compléter des bases de vocabulaire). Mais se fonder sur un corpus d'observations peut surtout éviter de graves erreurs conceptuelles, qui seraient elles susceptibles de remettre en cause le fonctionnement même de l'outil. Ainsi, par exemple, un oubli de classes de requêtes pour un système de classification avec apprentissage préalable est susceptible de rendre le fonctionnement de celui-ci très aléatoire.

On peut retracer l'idée consistant à comparer le résultat de différentes analyses à partir d'exemples issus d'un corpus commun de phrases au début des années 1990, avec les travaux précurseurs de Abney *et al.* [1991] dans le cas d'annotations manuelles, la mise en pratique de cette méthodologie avec des systèmes de TALN n'arrivant que quelques années plus tard, par exemple avec Black *et al.* [1993]. Depuis lors, l'idée d'évaluation et de comparaison à partir d'un corpus s'est progressivement imposée à la communauté du TALN. On peut citer par exemple à l'échelle française les initiatives de la plate-forme Infrastructure d'EValuation à ELDA (EVALDA) de 2003 à 2006 visant à l'élaboration de campagnes d'évaluation dans des domaines variés du TALN : EASy pour l'analyse syntaxique [Paroubek *et al.* , 2008], EqueR pour les systèmes de questions-réponses [Ayache *et al.* , 2006], MEDIA pour la compréhension du dialogue [Devillers *et al.* , 2004]... L'important développement de ce domaine depuis le début des années 2000 peut d'ailleurs se voir avec la montée en puissance de la conférence biennale Language Resources and Evaluation Conference (LREC) [Calzolari *et al.* , 2008]. Au niveau européen, le 7e programme cadre de l'Union Européenne finance la tenue annuelle de l'atelier Cross Language Evaluation Forum (CLEF) [Peters *et al.* , 2008], visant à évaluer les systèmes travaillant sur des documents multilingues (traduction automatique, recherche multilingues, *etc.*). Au niveau mondial enfin, depuis 1992, la conférence Text REtrieval Conference (TREC) (devenue TAC) organise chaque année une évaluation de différents systèmes de recherche d'informations sur un corpus commun de documents [Voorhees & Buckland, 2008].

3.1.2 Panorama de corpus existants

L'un des principaux problèmes rencontrés dans la linguistique de corpus est la difficulté d'accès aux corpus eux-mêmes. Comme on l'a vu dans la section 3.1.1, ils constituent une ressource précieuse pour le développement d'outils de TALN. Ainsi, l'association European Language Ressources Association (ELRA)[1] qui vise à recenser, rassembler et diffuser différents corpus commercialisés via Evaluations and Language resources Distribution Agency (ELDA) recense, à la mi-2009, 610 corpus écrits (toutes tailles, types et langues confondus). On peut en distinguer quelques-uns en particulier, réunis dans la table 3.1.

3.1.2.1 Quelques corpus historiques

Historiquement, le premier corpus assemblé en vue d'être étudié de manière statistique en TALN est le corpus Brown [Francis, 1964; Kučera & Francis, 1967], recueilli à l'université éponyme entre 1963 et 1964, réunissant 500 textes pour un total d'un million de mots issus de genre différents (journaux, romans, textes scientifiques...) et dont le seul point commun est leur date de publication (1961), afin d'offrir un panel des usages de l'anglo-américain à un instant donné. Son pendant dans le cadre de la langue parlée est le corpus Sankoff-Cedergren [Sankoff & Sankoff, 1973], recueilli à Montréal en 1971 dans le cadre d'entrevues

[1] http://www.elra.info

semi-dirigées avec 120 locuteurs de langue française. Bien que ce corpus ou son successeur recueilli dans des conditions similaires en 1984 [Thibault & Vincent, 1990] pourraient être considérés comme des corpus de dialogue (dans la mesure où ils contiennent les interventions de l'interviewer et de l'interviewé), c'était l'aspect lexicographique qui intéressait alors les chercheurs l'ayant recueilli. Le développement de l'exploitation des corpus pour les outils de TALN à partir des années 90 a stimulé le recueil de corpus généralistes de très grande taille, toujours essentiellement de langue anglaise, qu'ils soient de taille fixe comme le British National Corpus (BNC) ou l'Oxford English Corpus (OEC) – le plus important à ce jour avec 2 milliards de mots – ou enrichis de manière régulière à la manière du Corpus of Contemporary American English (COCA) composé de façon à pouvoir suivre l'évolution de la langue au cours des années. En français, l'un des plus conséquents corpus généraliste est MULTITAG [Paroubek, 2000], qui présente en outre la particularité d'être historiquement lui-même issu d'une campagne d'évaluation d'**étiqueteurs morphosyntaxiques** [Adda *et al.* , 1999].

3.1.2.2 Corpus de dialogues orientés tâches

Si l'on s'intéresse plus particulièrement aux corpus contenant des interactions ou dialogues plus proches du cadre de cette thèse et donc davantage susceptibles de nous intéresser ici, il faut attendre les travaux de Allen *et al.* [1995] sur le système TRAINS pour trouver un exemple pionnier de corpus recueilli en vue de la conception d'un SDHM. Il s'agit d'un corpus de dialogues oraux entre deux humains (un utilisateur ayant une tâche à accomplir et un assistant) qui ont permis de construire un système d'analyse linguistique, de planification et de raisonnement dans le cadre de dialogues visant à planifier le transport de marchandises d'un point à un autre par voie ferrée. Dans la mesure où les dialogues n'étaient ni des entrevues comme dans Sankoff & Sankoff [1973], ni des conversations de la vie quotidienne mais des dialogues entre deux humains visant à l'accomplissement d'une tâche déterminée, on qualifiera plus simplement par la suite ce type de corpus comme étant "orienté tâche". Ce sont aussi ceux qui se rapprochent le plus du cadre de la fonction d'assistance, où le dialogue engagé entre l'utilisateur novice et l'agent assistant a pour objectif l'accomplissement d'une tâche par l'usager au sein de l'application. Le tableau 3.2 résume quelques caractéristiques des principaux corpus *librement accessibles*[2] de ce type que nous avons pu recenser.
Air France, SNCF et CIO, recueillis à la fin des années 80 par Morel [1989] visent également à réaliser un système de dialogue automatique devant se substituer à l'opérateur humain dans des tâches relativement bien circonscrites (réservation de billets et exposé d'options d'orientations disponibles). Ils contiennent d'ailleurs à la fois des interactions entre humains, recueillies a priori pour la réalisation du système, et des interactions homme-machine a posteriori, une fois le système réalisé. Le corpus GOCAD de Chapelier *et al.* [1995] contient lui uniquement le recueil d'interactions orales entre des utilisateurs et un système d'agent assistant guidant des utilisateurs d'un logiciel de modélisation de surface dans la réalisation

[2]De nombreux corpus un peu anciens ne sont en effet pas vraiment disponibles librement.

Nom	Taille	Modalité	Langue	Genre	Origine
Brown	500 textes, 1 million de mots	E	AA	Gén	Textes divers publiés aux États-Unis en 1961 [Francis, 1964; Kučera & Francis, 1967]
Sankoff-Cedergren	120 entrevues	O	F	Q/R	Entrevues réalisées à Montréal en 1971 [Sankoff & Sankoff, 1973]
BNC[a] (British National Corpus)	100 millions de mots	E & O (90/10)	AB	Gén	Textes (romans, journaux, rapports, lettres...) et interventions orales (non scriptées, dans des situations formelles ou informelles), recueillis de 1991 à 1994
COCA[b] (Corpus of Contemporary American English)	150 000 textes, 385 millions de mots (en 2009)	E & O (80/20)	AA	Gén	Enrichi de 20 millions de mots par an depuis 1990. Divisé en 5 catégories : oral, fiction, magazines, journaux, revues scientifiques
ICE[c] (International Corpus of English)	1 million de mots (500 textes de 2 000 mots)	E & 0 (40/60)	A	Gén	Majoritairement issu de données orales, recueilli depuis 1989 dans les pays dont l'anglais est une des langues officielles
Oxford English Corpus[d]	2 milliards de mots	E	A	Gén	Sources extrêmement variées (journaux, romans, sites Web, chat...) avec enregistrement de méta-informations pour chaque document de la base (XML)
MULTITAG	1 million de mots	E	F	Gén	Extraits du journal "Le Monde" et textes littéraires du XV^{e} au XX^{e} siècle, annotés en termes de POS [Paroubek, 2000]
Switchboard Dialog Act	1.4 millions de mots (1 155 dialogues)	O	AA	Dial	Conversations téléphoniques spontanées entièrement retranscrites [Jurafsky *et al.* , 1998]. Sous-ensemble de Switchboard (2 430 dialogues) [Godfrey & Holliman, 1997]

Tableau 3.1 Corpus francophones et anglophones significatifs
Modalité : E = Écrit, O = Oral, E & O (X/Y) = X% écrit et Y% oral
Langue : F = Français, A = Anglais, AA/AB = Anglais américain/britannique
Genre : Gén = Généraliste, Q/R = Questions/Réponses, Dial = dialogue

[a] http://www.natcorp.ox.ac.uk
[b] http://www.americancorpus.org
[c] http://ice-corpora.net/ice
[d] http://www.askoxford.com/oec

de certaines tâches.
Parmi les corpus orientés tâche existants, on peut également citer Map Task, constitué de 128 dialogues recueillis par Anderson *et al.* [1991] dans un contexte où une personne doit suivre et reproduire un itinéraire sur une carte géographique à partir d'instructions données par une seconde personne (l'assistant) dotée d'une carte similaire. Plusieurs corpus ont depuis été constitués selon un protocole similaire, comme par exemple par Post [2000] en français.

3.1.3 Le besoin d'un corpus pour le projet DAFT

Le recueil d'un corpus de dialogue demeure une opération coûteuse :

- techniquement : pour mettre en place une version préliminaire du système et/ou une plate-forme de type Magicien d'Oz (MOZ),
- humainement : pour disposer d'un échantillon de sujets humains suffisamment important et représentatif de la catégorie que l'on souhaite étudier (des utilisateurs novices dans notre cas, ce qui exclut de faire appel, comme souvent, à des collègues ou étudiants en informatique).

Par ailleurs, on a vu qu'un certain nombre de corpus de requêtes dans des domaines parfois très proches de l'assistance (comme GOCAD ou TRAINS) d'une part existaient, et d'autre part étaient accessibles librement. Il semble donc légitime de considérer la possibilité d'avoir recours à un des corpus présenté en 3.1.2.2. Plusieurs raisons nous ont toutefois fait préférer la collecte d'un corpus dédié, que nous nommerons, rappelons-le, "le corpus *Daft*".

3.1.3.1 Avantages pratiques de la modalité écrite

Dans le projet DAFT, les agents assistants sont amenés à être déployés sur des pages Web et nous avons donc opté pour une saisie par écrit des requêtes. En effet, bien que le clavier ne permette pas une transmission aussi rapide des requêtes (surtout pour des utilisateurs novices tapant plus lentement), l'interaction orale est encore peu répandue, notamment sur le Web. Elle peut donc potentiellement apparaître comme déstabilisante, voire intrusive dans un contexte d'assistance (nous partageons sur ce point le point de vue de Isbister *et al.* [2000]). En outre, la modalité orale présente des contraintes techniques qui nous paraissent trop importantes pour être utilisée dans le cas d'un agent destiné essentiellement à des utilisateurs novices, qui se contentent généralement d'une configuration standard pour accéder au Web. En effet, en supposant que l'utilisateur dispose d'un micro offrant une qualité sonore correcte, il faut encore disposer d'un système de reconnaissance de la parole. Celui-ci doit être présent au choix :

- sur la machine de l'utilisateur : ce qui était marginal au début du projet en 2005, et demeure encore relativement rare en 2009, en dépit de l'intégration d'un système de ce genre au sein de Windows Vista (disponible depuis 2007) qui nécessite une phase d'apprentissage que peu d'utilisateurs suivent.

Nom	Taille	Modalité	Interlocuteur	Mode	Langue	Origine
TRAINS	98 dialogues 5900 tours ~ 60 tours/dial	O	H	?	A	Simulation de dialogues avec un agent assistant pour le routage de marchandises par voie ferrée [Heeman & Allen, 1995]
HCRC Map Task	128 dialogues	O	H	Direct	A	Suivi d'itinéraire sur une carte géographique [Anderson *et al.* , 1991]
Map Task français	4 dialogues 638 tours ~ 160 tours/dial	O	H	Direct	F	Suivi d'itinéraire sur une carte géographique [Post, 2000]
Air France	95 dialogues 4 849 tours ~ 51 tours/dial	O	H	Tél	F	Réservation de billets d'avion [Morel, 1989]
SNCF	120 dialogues 5 479 tours ~ 46 tours/dial	O	H	Tél	F	Demande de renseignements et réservation de billets de train [Morel, 1989]
	140 dialogues 2 787 tours ~ 20 tours/dial		M			
CIO	8 dialogues 2 056 tours ~ 257 tours/dial	O	H	Tél	F	Demande de renseignements auprès d'un centre d'orientation universitaire [Morel, 1989]
	26 dialogues 2 569 tours ~ 99 tours/dial		M			
GOCAD	32 dialogues 3 232 tours ~ 101 tours/dial	O	M	Micro	F	Contrôle d'une application de modélisation de surface [Chapelier *et al.* , 1995]
OZKAN	33 dialogues 1 678 tours ~ 51 tours/dial	O	H	Direct	F	Réalisation d'un dessin à partir d'instructions [Ozkan, 1994]
Bugzilla	128 000 rapports de bugs 1 200 000 commentaires ~ 9 commentaires/rapport	E	H	Indirect	A	Échanges par le biais d'un forum en vue de résoudre les bugs de la suite Mozilla [Ripoche, 2006]

Tableau 3.2 Corpus de dialogues francophones et anglophones orientés tâches
Modalité : E = Écrit, O = Oral
Interlocuteur : H = Humain, M = Machine
Langue : F = Français, A = Anglais

– sur un serveur distant : auquel cas il est nécessaire de disposer d'une connexion haut débit pour transmettre le signal audio, ce qui n'était pas encore aussi répandu il y a quelques années.

Pour ces raisons, il nous a semblé préférable d'opter pour une interface dotée d'une zone de saisie des requêtes au clavier.

Cependant, à l'exception du cas assez particulier du corpus Bugzilla [Ripoche, 2006], où il s'agit d'interactions non temps réel entre un grand nombre de personnes, tous les corpus de dialogues orientés tâche présentés dans le tableau 3.2 sont des recueils d'interactions orales en face à face, retranscrites à l'écrit pour permettre leur traitement. Or de nombreuses études ont montré que la modalité orale entraînait la génération de tours de dialogues sensiblement plus longs [Kelly & Chapanis, 1977], ce qui a pour effet direct d'exclure la plupart des corpus disponibles de la liste des corpus potentiellement exploitables pour nos travaux.

3.1.3.2 Relative rareté des ressources en français

Pour des raisons à la fois institutionnelles et pratiques (disposer de sujets parlant la langue française), nous souhaitons réaliser un système traitant la langue française. Si la langue importe moins lorsqu'il s'agit de travailler au niveau de la **pragmatique** pour identifier des actes de dialogue, il est en revanche crucial pour construire un analyseur grammatical "sur mesure"[3] de disposer d'un échantillon représentatif dans lequel figurent des mots et locutions représentatives du vocabulaire des futurs utilisateurs du système.
Or l'importante proportion de corpus en français figurant dans le tableau 3.2 ne témoigne pas de la réalité des choses : ils apparaissent surreprésentés dans la mesure où nous leur avons accordé une attention toute particulière, mais il est évident que les corpus (surtout de taille conséquente) sont en anglais, et se retrouvent ainsi exclus de la liste des choix possibles.

3.1.3.3 Spécificités de l'interaction langagière Homme-Machine

Notre objectif étant la réalisation d'un système d'analyse de requêtes Homme-Machine, il nous faut recueillir des requêtes représentatives de la façon dont se comporteront les utilisateurs finaux du système. Cependant, on a pu constater que les utilisateurs se comportent de manière sensiblement différente lorsqu'ils interagissent avec une machine plutôt qu'avec un humain, tout du moins dans le cas d'interactions orales [Morel, 1989; Luzzati, 1995], ce qui suggère que l'utilisation d'un corpus d'interactions homme-homme n'est pas forcément très appropriée. Pire, il semble que leurs attentes elles-mêmes vis-à-vis du système soient alors différentes de celles qu'ils ont face à un humain dans le même contexte d'activité [Amalberti *et al.* , 1993], or ce sont bien ces attentes que l'on cherche à identifier au final dans l'analyse sémantique

[3]Par "sur mesure", nous entendons ici l'idée de définir un analyseur dont les performances soient optimales sur un sous-domaine de la langue circonscrit au mieux. Cette restriction permet d'éviter les problèmes de passage à l'échelle qui se posent immanquablement lorsque l'on s'intéresse à la langue dans son ensemble.

puis pragmatique des actes de dialogue.
De plus, les caractéristiques et limitations propres à chaque système influencent de fait les requêtes et dialogues de corpus d'interactions homme-machine. Dès lors, se pose la question de la transposabilité des requêtes : il n'est pas évident de considérer que les requêtes recueillies avec un système A seront identiques à celles qui seraient recueillies avec un système B, ni même forcément pertinentes dans le contexte d'interaction du système B.

3.1.3.4 Dialogue ou interface en langue naturelle ?

Lors d'une expérience annexe dans laquelle l'agent assistant force les utilisateurs à produire plusieurs tours de parole en reformulant ou en précisant leur question, nous avons pu constater dans [Hasson, 2007] que ceux-ci acceptent au maximum trois tours (au cours desquels ils précisent leur requête) avant d'abandonner le système d'aide. Ces résultats sont synthétisés par la figure 3.1 qui montre que moins de 5% des sujets insistent en posant une quatrième fois leur question (la majorité abandonne après la deuxième ou troisième itération). De plus, lors de la quatrième requête, on observe que le nombre moyen d'éléments d'information (pour "aider" le système à comprendre la requête) cesse d'augmenter et diminues même substantiellement, signe d'un certain renoncement. Ce résultat est d'ailleurs corroboré par Capobianco & Carbonell [2002] qui rappellent que les utilisateurs en quête d'assistance se satisfont souvent d'une interaction plus limitée [Harris *et al.* , 1980; Falzon *et al.* , 1986].

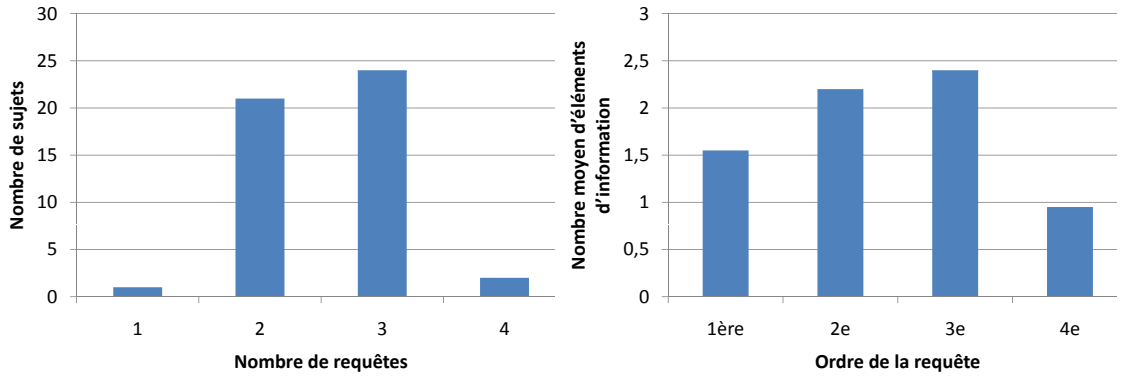


Figure 3.1 Phénomène de répétition et précision des requêtes à trois reprises maximum (d'après [Hasson, 2007])

De plus, l'agent assistant que nous souhaitons concevoir n'est pas proactif, c'est-à-dire qu'à part peut-être au lancement de l'application pour signaler sa présence, il n'interagit avec l'utilisateur que lorsque celui-ci lui a préalablement adressé la parole. Ce choix vise à diminuer l'intrusivité de l'agent assistant, qui a souvent un effet contre-productif en incitant plutôt les utilisateurs à chercher un moyen de faire taire l'agent, comme le souligne Doyle [1999].
À partir de là, nous pouvons en déduire que les interactions seront généralement courtes, se limitant souvent à deux tours de parole (question – réponse), parfois à trois si l'utilisateur accuse réception de la réponse (question – réponse – commentaire) et au maximum et plus

rarement encore à quatre tours (question – réponse – demande d'éclaircissement – deuxième réponse). La question se pose donc de savoir si, finalement, un système d'assistance à base d'**agents conversationnels animés** doit réellement être conçu comme un SDHM et ne se rapprocherait pas plutôt d'une **interface en langue naturelle**, au sens d'Androutsopoulos & Aretoulaki [2003]. En effet, dans une interface en langue naturelle, on considère que le système reçoit des requêtes ponctuelles qui peuvent être traitées comme étant isolées et essentiellement indépendantes les unes des autres. Partant sur cette hypothèse, nous avons fait le choix de conserver les requêtes recueillies dans le cadre de la constitution du corpus *Daft* (décrite dans la section suivante) de manière isolée les unes des autres, plutôt que sous la forme de tours de parole.

3.2 Recueil du corpus *Daft*

3.2.1 Méthodologie

Afin de collecter un corpus de requêtes d'assistance, nous avons choisi de développer un système d'agents conversationnels animés lié à un premier moteur d'analyse permettant à l'agent de réagir aux requêtes des utilisateurs. À défaut d'être capable de réagir de manière pertinente dans la plupart des cas, le système permet au moins d'accuser réception de la requête. L'architecture de ce premier système, développé par Le Guern [2004], `DAFT 1.0`, a été présentée en 2.2.3. À partir de ce système, deux environnements ont été utilisés (suivant ainsi l'évolution technique, justifiée dans la section 2.2.4) avec différentes applications, de manière à recueillir deux corpus distincts.

3.2.1.1 Simples applets indépendantes

Une première campagne de recueil de requêtes d'utilisateurs a été menée en 2005. Une cinquantaine de sujets ont interagi avec une applet Java parmi trois disponibles (Coco, Hanoï et AMI) développées précédemment dans le cadre du projet Interviews[4] et décrites dans la table 3.3. Ces applets étaient incluses dans l'environnement de `DAFT 1.0` réalisé en Java qui se présentait sous la forme d'une application contenant trois parties distinctes (*cf.* figure 3.2) :

- un agent animé LEA ;
- la zone de saisie de requêtes à l'agent dans la partie inférieure de l'application ;
- la zone principale, vide par défaut, destinée à contenir l'application à assister.

Cette campagne a permis de constituer un premier corpus ($Daft_{app}$– pour **app**lets) formé de près de 3 000 requêtes : quelques-unes d'entre elles sont données en exemple dans le tableau 3.4.

[4] `http://www.limsi.fr/~jps/interviews/index.html`

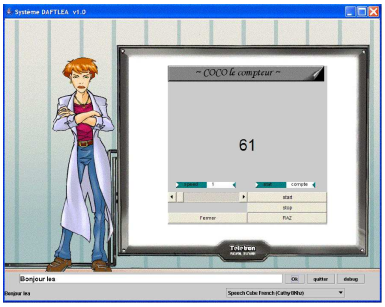
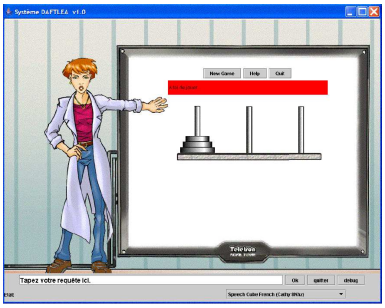
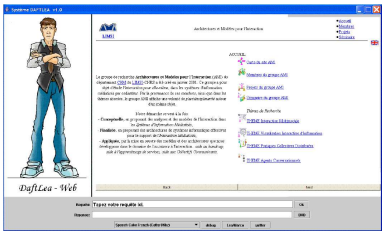
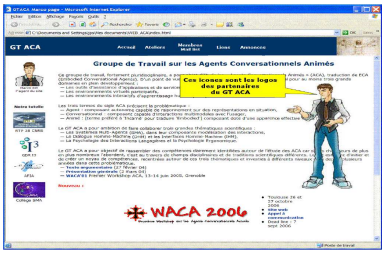
Application	**Description**	**Méthodologie de conception**
 Coco	Simple compteur (multithreadé) en Java. Déclenché ou arrêté par l'appui sur le bouton on/off et dont la vitesse peut être changée.	Application et ACA construits en même temps. Le modèle est une image exacte des éléments de l'application.
 Hanoï	Tours de Hanoï en Java. L'utilisateur doit déplacer la tour du piquet gauche au piquet droit par mouvements successifs des disques qui la composent, en ne pouvant placer un disque que sur un de taille supérieure.	Application puis ACA. Le modèle de l'application est obtenu a posteriori par filtrage du code Java.
 AMI	Site intranet du groupe AMI du LIMSI. Il peut être parcouru et édité par l'utilisateur via des commandes à l'agent.	Application et ACA construits à partir du modèle. Le site, inclus dans une applet, est entièrement contrôlé via le modèle de l'application stocké dans une base de données.
 Marco	Site internet du Groupe de Travail sur les Agents Conversationnels Animés (GTACA)[a] Agent WebLea intégré à la page (sans Java)	Application puis ACA, sans modèle. Phase 1 : recueil des messages des visiteurs. Phase 2 : ajout de réponses aux questions posées lors de la phase 1.

Tableau 3.3 Les trois applets et le site Web dotés d'un ACA utilisés pour le recueil du corpus $Daft_r$.

[a] http://www.limsi.fr/aca

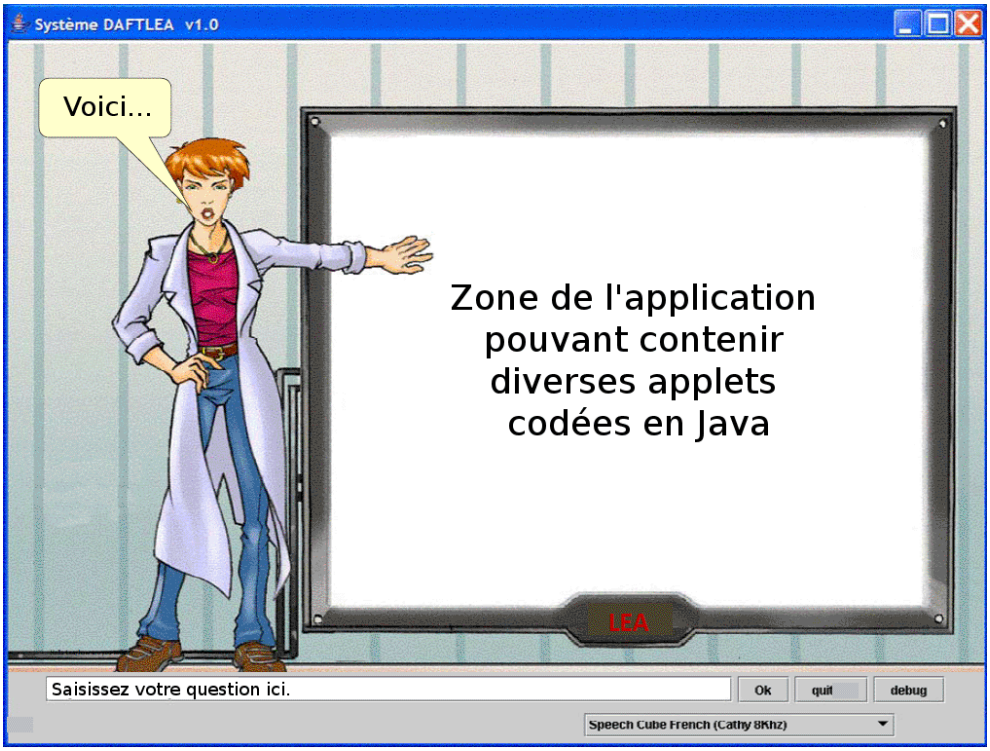


Figure 3.2 Agent `DAFT 1.0` au sein de l'environnement d'assistance intégré

N°	Requête d'utilisateur
1	actionne lebouton du haut
2	il y a une ressemblance entre la commande "stop" et "raz" ?
3	lance le compteur
4	il est tellement bête qu'il n'est même pas capable de dire quel disque !
5	tu parles anglais ?
6	joues
7	pourquoi ne peux-tu pas compter par pas négatif ?
8	Bon je me casse. Bye.
9	peut-on tricher ?
10	le compteur a du tomber en panne
11	comment faire pour gagner ?
12	puis-je te faire confiance ?
13	qu'est ce que c'est Coco le compteur ?
14	je suis Mr. Martin
15	y at-il un minimum pour la vitesse ?

Tableau 3.4 Exemples de requêtes d'utilisateurs recueillies, issues de $Daft_{app}$

3.2.1.2 Agents sur le Web

Une seconde campagne de recueil a été menée en 2006 avec une architecture de système sensiblement modifiée, avec des agents WebLea dotés du même moteur que dans la campagne précédente (en termes de fonctionnalités) mais réécrit en JavaScript. Les agents assistants sont ainsi totalement intégrables au sein d'une page Web, au lieu d'intégrer comme précédemment l'application à l'environnement contenant l'agent (*cf.* figure 3.3). Ce changement technique libère de la contrainte d'organisation qui obligeait à faire se déplacer les sujets dans un environnement contrôlé lors de la campagne précédente, et permet donc de disposer d'un échantillon de sujets potentiellement plus ciblé et représentatif des utilisateurs novices auxquels les agents assistants sont en premier lieu destinés.

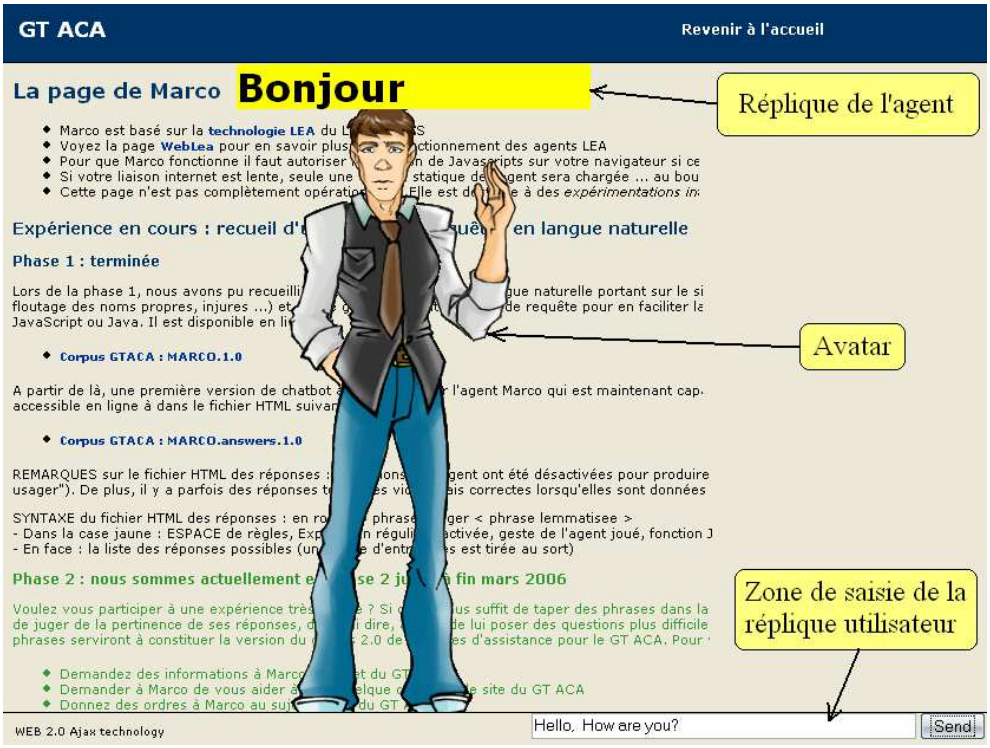


Figure 3.3 Agent `DAFT 1.0` déployé directement sur le Web

Le seul site Web utilisé lors de cette campagne est celui le site du GTACA (*cf.* la dernière ligne du tableau 3.3) : une cinquantaine de visiteurs de celui-ci ont interagi avec l'agent Marco présent sur le site, permettant de recueillir environ 1 500 requêtes qui constituent ainsi le corpus que nous nommerons $Daft_{web}$, et dont quelques exemples figurent dans le tableau 3.5. En faisant l'union de ces requêtes avec celles de $Daft_{app}$, on obtient le corpus de l'ensemble des requêtes recueillies, auxquels nous ferons référence sous le terme de $Daft_r$.

N°	Requête d'utilisateur
1	comment participer au GT ACA ?
2	bouge toi de devant la table ! !
3	que peut bien vouloir dire ACA ?
4	peut je changer ton taille ?
5	marco ?
6	aller à la page précedente
7	y a-t-il souvent des reunions ?
8	vas t'en
9	j'ai une critique à faire sur le site
10	t'es marié
11	donne moi le mail du webmaster de ce site
12	parle moi de gt aca
13	je voudrais m'abonner stp
14	tu peux me confirmer que je suis bien membre du GT ?
15	ajouter "jean-paul"

Tableau 3.5 Exemples de requêtes d'utilisateurs recueillies, issues de $Daft_{web}$

3.2.2 Discussion

3.2.2.1 Agent *vs* Magicien d'Oz

Afin de recueillir des requêtes d'utilisateurs, nous avons opté pour une approche consistant à faire interagir les utilisateurs avec une première version de l'agent plutôt qu'une approche plus classique fondée sur une expérience en Magicien d'Oz. Cette approche était possible dans la mesure où nous avons fait l'hypothèse initiale que nous étions dans un système de type Interface en Langue Naturelle (NLI) (*cf.* section 3.1.3.4). En effet, contrairement à une session de dialogue, même si le système échoue pour répondre à la requête N, cela impactera peu sur le traitement de la requête N+1 par l'agent.
De plus, comme le souligne Salmon-Alt [2001, p.19], l'usage d'une approche de type Magicien d'Oz pour recueillir un corpus n'est pas sans inconvénients. Ainsi, l'expérimentateur jouant le rôle du magicien apparaît souvent incapable de simuler les limites supposées de la machine (c'est-à-dire, pour nous, d'un agent artificiel), surtout s'il s'agit d'une interaction de longue durée ou lorsque l'utilisateur sort des limites imaginées pour le scénario et se met à parler d'un domaine pour lequel les réponses n'ont pas été envisagées au préalable [Morel, 1989; Vivier & Nicolle, 1997].

3.2.2.2 Environnement contrôlé *vs* environnement ouvert

Pour conduire ce type d'expérience d'interaction entre un utilisateur et un système informatique, l'expérimentateur peut fixer de manière très stricte les conditions d'interactions (lieu, éclairage, présence au côté du sujet tout au long de l'expérience, *etc.*) ou au contraire lui laisser une certaine liberté. Sur ce point, notre approche a quelque peu évolué entre les deux

campagnes de recueil de requêtes.

Première campagne : environnement contrôlé. Lors de la première campagne de recueil, l'expérimentateur était physiquement présent aux côtés de l'utilisateur intéragissant avec le système et avec l'agent. Cette présence se justifie lorsqu'il est crucial que l'utilisateur puisse avoir une réponse à sa question pour lui permettre de progresser dans la tâche, endossant totalement le rôle de "l'ami derrière l'épaule" [Capobianco & Carbonell, 2001] si l'agent échoue. Cette présence peut aussi être vue comme une alternative simple à l'approche par Magicien d'Oz, et est suffisante lorsqu'on s'intéresse plus à la nature des questions posées (sémantique et pragmatique) qu'aux mots choisis pour les formuler (lexicologie), comme nous avons pu le faire dans d'autres contextes (*cf.* [Miletto *et al.* , 2009] et la section 8.1.3).

Seconde campagne : environnement semi-ouvert. Toutefois, s'il s'agit de recueillir des exemples de l'ensemble des types de requêtes que peut recevoir l'agent, nous pensons que l'approche suivie lors de la deuxième campagne, où les utilisateurs interagissaient à distance avec l'agent, sans contrainte induite de par la présence de l'expérimentateur, est préférable car plus représentative du comportement réel des utilisateurs finaux d'un ACA. Ceux-ci conservent en effet la liberté d'interagir avec l'agent :

- quand ils le souhaitent : ils ne se sentent pas "forcés" de poser une question pour satisfaire l'expérimentateur,
- de la manière dont ils le souhaitent : nous verrons par la suite que dans de nombreux cas, les utilisateurs ne se limitent pas dans leurs interactions au contexte de la tâche à accomplir, et ce genre d'écarts est beaucoup moins susceptible de se manifester lorsqu'un expérimentateur est physiquement présent à proximité du sujet.

Cette différence justifie donc à elle seule le besoin du corpus $Daft_{web}$, pour compléter le précédent. Notons cependant qu'au cours de cette campagne, l'agent était déployé sur un site avec un public relativement restreint de personnes au courant de l'objectif de l'expérience. Bien que les données recueillies ne contiennent pas d'informations permettant de retrouver leur auteur, il s'agissait donc plutôt d'un environnement que l'on pourrait qualifier grossièrement de "semi-ouvert". En effet, dans un environnement réellement ouvert, n'importe quel internaute serait potentiellement susceptible de se retrouver en contact avec l'agent et de lui parler : il aurait alors uniquement des intentions communicatives et non d'accomplissement d'une tâche. Souhaitant collecter un corpus demeurant axé sur un domaine particulier (l'assistance), et non sur l'ensemble des requêtes possibles que peut recueillir un agent de type chatbot[5], l'environnement semi-ouvert nous semble donc constituer un bon compromis.

[5]Nous reviendrons sur ce concept de chatbot dans la section 8.1.1.

3.2.2.3 Application unique *vs* applications multiples

Un des objectifs essentiels du projet DAFT consiste à offrir aux développeurs un agent assistant le plus générique possible de manière à ce qu'il soit en mesure de s'interfacer (pour un coût d'adaptation faible) à des applications diverses. Par conséquent, il est crucial que les structures de requêtes observées soient elles-mêmes le plus génériques possibles, c'est-à-dire capables de modéliser la sémantique de requêtes issues d'applications différentes. Pour cette raison, et contrairement à des corpus comme GOCAD, nous avons fait le choix de *recueillir*, de *mettre à plat* (*cf.* la section 3.1.3.4 discutant de la différence avec du dialogue à proprement parler) puis de *mélanger* les requêtes des utilisateurs interagissant avec différentes applications. En contrepartie, ce choix a pour effet d'augmenter légèrement la part de vocabulaire spécifique à l'application (par exemple "vitesse" ou "compter" pour le compteur Coco, "disque" ou "déplacer" pour le jeu des tours de Hanoï, "groupe AMI" ou "ajouter" pour le site Web AMI...). Les applications ne présentant pas la même complexité, elles n'ont pas permis de recueillir un nombre rigoureusement identique de requêtes, mais nous nous sommes assurés d'en avoir environ un millier pour chacune. Cette différence entre les applications peut donc éventuellement entraîner un léger biais dans la fréquence d'apparition de certains mots.

3.2.2.4 Corpus unique *vs* corpus multiples

Parmi les arguments justifiant notre choix de constituer un corpus dédié (*cf.* section 3.1.3.3), nous avons avancé que lorsqu'un corpus est formé de requêtes échangées entre un système et un utilisateur de celui-ci, les limites particulières de ce système influent dans une certaine mesure sur ce que va dire l'utilisateur. Cela signifie aussi que même dans le cadre de notre propre système, l'utilisation de ce corpus comme base de développement est conditionné par l'absence d'évolution majeure des performances du système. Cette considération plaide donc en faveur d'une approche consistant à récolter plusieurs corpus successifs au cours du développement, à la manière du cycle de développement en spirale [Boehm, 1988], la version N+1 du système étant développée à partir du corpus recueilli avec la version N.
Ainsi, la chaîne de traitement `DAFT 1.0` utilisée pour le recueil de ce corpus offrant des performances moyennes, il n'est pas interdit d'imaginer que confrontés à un système plus efficace (le système `DAFT 2.0`, développé à partir de ce même corpus), des utilisateurs auraient pu avoir moins tendance à abandonner leur tâche initiale en se mettant à discuter avec l'agent.
En outre, cette approche semble d'autant plus importante qu'on avance dans la chaîne de traitement de `DAFT`, car si une idée générale du vocabulaire utilisé par les utilisateurs suffit aux outils de `GRASP`, l'agent rationnel $\mathtt{A}_R$ est plus sensible à une évolution du contenu sémantique des requêtes en entrée. En effet, une amélioration du fonctionnement des premières étapes du système peut améliorer les performances de l'agent, et devenant plus compétent aux yeux de l'utilisateur, ce dernier lui posera des questions plus complexes qui ne seraient pas apparues antérieurement.

Dans le cadre de cette thèse, nous avons choisi d'ignorer cet impact potentiel et de considérer le corpus *Daft* recueilli comme demeurant représentatif des requêtes des utilisateurs en entrée, mais ce phénomène serait très certainement à prendre en compte dans le cadre du développement de la prochaine génération du système `DAFT`.

3.3 Enrichissement du corpus

Avec ses quelques 4 500 requêtes d'utilisateurs, $Daft_r$ est comparable en taille avec les corpus les plus proches, présentés dans le tableau 3.2. Cependant, dans notre approche, le corpus *Daft* se doit de couvrir la variété des requêtes en entrée le plus exhaustivement possible, de manière à circonscrire le registre linguistique de l'assistance. Et par rapport à cette ambition, 4 500 requêtes apparaît comme un nombre relativement modeste.
Ainsi, si de manière générale les deux caractéristiques essentielles d'un corpus sont son équilibre et sa représentativité des phénomènes linguistiques du domaine qu'il vise à étudier, nous nous proposons de renforcer cette deuxième caractéristique au détriment de la première.

3.3.1 Thésaurus

La solution la plus évidente pour améliorer la représentativité d'un corpus est d'augmenter sa taille, puisque plus le corpus est grand, plus il est probable qu'un phénomène linguistique donné, même peu fréquent, y figure. Mais bien que nous ayons dans un second temps rendu les agents `DAFT 1.0` plus facilement déployables sur le Web afin d'augmenter le nombre de sujets succeptibles d'interagir avec eux (*cf.* section 3.2.1.2), il demeure difficile d'en obtenir suffisamment pour augmenter significativement la taille du corpus d'un ou deux ordres de grandeur. Pour pallier ce problème, nous avons choisi d'enrichir le corpus de phrases semi-construites en procédant à une combinaison des ressources linguistiques issues de thésaurus avec le vocabulaire provenant de phrases réellement recueillies [Molinsky & Bliss, 1994; Atkins & Lewis, 1996].

Molinsky & Bliss [1994] présentent différentes formulations possibles en anglais pour 57 fonctions et stratégies de communication courantes. À partir d'un échantillon de phrases issues de $Daft_r$, il est possible de déterminer à quelle(s) fonction(s) chacune d'elles peut être rattachée, et d'utiliser une formulation alternative pour générer une phrase semi-construite à ajouter au corpus. Quelques fonctions utilisées sont présentées dans la table 3.6 avec pour chacune un exemple de phrase issue de $Daft_r$, la structure correspondante proposée par Molinsky & Bliss [1994] et une phrase construite à partir d'une formulation alternative.

Quelques 1 500 phrases supplémentaires ont ainsi pu être générées, formant le corpus $Daft_{thé}$ (*cf.* tableau 3.7 pour davantage d'exemples).

Fonction	Exemple issu de $Daft_r$ *Structure correspondante*	*Structure alternative* Alternative construite
Capacité / Incapacité	Peux-tu déplacer le disque ? ↪ *Can you _____ ?*	*Do you know how to _____ ?* ↪ Sais-tu comment déplacer le disque ?
Demande d'informations	Sais tu les règles du jeu ? ↪ *Do you/Would you know _____ ?*	*Could you (please) tell me _____ ?* ↪ Pourrais-tu me dire les règles du jeu ?
Introduction	moi je m'appelle Jean-Paul ↪ *My name is _____*	*I'm _____* ↪ Je suis Jean-Paul
Permission	est-ce possible de modifier la table ? ↪ *Is it possible to _____ ?*	*Is one allowed to _____* ↪ Peut-on modifier la table ?
Regrets	Dommage qu y ai pas de page demos ↪ *It's a pity that _____*	*I'm disappointed that _____* ↪ Je suis déçu qu'il n'y ait pas de page démos
Requêtes	Pourrais tu trier cette liste ? ↪ *Could you (please) _____ ?*	*I'd like you to _____* ↪ J'aimerais que tu tries cette liste.

Tableau 3.6 Exemples de structures fonctionnelles et de leur utilisation pour la construction de requêtes

N°	Requête construite
1	a t-on le droit de choisir l'assistant qu'on veut ?
2	moi je pense que cette page est bien plus moche que celle d'avant
3	il n'ya aucune raison pour que le compteur reparte pas à 0
4	ça serait beaucoup mieux si on se disait tu
5	selon toi, qui est à l'origine de la conception du site ?
6	comment faire pour manipuler les disques ?
7	je vois pas du tout pourquoi ça marche pas
8	qu'est-ce qu'on peut envisager de faire comme action ?
9	ça me chagrine de ne pas pouvoir modifier cette table
10	est ce que je pourrai avoir de l'info sur le site
11	Pourquoi est-ce que les boutons du jeu sont en Anglais ?
12	je jure que je n'ai pas voulu détruire la ligne
13	dis moi ce que tu sais au sujet de la notion de composant
14	c'est sûr qu'il manque Ripoche dans la liste des membres
15	quelle est la cause de l'arret

Tableau 3.7 Exemples de requêtes construites issues de $Daft_{thé}$

3.3.2 FAQ de Word et LaTeX

Bien qu'un site Web puisse avoir une structure complexe, les exemples d'applications traitées dans la section 3.2 demeurent relativement simples en comparaison de systèmes plus complexes tels qu'un logiciel de traitement de texte comme Word, où dans les dernières versions (avant l'adoption de l'interface à ruban) Beaudoin-Lafon [1997] a pu dénombrer pas moins de 150 actions basiques accessibles dans les menus, 60 boîtes de dialogues et 80 outils accessibles au travers d'icônes (*cf.* figure 3.4 pour un exemple caractéristique). De plus, même si l'on souhaite se cantonner à des agents déployés sur le Web uniquement, depuis 2005 et l'introduction du concept d'Asynchronous JavaScript and XML (AJAX)[Garrett, 2005], ont émergé des "web applications" suffisamment rapides et performantes pour être en mesure de remplacer des logiciels dédiés. Ainsi, des applications comme Zoho[6] ou Google Apps[7] sont devenues des alternatives crédibles à OpenOffice.org ou Microsoft Office [Beer, 2007].

Figure 3.4 Barres de menus de Word 2003

Nous avons donc sélectionné environ 5 000 phrases issues de Foire Aux Questions (FAQ) (*cf.* exemples dans le tableau 3.8) trouvées sur Internet concernant deux systèmes de composition de documents largement répandus (LaTeX et Word) pour former le corpus $Daft_{faq}$. En les réunissant aux requêtes construites précédemment, on obtient le corpus enrichi $Daft_e$.

N°	Requête d'utilisateur
1	Pour supprimer les polices que je n'utilise pas je peux utiliser le menu Polices ?
2	Comment ajouter et/ou supprimer des pointes de flèches
3	Avec Word, est-il possible de saisir du texte avec un micro ?
4	Peut-on savoir comment on peut définir les hauts et bas de page ?
5	Que puis-je lire sur TeX ?
6	Les caractères spéciaux ne m'affichent que de gros carrés.
7	Ce problème sera t-il a priori corrigé par la mise à jour d'Acrobat ?
8	Toutes les 2 minutes, je suis obligé d'arrêter Word et de le relancer !
9	Où puis-je trouver un vérificateur de syntaxe LaTeX ?
10	Suite à la mise à jour de Word, impossible de récupérer les mots ajoutés dans le dictionnaire
11	Avant la version actuelle, il n'existe plus d'option qui permette de conserver les lignes vides
12	Lorsque j'imprime tout est ok sauf qu'il me manque les accents.
13	Un exemple ".tex" déjà tout fait ne m'intéresse pas trop
14	Je ne sais pas quoi faire pour afficher des notes de bas de page dans un document Word
15	Y a t-il d'autres possibilités que d'appuyer sur "F8" pour imprimer ?

Tableau 3.8 Exemples de requêtes issues de $Daft_{faq}$

[6] http://www.zoho.com
[7] http://docs.google.com

3.3.3 Discussion

3.3.3.1 Composition du corpus *Daft*

Le corpus *Daft* final est donc né de l'union de quatre sous-corpus constitués dans des circonstances différentes :

$$\begin{aligned} Daft &= (Daft_{app} \cup Daft_{web}) \cup (Daft_{thé} \cup Daft_{faq}) \\ &= Daft_r \cup Daft_e \end{aligned}$$

Néanmoins, l'apport de $Daft_e$, tel qu'il a été réalisé, et malgré les précautions prises, dénature fatalement l'homogénéité relative du corpus $Daft_r$. Ceci n'est toutefois réellement problématique que lorsque l'on souhaite disposer d'informations précises quant à la fréquence d'un phénomène linguistique, les moins fréquents se trouvant surreprésentés par leur ajout manuel au corpus. Par conséquent, dans la suite nous allons travailler sur des sous-corpus différents en fonction des besoins à satisfaire : lorsque nous aurons besoin d'étudier des fréquences, nous nous baserons donc sur le corpus entièrement constitué des 4 500 requêtes recueillies $Daft_r$, tandis que lorsque nous souhaiterons étudier de manière exhaustive l'ensemble des phénomènes linguistiques potentiellement présents, nous nous référerons aux 11 626 requêtes du corpus *Daft* dans son ensemble.

La répartition des requêtes au sein des différents sous-corpus de *Daft* est rappelée par la figure 3.5. Pour un aperçu plus conséquent des requêtes du corpus *Daft* dans son intégralité, on pourra se reporter à l'annexe A.

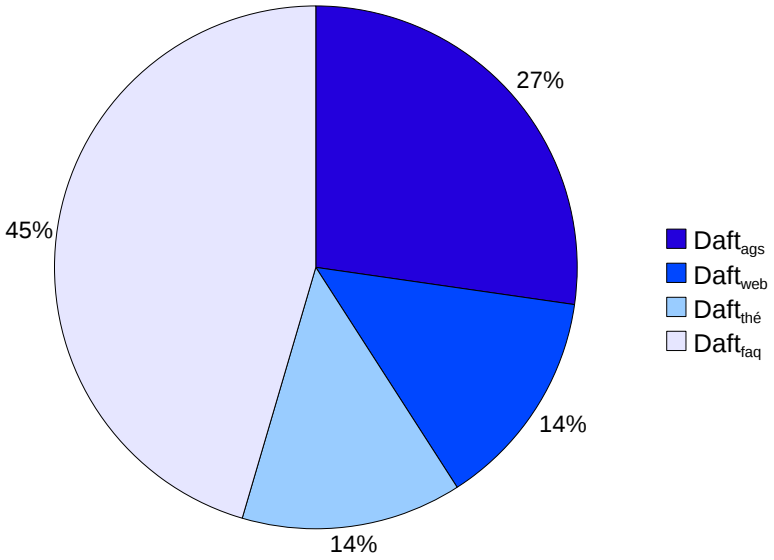


Figure 3.5 Origine des différentes requêtes constituant le corpus *Daft*

3.3.3.2 Définition du sous-corpus d'étude $Daft_{sub}$

À plusieurs reprises dans la suite de cette thèse, nous aurons besoin de mener une étude particulière basée sur le corpus *Daft*, sans pour autant vouloir (ou pouvoir – le processus d'annotation pouvant être long selon les phénomènes que l'on cherche à identifier) traiter l'ensemble de celui-ci. Nous aurons alors souvent recours à deux sous-ensembles distincts du corpus $Daft_r$ représentant chacun environ un dixième du corpus $Daft_r$ total, composés de requêtes sélectionnées aléatoirement, en veillant simplement à ce qu'aucune phrase ne soit commune aux deux sous-ensembles.
Nous nommerons ces deux sous-ensembles $Daft_{sub_1}$ et $Daft_{sub_2}$, et travaillerons parfois sur l'union des deux, $Daft_{sub}$.
En résumé, nous avons donc :

$$\begin{cases} |Daft_{sub_1}| &= 522 &\approx \frac{1}{10}|Daft_r| \\ |Daft_{sub_2}| &= 552 &\approx \frac{1}{10}|Daft_r| \\ Daft_{sub_1} \cap Daft_{sub_2} &= \emptyset & \\ Daft_{sub} &= Daft_{sub_1} \cup Daft_{sub_2} & \\ |Daft_{sub}| &= 1\,074 &\approx \frac{1}{10}|Daft| \end{cases}$$

3.4 Conclusion

Nous avons montré dans ce chapitre l'intérêt fondamental de l'approche corpus dans la réalisation de systèmes d'interactions homme-machine à base de langue naturelle, et les raisons pour lesquelles le recueil d'un corpus spécifique à notre projet était indispensable. L'approche mixte utilisée pour la constitution de celui-ci (d'une part le recueil expérimental, d'autre part l'utilisation de thésaurus) nous garantit une couverture relativement correcte du domaine de l'assistance, tandis que l'utilisation d'applications de différentes natures assure que les structures linguistiques utilisées sont relativement génériques et indépendantes de l'élément assisté, ce qui nous permet d'espérer une certaine robustesse.

Chapitre 4

Analyseur syntaxico-sémantique : GRASP

Sommaire

Dans ce chapitre, nous allons détailler le fonctionnement du module `GRASP`, qui au sein de `DAFT` est en charge de l'analyse linguistique des requêtes d'utilisateurs. Dans un premier temps, nous analyserons les raisons qui nous poussent à développer un analyseur syntaxico-sémantique dédié à partir du corpus *Daft*, puis nous décrirons le fonctionnement technique des différentes étapes de traitement de la requête : la lemmatisation, l'étiquetage sémantique et l'analyse grammaticale en constituants.

4.1 Motivation

L'outil d'analyse grammatical `GRASP` est relativement traditionnel dans sa conception, se basant pour ses différents étages sur l'état de l'art (*cf.* section 1.2.2). On peut donc s'interroger sur la nécessité de développer un module spécifique à cet effet plutôt que d'avoir recours à certains des analyseurs librement disponibles existants.

4.1.1 Besoin de précision

Ce choix se justifie par le besoin de précision dans la modélisation des requêtes, car une analyse erronée d'une requête est susceptible de produire des faux-sens voire des contre-sens dans la réponse apportée par l'agent à l'utilisateur. Or une réponse non pertinente par rapport à la question[1] est extrêmement préjudiciable pour la crédibilité de l'agent, d'autant plus lorsque celui-ci est doté d'une personnification : c'est ce que Galluccio [2006] a nommé le "Clippy Effect", phénomène décrit par Xiao *et al.* [2004].
De plus, la plupart des analyseurs grammaticaux utilisés par exemple en recherche d'information sont efficaces pour la recherche d'entités nommées et de syntagmes nominaux, mais moins lorsqu'il s'agit d'analyser les relations des différents constituants par rapport aux syntagmes verbaux. Or les verbes ont un rôle essentiel dans l'analyse en termes d'actes de langage que nous souhaitons réaliser au final.

4.1.2 Possibilité de personnalisation

Un analyseur peut être plus précis s'il a été construit pour travailler sur un registre de langue fini qui constitue naturellement un langage contrôlé ou un **sous-langage** [Grishman & Kittredge, 1986]. Kittredge [2003] définit la notion de sous-langage naturel comme nécessitant :

1. une communauté de locuteurs partageant le même savoir spécialisé sur un domaine sémantique restreint ;
2. que les locuteurs communiquent dans des situations hautement similaires.

Dans le cas présent, la similarité des situations des utilisateurs en besoin d'assistance est claire. Le premier point est en revanche plus ambigu, puisque Kittredge cite comme exemple typique de communauté les experts d'un domaine particulier (par exemple, les météorologues), et que les utilisateurs novices sont (par définition) tout sauf des experts. Néanmoins, c'est davantage l'idée d'un domaine sémantique distinct de la langue naturelle prise dans son ensemble qui prime ici, et de ce point de vue, le domaine restreint par le contexte des applications assistées permet cela (ce qui est confirmé par l'étude présentée dans la section 7.1.1 où l'on compare le corpus *Daft* à un corpus généraliste).

[1] Notons qu'il y a une différence entre ce qu'on appelle ici une réponse *non pertinente*, et une réponse *non utile* mais néanmoins pertinente que serait une phrase du type "J'ai bien compris que vous me posiez une question sur X, mais je n'ai pas d'information à ce sujet".

De plus, toujours d'après Kittredge [2003], si par ailleurs les phrases de ces locuteurs suivent des schémas particuliers les distinguant du langage pris dans son ensemble (classes de mots particulières, ensemble circonscrit d'expressions et phrases elles-mêmes plus réduites), on a alors affaire à un sous-langage.
Or lors de l'analyse des ressources linguistiques recueillies, nous avons en particulier montré par la comparaison à un corpus généraliste (*cf.* section 7.1.1) et par l'annotation manuelle des phrases faite pour définir la nouvelle version du langage *DAFT* (*cf.* section 5.3) que si l'on considère uniquement les requêtes relevant de l'assistance, elles semblent bien présenter des caractéristiques propres aux sous-langages.
Par conséquent, une approche visant à développer des outils de traitements spécifiques à notre cadre d'étude est tout à fait envisageable.

4.1.3 Approche d'analyse choisie

N°	Requêtes issues de *Daft*
1	**Cliques** sur le bouton quitter
2	**cliquesur** le bouton précédent
3	ok, reviens à **l apage** d'accueil
4	**a** quoi sert cette fenêtre**,**
5	**c koi** le GT ACA
6	est-ce que le bouton "fermer" et le bouton "quitter" **marches** exactement pareil ?
7	Je ne **trovue** pas la page de **demso** ! !
8	Je suis vraiment surprise qu'il n'y ait pas de fonction d'annulation **global**
9	**Ca** serait mieux de pouvoir retourner directement au début
10	**auf viedersen**
11	bon à rien !
12	ça marche pour moi **:-)**

Tableau 4.1 Exemples de requêtes de *Daft* présentant différents type d'erreurs nécessitant un traitement particulier

Nous avons vu dans la section 1.2.2.2 que deux approches étaient possibles pour l'analyse syntaxique : une analyse montante (bottom-up) ou descendante (top-down). Ici, l'étude de quelques phrases issues du corpus (*cf.* tableau 4.1) permet de voir que les requêtes reçues par le système sont fortement bruitées (expressions orales, fautes de frappe, acronymes issus du langage SMS, *etc.*) et bien souvent agrammaticales, pour près de la moitié d'entre elles. Ceci exclut donc le recours à des méthodes d'analyse descendante. À défaut de permettre une analyse grammaticale complète de la phrase, les approches bottom-up permettent en effet de définir des îlots de compréhension pouvant être utiles pour faciliter l'analyse sémantique par la suite. Toutefois, le système doit être suffisamment robuste pour traiter également des phrases qui n'ont pas pu être correctement analysées d'un point de vue grammatical : l'analyse syntaxique est donc considérée dans le cas de `GRASP` comme optionnelle, et ne sera effectuée qu'en dernier.

4.2 Lemmatisation

4.2.1 Lexique

Comme détaillé dans la section 1.2.2.1, un lexique est une ressource indispensable à la lemmatisation. Nous avons donc constitué le lexique utilisé par `GRASP`, que nous appellerons $\mathbb{L}_G$, qui couvre de manière exhaustive tous les lemmes et locutions apparaissant dans le corpus *Daft*, et même un peu plus : il y a en effet environ 10% de lemmes supplémentaires, qui ont été ajoutés lorsqu'ils semblaient suffisamment proches de lemmes déjà présents pour avoir une forte probabilité d'apparaître dans des requêtes ultérieures.

4.2.1.1 Entrées du lexique

Une entrée du lexique est de la forme suivante :

{

LEM[*id*] | **LOC**[*id*],	$id \in String$ est l'identifiant unique du lemme ou de la locution
POS[*pos*],	$pos \in \mathbb{P}_G$ est la nature de l'entrée, sous la forme d'un tag de deux lettres, inspirée du Penn Treebank [Marcus *et al.* , 1994]
FLEX[*flex*],	$flex \in \mathbb{F}_G$ est la liste de flexions associées au lemme
KEYS[*keys*],	$keys \in (\mathbb{K}_G)^*$ est la liste des clés sémantiques associées
GEND[*gender*],	*gender* correspond au genre (M ou F) d'un nom
DICO[*dico*]	$dico \in \mathbb{D}_G$ est le sous-ensemble de vocabulaire de l'entrée

}

Les champs en gras sont présents dans chaque entrée, les autres sont optionnels. $\mathbb{P}_G$, $\mathbb{F}_G$, $\mathbb{K}_G$ et $\mathbb{D}_G$ correspondent respectivement à l'ensemble des natures, des **flexions**, des **clés sémantiques** et des dictionnaires utilisés par `GRASP`, et ils sont définis dans les paragraphes qui suivent. D'autres exemples d'entrées du lexique sont donnés en annexe C.

Nature (POS). À chaque entrée de $\mathbb{L}_G$ (*i.e.* chaque lemme ou locution) est associée une nature unique : si pour un même lemme il existe plusieurs natures différentes, on définira autant d'entrées différentes que nécessaire (ex : "rougeN" pour le nom, "rougeJ" pour l'adjectif). La liste des 10 tags contenus dans $\mathbb{P}_G$ est donnée dans le tableau 4.2.

Flexions (FLEX). La liste de flexions associées au lemme d'une entrée dépend de la nature de celui-ci :

- formes singulière (FS) et plurielle (FP) pour un nom (NN),
  e.g. `FLEX[FS["bouton"], FP["boutons"]]`

Tag	Nature	Exemples
NN	nom	accueil, bouton, chargement
JJ	adjectif	petit, rouge, bleu
VV	verbe	cliquer, modifier
RR	adverb	absolument, seulement
DD	déterminant	chaque, le, une, au moins
GG	grammatical	actuellement, quoi, rien
CD	cardinal	cinq, mille, quarante-deux, 2, 134
PP	pronom	je, ceci, ça
KK	ponctuation	$pvirg$ (,), $quote$ (“)
BB	limite de phrase	.!? ...

Tableau 4.2 Liste des tags de POS utilisés par `GRASP` avec quelques exemples

- formes variant en genre et en nombre (MS/MP/FS/FP) dans le cas d'un adjectif (JJ),
  e.g. `FLEX[MS["petit"], MP["petits"], FS["petite"], FP["petites"]]`
- table de conjugaison associée (CONJ) issue de Hatier [2006] pour un verbe (VV),
  e.g. `FLEX[CONJ["cliquer",7]]`
- flexion unique dans les autres cas.
  e.g. `FLEX["maintenant"]`

Pour un lemme donné, toutes les flexions existantes ont été prises en compte, même lorsqu'elles n'étaient pas présentes dans le corpus, par exemple les différentes formes issues de la conjugaison d'un verbe (par ajout des terminaisons de la table de conjugaison à la racine).

Clés (KEYS). Les clés sémantiques représentent des concepts cognitifs, à la manière des synsets[2] de WordNet. Elles seront définies de manière plus complète dans la section 4.3.1 consacrée à l'analyse sémantique. Contrairement au POS, à une entrée du lexique peuvent être associées plusieurs clés sémantiques (*cf.* figure 4.1).

```
{                                   {
  LEM[‘surG’],                        LEM[‘surJ’],
  POS[GG],                            POS[JJ],
  FLEX[‘sur’],                        FLEX[{MS[‘sûr’], MP[‘sûrs’], FS[‘sûre’], FP[‘sûres’]}],
  KEYS[{‘ABOUT’, ‘ISPOSUPON’}]        KEYS[{‘ISCERTAIN’, ‘ISSECURE’}
}                                   }
```

Figure 4.1 Exemple de clés multiples associées à deux lemmes (‘surG’ et ‘surJ’), eux-mêmes associés à une même flexion (‘sur’), les accents étant éliminés lors du prétraitement (*cf.* algorithme 1

Dictionnaire (DICO). Les dictionnaires permettent d'associer à certains lemmes l'application utilisée dans l'exemple où la requête dont ils sont issus a été recueillie. Cela permet de séparer le vocabulaire spécifique à l'application (qui pourrait n'être chargé qu'au cas par cas) du vocabulaire plus générique relevant à proprement parler de l'assistance. Les dictionnaires

[2] Ou “ensemble de synonymes”, est un ensemble de mots considérés comme étant sémantiquement équivalents.

sont liés aux différentes phases de recueil décrites dans le chapitre 3. De plus, nous sommes également servi de ce champ pour distinguer les locutions et les mots d'origine étrangère.
On a donc $\mathbb{D}_G$ = {LEXIC, LOCUTION, FOREIGN, ENGLISH, WORDTEX, WEBSITE, HANOI, GTACA, COCO}.

4.2.1.2 Méthodologie et quantification

Les 11 626 requêtes de *Daft* sont composées au total de 105 396 mots. En faisant l'union de cet ensemble, on obtient 6 325 flexions différentes (uniques) que nous avons regroupées manuellement en les rattachant à 3 736 lemmes. Cette méthode ne permettant pas d'extraire automatiquement les locutions (ensemble de mots), celles-ci ont été ajoutées par la suite manuellement, ainsi qu'environ 400 lemmes et locutions supplémentaires non présents dans le corpus mais proches d'éléments existants (*e.g.* des éléments liés aux nombres (sixième, onze, *etc.*) ou aux couleurs, puisque le corpus en contenait déjà un certain nombre). Au total, nous avons ainsi obtenu 4 266 entrées dans le lexique $\mathbb{L}_G$.

Parmi ces entrées, 395 sont des locutions et 3 645 (soit 85%) appartiennent au dictionnaire principal LEXIC. La répartition en fonction de leur nature (POS) est donnée par la figure 4.2.

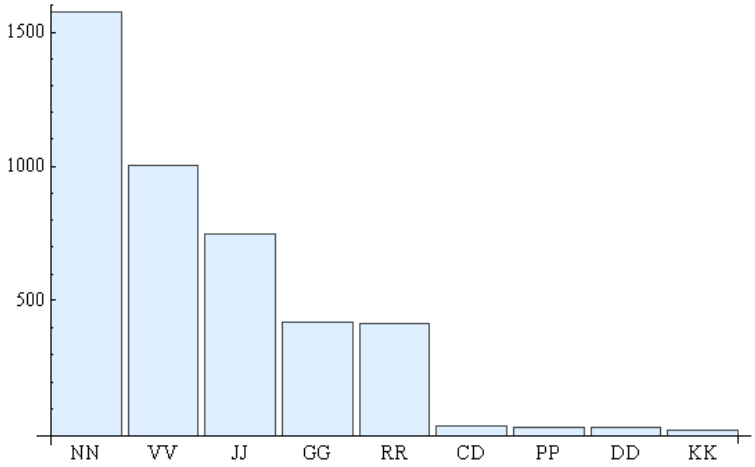


Figure 4.2 Répartition des 4 266 entrées de $\mathbb{L}_G$ en fonction de leur POS

4.2.2 Lemmatiseur

À l'aide du lexique $\mathbb{L}_G$ précédemment constitué, il est possible de procéder à la lemmatisation des requêtes utilisateurs en entrée. Le but du lemmatiseur est de produire une liste d'entités formelles que nous appellerons des nœuds, contenant des informations liées aux mots et locutions constituant la phrase. Un exemple de sortie est donné sur la figure 4.3.

Le principe de l'algorithme de lemmatisation utilisé est donné ci-dessous (*cf.* algorithme 1). Les principales fonctions annexes qui y sont appelées sont décrites dans les sous-sections suivantes.

Algorithme 1: Lemmatisation

Input : Requête R_0 en langue naturelle (chaîne de caractères de w mots)
Output : Ensemble de n noeuds représentant la requête lemmatisée R_f

```
R1 ← preprocess(R0);
R2 ← {};
foreach mot m de R1 do
    if m ≠ EXPRESSION then                    // Passage en minuscules sans accent
        m ← lowerCase(deleteAccents(m));
    end
    R2 ← R2 ∪ {m};
end
R3 ← {};
F ← formSegmentation(R2) ;  // Ensemble de flexions/expressions/locutions
foreach élément e de F do
    switch type de e do
        case flexion
            L ← lemmes associés à e dans L_G ;
            switch |L| do
                case 0  n ← handlingUnknownFlexion(e) ;   // Pas de lemme connu
                case 1  n ← createNode(L[0]) ;                // Pas d'ambigüité
                otherwise                          // Plusieurs lemmes possibles
                    A ← {};
                    foreach lemme l de L do
                        A ← A ∪ createNode(l);
                    end
                    // Sélectionne le nœud le plus probable de A
                    n ← flexionToNodeDisambiguation(A, R3);
                    // Stocke tous les nœuds possibles dans n
                    addFieldToNode(n, ALIST, A);
                endsw
            endsw
        endsw
        case locution  n ← createNode(e);
        case expression // Crée un nœud avec les infos propres à l'expression
            l ← getWrapperContent(e);
            i ← getWrapperNodeInfo(e);
            n ← createNode(l,i);
        endsw
    endsw
    R3 ← R3 ∪ {n};
end
Rf ← R3 ∪ { makeLastNode(R3) } ;          // Ajoute un nœud marquant la fin
Rf ← { makeFirstNode(Rf) } ∪ Rf ;        // Ajoute un nœud spécial au début
return Rf;
```

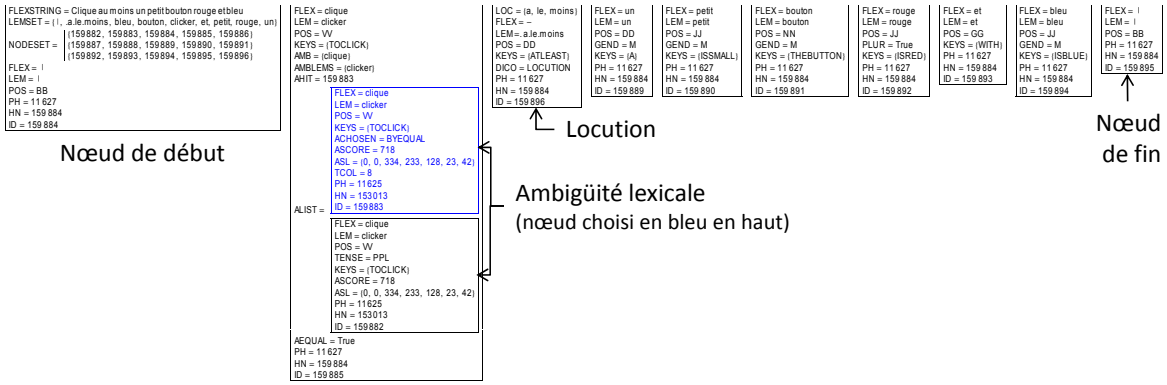


Figure 4.3 Exemple de lemmatisation par GRASP de la requête “Clique au moins un petit bouton rouge et bleu”. Chaque mot est associé à un nœud. Un nœud de début et un nœud de fin encadrent la phrase et servent à la gestion et à la trace du processus d'analyse.

4.2.2.1 Structure d'un nœud

Un nœud est constitué de plusieurs champs de la forme attribut-valeur, selon la syntaxe $ATTRIBUT = v$, où v peut être :

- une valeur terminale : chaîne de caractères, nombre, booléen ou symbole,
- une liste de valeurs terminales,
- un autre nœud (d'où une structure en arbre).

Les principaux attributs de nœuds sont donnés dans la tableau 4.3. Les autres attributs n'ont pas de valeur linguistique et servent pour la gestion du processus (*e.g.* PH représente le numéro de phrase dans le corpus, ID l'identifiant unique du nœud, *etc.*).

Attribut de nœud	Type de valeur	Description
FLEX	String	Flexion issue de la requête d'utilisateur
LOC	Liste de Strings	Flexions issues de la requête d'utilisateur et formant une locution (mutuellement exclusif avec FLEX)
LEM	Symbole	Entrée de $\mathbb{L}_G$ associée à la flexion (LEM) ou la locution (LOC)
POS	Symbole	Nature associée au noeud (*cf.* tableau 4.2)
GEND	Symbole	Pour une flexion, M ou F selon le genre
PLUR	Booléen	Pour une flexion, *vrai* si le nombre est pluriel
TENSE	Symbole	Pour un verbe, code représentant le temps de conjugaison
KEYS	Liste de symboles	Clé(s) dans $\mathbb{K}_G$, associée(s) à l'entrée d'après $\mathbb{L}_G$

Tableau 4.3 Liste des principaux attributs de nœuds produits par le lemmatiseur de GRASP

4.2.2.2 Prétraitement

Motivation : Aussi complet soit-il, un lexique ne peut contenir tous les éléments susceptibles d'être fournis en entrée d'un système de TALN, et certaines catégories de mots doivent être

traitées à part, notamment tout ce qui concerne les entités nommées (*e.g.* noms propres, dates, quantités, *etc.*).

Solution : Le prétraitement de la phrase (fonction preprocess dans l'algorithme 1) permet de gérer les idiosyncrasies présentes dans la requête d'origine de l'utilisateur. Deux traitements sont possibles, et elles peuvent être selon les cas :

- transformées en une ou plusieurs flexions, après quoi elles seront traitées comme si l'utilisateur les avait saisies ainsi. C'est notamment le cas des élisions ("j' " devient "je"), des erreurs classiques ("ya" devient "il y a"), des raccourcis ("a+" devient "à plus tard"), des cardinaux ("1er" devient "premier"), *etc.*
- encapsulées dans un marqueur EXPRESSION, et être traitées spécifiquement par la suite. Ceci est utile pour tous les éléments n'apparaissant pas directement dans $\mathbb{L}_G$, par exemple les entités nommées pour lesquelles un nœud spécifique ayant pour POS "NE" (named entity) sera créé, les nombres (*e.g.* "42") pour lesquels on crée un nœud de POS "CD", les adresses web associées à un nœud de POS "WA" (web address), *etc.*

4.2.2.3 Représentation des ambigüités flexion-lemme

Motivation : Si à chaque entrée de $\mathbb{L}_G$ ne correspond qu'un lemme unique, la relation flexion-lemme n'est pas toujours unique. L'obtention de plus d'un lemme à partir d'une flexion donnée peut provenir de l'existence :

- de deux entrées différentes dans $\mathbb{L}_G$ associées à une même graphie en cas de nature différente, *e.g.* "la porte" (entrée "porteN") *vs* "je porte" (entrée "porteV").
- de plusieurs flexions ayant la même graphie au sein d'une même entrée de $\mathbb{L}_G$ (donc une même nature) mais des attributs grammaticaux différents :
  - en genre : "un bouton rouge" (MS) *vs* "une fenêtre rouge" (FS),
  - en nombre : "un bouton gris" (MS) *vs* "des boutons gris" (MP),
  - en temps : "clique le bouton" (présent) *vs* "le bouton clique" (participe passé dont l'accent a disparu lors du prétraitement).

De plus, il apparaît que la fréquence des ambigüités est loin d'être négligeable (*cf.* tableau 4.4). Elles représentent moins de 2% de toutes les flexions répertoriées dans les entrées de $\mathbb{L}_G$ (nombre nettement plus grand que le nombre de flexions apparaissant en pratique dans le corpus), mais elles correspondent en pratique à près de 10% des nœuds construits à partir des requêtes du corpus, soit près d'un nœud par phrase en moyenne (0,8).

Solution : Un choix doit donc être effectué (fonction flexionToNodeDisambiguation décrite en 4.2.2.5) pour permettre la suite des traitements de la phrase. Toutefois, des erreurs étant possibles, nous préférons conserver au niveau d'un nœud ambigu les différentes interprétations possibles pour une éventuelle correction ultérieure (par exemple en fonction du contexte). Pour

Ressource	Nombre d'éléments	Éléments ambigus		Ambigüité à 2 possibilités	
		Nombre	Proportion	Nombre	Proportion
$\mathbb{L}_G$	64 313 flexions	1 026	1,6%	988	96,3%
Daft	101 732 nœuds	9 087	8,9%	7 620	83,9%

Tableau 4.4 Fréquence des flexions ambigües au sein de $\mathbb{L}_G$ et *Daft*

cela, nous avons recours à l'ajout d'un attribut ALIST au sein du nœud construit à partir de l'interprétation choisie, qui conserve tous les nœuds possibles (y compris celui choisi), via la fonction addFieldToNode. C'est le cas par exemple du nœud construit à partir de la flexion "clique" sur la figure 4.3.
À noter que l'on stocke aussi dans cet attribut les différentes possibilités éventuellement engendrées par le correcteur orthographique (*cf.* section 4.2.2.4), ce qui peut faire monter le nombre de nœuds associés à cet attribut à plus de 8.

4.2.2.4 Correction orthographique

Motivation : Dans la mesure où nous traitons des requêtes saisies au clavier, l'utilisation d'un correcteur orthographique, au moins basique, est indispensable, au risque de ne pas pouvoir traiter le moindre mot mal orthographié. Une estimation sur le corpus *Daft* montre que près de 10% des requêtes présentent au moins une faute d'orthographe (on ne compte pas les problèmes d'accord grammaticaux) : il s'agit essentiellement de fautes de frappe entraînant l'inversion de deux lettres, le mauvais découpage d'un mot ou l'absence d'une lettre ou d'un accent.

Solution : Un correcteur orthographique est une simple fonction prenant un mot en paramètre et retournant soit le mot lui-même s'il est correctement orthographié, soit le mot correct le plus proche, ce qui suppose donc l'utilisation d'une métrique. Pour `GRASP`, nous avons eu recours, de manière classique, à une distance de Damerau-Levenshtein (D-L) qui prend en compte l'insertion, la suppression ou la substitution d'un caractère, ainsi que la tranposition de deux caractères – ce qui représente, d'après Damerau [1964], 80% des erreurs humaines. Cette distance est utilisée au sein d'une fonction (handlingUnknownFlexion) appelée uniquement lorsqu'aucun lemme ne peut être associé à une flexion donnée, et l'algorithme 2 décrit en détail son fonctionnement.

Évaluation : Parmi les 6 325 flexions différentes issues du corpus *Daft*, 532 ne peuvent pas se voir associer de lemme (soit 8,4%). Après introduction du correcteur implémenté dans `GRASP`, seules 61 flexions ne sont pas corrigées. Par définition, $\mathbb{L}_G$ ayant été défini à partir du corpus, ces flexions ont une entrée correcte associée existante, mais c'est alors que celle-ci se situe à

une distance D-L > 1. Cette contrainte d'une distance de 1 se justifie cependant, car elle évite de multiplier les nœuds ambigus.

Algorithme 2: Correction orthographique : handlingUnknownFlexion

Input : Un mot w n'ayant pas de lemme associé dans $\mathbb{L}_G$
Output : Un nœud n construit autour du/des lemmes le(s) plus proche(s) trouvé(s)

```
n ← {};
// Liste les lemmes de 𝕃_G avec une flexion à une distance D-L de 1 de w
L ← spellingCorrection(w,Damerau-Levenshtein,1);
switch |L| do
    case 1  n ← createNode(L[0]) ;        // crée un nœud à partir de l'entrée
    case > 1                              // crée un nœud avec un attribut ALIST
        A ← {};
        foreach lemme l de L do
            A ← A ∪ createNode(l);
        end
        n ← flexionToNodeDisambiguation(A);
        addFieldToNode(n, ALIST, A);
    endsw
    case 0                                // tente de découper w en mots de 𝕃_G
        G ← searchGluedWords(w);
        if |G| = 0 then  n ← ∅;
        else  foreach e de G do  n ← n ∪ createNode(e);
    endsw
endsw
return n;
```

4.2.2.5 Étiquetage morphosyntaxique et désambigüisation

Motivation : Dans GRASP, on a vu que la nature d'un lemme était liée à son entrée via l'attribut POS. Sur les 124 976 nœuds produits lors de l'analyse du corpus, 9 258 sont ambigus. Le principal problème à résoudre consiste donc à déterminer, en fonction du contexte, la bonne entrée (et donc, en particulier, la bonne nature) associée à ces nœuds.
On a vu en 1.2.2.1 l'importance de l'étiquetage morphosyntaxique pour la suite de l'analyse d'une requête en langue naturelle. Toutefois, au sein de GRASP, il était difficile d'avoir recours à un analyseur classique (comme TreeTagger [Schmid, 1994]) pour plusieurs raisons :

- Nous souhaitions disposer d'informations complémentaires à la nature (temps de conjugaison, genre) qui ne sont pas toujours fournies par les analyseurs librement disponibles (TreeTagger en français fournit par exemple le temps mais pas le genre) ;
- Comme expliqué précédemment, le fait de travailler sur un sous-langage nous permettait d'espérer raisonnablement d'obtenir de meilleurs résultats, notamment ici de par le nombre limité de flexions considérées (générant donc moins d'ambigüités) ;
- Le choix d'un nœud parmi les alternatives possibles (attribut ALIST) ne dépend pas seulement de leur nature ;

– D'un point de vue technique, l'intégration d'un outil déjà existant au sein de l'environnement *Mathematica* utilisé pour `GRASP` n'était pas forcément triviale[3].

Solution : Nous avons donc proposé, pour implémenter la fonction de **désambigüisation lexicale** flexionToNodeDisambiguation, de se baser sur la fréquence d'apparition de différents N-grammes au sein du corpus *Daft*, en ne se basant pas seulement sur le lemme (attribut LEM) associé au nœud, mais aussi sur sa nature (attribut POS) et la clé sémantique associée (attribut KEYS – *cf.* section 4.3.1). De plus, procédant de manière automatique, toutes les requêtes du corpus ont dans un premier temps été analysées par une version de `GRASP` sans désambigüisation, et nous avons supprimé de la liste des résultats les N-grammes contenant un lemme inconnu (0,6%) ou ambigu (7,4%).
À partir de là, nous avons extrait sept listes de N-grammes en fonction de leur fréquence (par ordre décroissant – *cf.* exemples dans le tableau 4.5) :

1. $\mathbb{G}_L$ La liste de fréquence d'apparition des 1-grammes de lemmes et locutions de $\mathbb{L}_G$,
2. $\mathbb{G}_{\neg GL}$ La liste de fréquence d'apparition des 1-grammes de lemmes non grammaticaux,
3. $\mathbb{G}_{LL}$ La liste de fréquence d'apparition des bigrammes LEM-LEM,
4. $\mathbb{G}_{LP}$ La liste de fréquence d'apparition des bigrammes LEM-POS,
5. $\mathbb{G}_{PP}$ La liste de fréquence d'apparition des bigrammes POS-POS,
6. $\mathbb{G}_{PPP}$ La liste de fréquence d'apparition des trigrammes POS-POS-POS,
7. $\mathbb{G}_{KK}$ La liste de fréquence d'apparition des bigrammes KEYS-KEYS.

Elles sont utilisées au sein de l'algorithme 3. À noter que pour prendre en compte le contexte on s'intéresse essentiellement aux nœuds antérieurs au nœud à désambigüiser. Cela signifie que lorsque plusieurs nœuds successifs sont ambigus (en pratique, on constate que ce nombre ne dépasse jamais quatre dans le corpus), ce qui arrive dans 7,6% des cas, on commence par désambigüiser celui le plus "à gauche" dans la requête (*i.e.* le premier dans la requête en langue naturelle).

Évaluation : Un peu moins de 10% des nœuds ambigus finissent par avoir un score égal. Dans ces cas là, la règle basée sur le choix du nœud dont la nature est la plus fréquente dans le corpus permet généralement de choisir la bonne entrée de $\mathbb{L}_G$. À ce niveau, parmi les 9 258 nœuds ambigus, 9 001 sont correctement traités. Pour les 2,8% nœuds restants, ils sont correctement traités grâce à l'utilisation d'une 'stoplist' gérant de manière individuelle 65 cas particuliers (*e.g.* "sous" suivi par un nœud de nom ou de verbe a pour valeur de POS NN).
À la sortie du lemmatiseur de `GRASP`, toutes les phrases du corpus *Daft* sont donc correctement annotées en termes de POS, ce qui permet d'évaluer le taux de succès des phases de traitement ultérieures sans que celui-ci soit impacté par le taux de succès du lemmatiseur.

[3]Cela a été fait à partir de la version originale de TreeTagger en anglais, mais pas en français.

Rang	N-grammes						
	$\mathbb{G}_L$	$\mathbb{G}_{\neg GL}$	$\mathbb{G}_{LL}$	$\mathbb{G}_{LP}$	$\mathbb{G}_{PP}$	$\mathbb{G}_{PPP}$	$\mathbb{G}_{KK}$
1	le	etre	$intg$, |	le, NN	DD, NN	GG, DD, NN	OF, THE
2	de	pouvoir	de, le	$intg$, BB	GG, DD	DD, NN, GG	$A, THE
3	$intg$	avoir	|, je	|, PP	PP, VV	VV, DD, NN	THEUSER, TOWANT
4	etre	faire	$excl$, |	|, GG	NN, GG	BB, BB, PP	THIS, TOBE
5	je	page	|, comment	de, DD	VV, GG	BB, BB, GG	LITERAL, QUEST
6	un	vouloir	|, le	|, VV	KK, BB	NN, GG, DD	THEUSER, NEG
7	a	dire	a, le	je, VV	GG, VV	BB, BB, VV	THE, THE-WEBPAGE
8	que	bouton	je, vouloir	un, NN	VV, DD	NN, KK, BB	THIS, $QUE
9	ce	savoir	|, pouvoirV	de, NN	BB, PP	BB, PP, VV	THEUSER, TOHAVE
10	il	compteur	ce, etre	|, DD	BB, GG	NN, BB, BB	TOBE, THE

Tableau 4.5 Les 10 N-grammes les plus fréquents dans chacune des sept listes de fréquences considérées pour la désambigüisation de POS

Algorithme 3: Désambigüisation lexicale : flexionToNodeDisambiguation

Input : Une liste N de k nœuds à désambigüiser et une liste S de nœuds précédents dans la requête

Output : Le nœud p le plus probable dans le contexte de la requête

```
// Pour les cas particuliers, on utilise une liste de règles ad hoc
if (Un nœud e parmi les k) ∈ StopList then p ← applyRule(e, StopList );
else
    score ← {};
    foreach nœud e de la liste N do        // On attribue un score à chaque cas
        // Formule ajustée empiriquement
        score ← score ∪ {2 * freq(G_LL, S[-1], e) + 2 * freq(G_KK, S[-1], e) +
        freq(G_¬GL, S[-1], e) + freq(G_PP, S[-1], e) + freq(G_PPP, S[-2], S[-1], e)};
    end
    idx ← getIndicesMax(score);
    if |idx| > 1 then   // On départage en choisissant le POS le plus fréquent
        scoreComp ← {};
        foreach nœud e de ⋃_{i∈idx} N[i] do
            scoreComp ← scoreComp ∪ {freq(G_L, e)};
        end
        idx ←getUniqueIndiceMax(scoreComp);
    end
    p ← N[idx];
end
return p;
```

4.3 Étiquetage sémantique et désambigüisation

4.3.1 Clés sémantiques

On a vu dans la section 1.2.2.3 que pour donner du sens aux constituants de la phrase, il fallait les lier à des concepts liés entre eux au sein d'une ontologie, à la manière des synsets de WordNet [Fellbaum, 1998]. Par ailleurs, lors de la présentation du lexique $\mathbb{L}_G$ de `GRASP`, nous avons vu que l'attachement d'une sémantique lexicale aux nœuds se faisait via l'attribut KEYS qui contient une ou plusieurs **clés sémantiques** (ou simplement "clés"), parmi l'ensemble $\mathbb{K}_G$ de toutes les clés définies.

4.3.1.1 Structure

Une clé sémantique de `GRASP` est définie par 5 attributs :

1. un identifiant unique, utilisé pour référencer la clé et représenté par un symbole écrit en majuscules ;
2. sa position dans l'ontologie sous forme d'arbre à deux niveaux (représentant la classe et la sous-classe de la clé) ;
3. une catégorie grammaticale (plus générique que celle contenue dans le tag POS) parmi quatre valeurs possibles : VERB, NAME, PRED (notamment pour les adjectifs) et GRAM ;
4. une glose (en anglais) expliquant le sens de la clé (comme dans WordNet) ;
5. la liste des lemmes du corpus *Daft* ayant cette clé parmi leurs sens possibles.

Un exemple de clé est donné par la figure 4.4. On pourra se reporter à l'annexe B pour une liste de l'ensemble des clés définies.

```
{
  ID[TOCLICK],
  ONTO[‘¤INTERFACE’, ‘*BUTTON’],
  TYPE[VERB],
  COM[‘Denotes the action of clicking with the mouse’]]
  LEM[‘appuyer’, ‘clicker’, ‘enfoncer’]
}
```

Figure 4.4 Exemple de clé sémantique de `GRASP` : TOCLICK

4.3.1.2 Intérêt

Moralement, les clés sémantiques ont la même valeur que les synsets. D'un point de vue ensembliste, on peut considérer que les clés sont donc plus ou moins incluses dans les synsets de WordNet, étant 100 fois moins nombreuses (puisqu'il y a quelques 130 000 synsets dans WordNet contre environ 1 300 clés dans $\mathbb{K}_G$). Cependant, la plupart des synsets de WordNet sont beaucoup plus fins et précis (à part dans les domaines liés aux applications informatiques),

et la correspondance 1 à 1 est donc assez difficile à établir. Enfin, par rapport à WordNet où une vingtaine de relations permettent de lier les concepts entre eux, l'ontologie de $\mathbb{K}_G$ est beaucoup plus simple, les classes et sous-classes permettant seulement de grouper des concepts autour de thèmes communs.

Il nous a toutefois semblé plus commode de définir nos propres clés à partir du corpus plutôt que de chercher les synsets WordNet équivalents, en regroupant les lemmes apparaissant effectivement dans le corpus dans des mêmes catégories sémantiques auxquelles on pouvait associer un identifiant et une glose qui soient adaptés à leur utilisation réelle en situation d'assistance.

4.3.1.3 Méthodologie de définition

En pratique, la définition des clés de `GRASP` s'est faite en trois temps. Tout d'abord, `GRASP` a été conçu comme un simple lemmatiseur (où les nœuds ne disposaient pas de l'attribut KEYS) que nous avons pu utiliser pour générer automatiquement une liste de l'ensemble des lemmes du corpus. Dans un second temps, nous avons organisé (par séparation et regroupement) ces lemmes manuellement en sous-ensembles associés à une glose et un identifiant. Finalement, ces identifiants ont été introduits dans $\mathbb{L}_G$ via l'attribut KEYS.

4.3.1.4 Évaluation

À partir des 4 266 entrées du lexique $\mathbb{L}_G$, nous avons défini 1 294 clés sémantiques, réparties en 17 classes et 128 sous-classes (*cf.* tableau 4.7). Cela signifie donc que même si un lemme peut avoir plusieurs sens, il y a globalement moins de sens possibles que de façon de l'exprimer et plusieurs lemmes sont donc synonymes (au moins pour un de leur sens). Ceci apparaît dans le tableau 4.6 pour les clés ayant le plus de lemmes associés. Bien que cela n'apparaisse pas directement, le fait que certaines clés aient plus de 20 lemmes associés laisse deviner qu'au contraire, la majorité (58,6%) ont seulement 2 lemmes associés ou moins, et cette distribution suit globalement une loi de Zipf (comme classiquement en linguistique).
Le tableau 4.6 permet également de voir quelques choix de regroupement effectués, par exemple le fait que nous avons choisi de regrouper par exemple tous les nombres (AKOCARDINAL) et adjectifs numéraux ordinaux (ORDNTH) plutôt que de créer une clé propre à chacun. La clé NOKEY est quant à elle associée à des mots qui n'ont pas (dans le corpus) de sens individuellement et appartiennent à une locution.

4.3.2 Désambigüisation sémantique

Nous avons vu précédemment (*cf.* figure 4.1) que des entrées de $\mathbb{L}_G$ pouvaient être associées à plusieurs clés de $\mathbb{K}_G$. D'un point de vue quantitatif, 287 des 4 266 entrées de $\mathbb{L}_G$ (6,7%) sont ambigües. Cette proportion est relativement faible par rapport à la langue en général. Cela est

Rang	Clé	Nombre de lemmes associés	Exemples de lemmes associés
1	ISFORCELARGE	65	bien, carrement, radicalement, tellement, tres...
2	ORDNTH	45	troisieme, huitieme, huitiemement, quatre-vingtieme...
3	AKOCARDINAL	42	cinq, treize, vingtaine, cent, centaine, milliard...
4	ISINCOMPETENT	41	abruti, bete, maladroit, simplet, LOC[bon, a, rien]...
5	NOKEY	40	abord, aujourd, ci, dalle, extenso, fur, train, vis...
6	AKOPUNCTUATION	33	accolade, virgule, LOC[point, de, exclamation]...
7	ISCERTAIN	29	certain, clair, inconstestable, surJ, LOC[sans, doute]...
8	ISSURPRISING	28	atypique, bizarre, inattendu, insolite, LOC[ha, bon]...
9	ISHABITUAL	27	classiquement, habituel, normal, LOC[en, general]...
10	ABOUT	27	concernant, dedie, relatif, surG, LOC[a, le, sujet, de]...

Tableau 4.6 Les 10 clés sémantiques les plus fréquentes dans $\mathbb{L}_G$

dû au fait qu'on circonscrit le domaine des clés sémantiques aux occurrences effectives dans le corpus *Daft*.

De plus, parmi celles-ci, on observe que le nombre de clés possibles peut varier de deux à quatre (*cf.* les exemples de la figure 4.5), avec une forte majorité de cas où seulement deux sens sont possibles (*cf.* tableau 4.8). Là aussi, ce nombre de variantes est plus faible que ce que l'on trouve dans la langue dans son ensemble, ce qu'on peut rapprocher du phénomène décrit par Gale *et al.* [1992] selon qui lorsqu'un lemme a plusieurs significations possibles, il n'est employé que dans un seul de ces sens au sein d'un même discours (ici, un même contexte : l'assistance). D'autres sens sont bien évidemment possibles de manière générale (par exemple "donner son soutien à" pour "appuyer"), mais nous ne les avons pas constatés dans le corpus *Daft*.

Nous pouvons alors définir l'ensemble $\mathbb{A}_{\text{EK}}$ que constituent les couples associés $(e, k) \in (\mathbb{L}_G \times \mathbb{K}_G)$ où $|e.KEYS| > 1$, et qui est tel que $|\mathbb{A}_{\text{EK}}| = 618$ (*cf.* tableau 4.8).

abandonner	:	TOABANDON	TOABORT		
appuyer	:	TOCLICK	TODEPENDON		
sur	:	ISPOSUPON	ABOUT		
proposer	:	TOADVISE	ISAVAILABLE	TOINTENT	
derriere	:	ISPOSUNDER	ISPOSVERSO	ISSUCCESSOR	AKOBODYPART
retrouver	:	TOMEET	TONEEDHELP	TOTURNOUT	TOFIND

Figure 4.5 Exemples d'entrées de $\mathbb{L}_G$ ambigües : 1 lemme - N clés sémantiques

Il faut donc, comme dans le cas où plusieurs lemmes étaient liés à une même flexion, procéder à une étape de désambigüisation en fonction du contexte de la phrase, généralement nommée Word Sense Disambiguation (WSD – *cf.* section 1.2.2.3). Ici, pour rester sur notre logique consistant fonder la chaîne de traitement sur l'étude du corpus, nous avons opté pour

Classe	Sous-classe (+ nombre de clés associées)
¤INTERACTING	*MEETING (9), *INTERACTION (8), *SPEAKING (13), *QUESTIONNING (4), *REQUESTING (3), *SOLLICITING (3), *THANKING (5), *HESITATING (3), *PROMPTING (2), *MANAGING (17), *SOCIALIZING (8), *EXCHANGING (16), *ENTERTAINING (8), *PLAYING (3)
¤ARGUING	*FIGHTING (14), *DISTURBING (5), *INSULTING (4), *CONGRATULATING (3), *APOLOGIZING (4), *PROMISING (2), *FRIENDLINESS (4), *PROUDNESS (9), *SATISFACTION (2), *PATIENCE (6), *HONESTY (8)
¤HELPING	*ASSISTING (9), *COOPERATING (4), *INFORMING (16), *DOCUMENTING (14)
¤OPINION	*FEELING (12), *WANTING (5), *CARING (5), *LIKING (15), *WORTH (21), *LUCK (2), *EFFORT (6), *REGULARITY (7), *ETHICS (3), *GENERALITY (2), *REALITY (6)
¤TIME	*DATE (41), *TIMEPOSITION (10), *TIMEDURATION (14), *FREQUENCY (14), *AGE (5)
¤THINKING	*REASONING (5), *RESOLVING (4), *REALIZING (6), *EXPECTING (7), *CHOOSING (5), *REMEMBERING (4)
¤PROCESS	*EVOLUTION (14), *PLANNING (9), *STATUS (14), *OPERATIONALITY (14), *CONTROLLING (35), *TUNING (4), *SPEED (5), *CAUSING (13), *INFLUENCING (8), *HISTORY (3)
¤CONFIDENCE	*CHECKING (7), *TRUSTING (4), *BELIEVING (11), *CORRECTNESS (10), *CONFUSING (9), *WORRYING (7), *DANGER (14), *PRIVACY (4)
¤AVAILABILITY	*OBLIGATION (7), *AUTHORIZATION (14), *CONSTRAINING (6), *POSSIBILITY (4), *EXISTENCE (9), *CAPACITY (9)
¤OPERATION	*OPERATING (8), *TRYING (10), *UNDOING (4), *REPEATING (3), *USING (8)
¤SYSTEM	*ACTORS (6), *ENVIRONMENT (4), *HARDWARE (14), *FILE (13), *APPLICATION (11), *DESIGNING (8)
¤INTERFACE	*WINDOW (9), *BAR (8), *BUTTON (4), *PICTURE (7), *TEXT (20), *SOUND (6), *DISPLAY (17), *SHAPE (12), *COLOR (22), *STYLE (9)
¤STRUCTURE	*STRUCT (11), *STRUCTURING (35), *EXTRACTING (11), *DATATYPE (34), *CALCULATING (18), *MEASURING (10), *COMPARING (21), *NAVIGATING (20), *SEARCHING (7), *MODIFYING (19), *CONFIGURING (11), *IDENTITY (17)
¤SPACE	*SPACEPOSITION (30), *SPACEDISPOSITION (5), *SPACEATTRIBUTE (14), *SPACEMOVE (12), *SPACEMODIFY (7)
¤QUANTIFYING	*INTENSITY (6), *QUANTIFIER (15), *QUANTITY (18), *ORDINALITY (13)
¤GRAMMAR	*PUNCTUATION (1), *LITERAL (2), *RELATION (34), *AUXILIARY (3), *DETERMINANT (7), *PRONOUN (15), *CONJUNCTION (2), *FRENCH (3)
¤WORLD	*PERSON (14), *GROUND (32), *ADHOC (6)

Tableau 4.7 Organisation de l'ontologie des 1 294 clés sémantiques de $\mathbb{K}_G$

Nombre de clés associées à un lemme	Nombre d'entrées de $\mathbb{L}_G$	Proportion de l'ensemble des entrées ambigües	Nombre de couples (lemme, clé)
2	247	86%	494
3	36	12,6%	108
4	4	1,4%	16
TOTAL	287	100%	618

Tableau 4.8 Entrées de $\mathbb{L}_G$ ambigües en fonction du nombre de clés associées

une approche à la fois :

- **statistique**, c'est-à-dire fondée sur une étude fréquentielle des sens en fonction du contexte de la requête, par opposition à l'utilisation de règles de désambigüisation générales, établies hors de tout contexte. L'utilisation de règles ad hoc nous ferait en effet perdre l'avantage de travailler sur un sous-langage.
- **supervisée**, c'est-à-dire passant par l'annotation manuelle du corpus, par opposition à l'utilisation de méthodes d'apprentissage non supervisé pour définir des clusters de sens. L'approche non supervisée permet en effet d'éviter le problème de subjectivité de l'annotation décrit par Véronis [1998], mais il est important pour nous par la suite de connaître précisément les sens des clés sémantiques employées.

4.3.2.1 Annotation du corpus

Parmi les 11 626 requêtes du corpus *Daft*, 1 241 contiennent au moins une des 287 entrées sémantiquement ambigües. La première étape nécessaire consiste donc à identifier pour chacune de ces phrases le sens correct des entrées ambigües qu'elles contiennent. Par exemple, deux clés sémantiques sont associées au lemme "abandonner" :

- TOABORT pour le sens "mettre fin à une action ou un processus en cours",
- TOABANDON pour le sens "laisser une personne seule".

On peut alors extraire automatiquement les (six) phrases lemmatisées du corpus où apparaît ce lemme et leur associer manuellement la clé sémantique correcte, ce qui donne la répartition représenté sur la figure 4.6.

TOABORT	**TOABANDON**
"J'abandonne"	"Je t'abandonne"
"j'abandonne la partie"	"je ne me sens pas vraiment abandonné"
	"Ne m'abandonne pas !"
	"si toi aussi tu m'abandonnes..."

Figure 4.6 Exemple de phrases associées aux différents sens d'une entrée ambigüe de $\mathbb{L}_G$ ("abandonner"), tels qu'observés dans le corpus *Daft*

Avec cette annotation, nous pouvons obtenir, sur les phrases du corpus, une performance de 100% de désambigüisation. Néanmoins, l'objectif étant de pouvoir désambigüiser efficacement

ces lemmes dans de nouvelles phrases, il nous faut utiliser un algorithme nous permettant de généraliser ces résultats.

4.3.2.2 Définition de l'algorithme

Pour effectuer la WSD, il est possible de se baser sur les lemmes ou sur les clés sémantiques présents dans la requête et qui fournissent donc le contexte d'emploi du lemme, et différentes fenêtres (nombre de voisins) peuvent être testées. Nous avons en particulier considéré deux méthodes possibles :

- une approche fondée sur les clés sémantiques de tous les autres lemmes de la phrase, que nous appellerons WSD_{K_*} et qui est implémentée par la fonction wsdSem,
- une approche prenant uniquement en compte une fenêtre restreinte composée des deux lemmes qui précèdent et des deux lemmes qui suivent celui à désambigüer (soit un 5-gramme centré sur le lemme à identifier), que nous appellerons WSD_{L_2} et qui est gérée par la fonction wsdLem.

Utilisation de toutes les clés (WSD_{K_*}). Afin d'exploiter l'annotation effectuée, il faut dans un premier temps construire la matrice de cooccurrence M_K (essentiellement creuse) de taille $|\mathbb{K}_G| \times |\mathbb{A}_{\text{EK}}|$ qui associe à chaque couple [(lemme ambigu l, clé associée k), clé k'] une pondération non nulle lorsque k' est présente dans une des phrases du corpus où le lemme l a la valeur sémantique k.

$$M_K = \begin{array}{c} \text{couples (lemme, clé)} \\ \left(\begin{array}{cccc} w_{1,1} & w_{1,2} & \cdots & w_{1,|\mathbb{A}_{\text{EK}}|} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,|\mathbb{A}_{\text{EK}}|} \\ \vdots & \vdots & \ddots & \vdots \\ w_{|\mathbb{K}_G|,1} & w_{|\mathbb{K}_G|,2} & \cdots & w_{|\mathbb{K}_G|,|\mathbb{A}_{\text{EK}}|} \end{array} \right) \text{clés} \end{array}$$

La matrice M_K peut se construire en appliquant la formule suivante :

$$\forall (l, k_l) \in \mathbb{A}_{\text{EK}}, M_K\left[k_i, (l, k_l)\right] = \sum_{r \in Daft} \sum_{\substack{n \in \mathcal{G}(r) \\ n_L \neq l \\ n_K \notin \mathbb{K}_{STOP}}} \frac{1}{|\mathcal{G}(r)|}$$

où : $\mathcal{G}(r)$ correspond à l'analyse par GRASP et mise sous forme de nœuds de la requête r,
$\mathbb{K}_{STOP}$ est une stoplist de clés de $\mathbb{K}_G$ non utiles pour la discrimination sémantique,
n_L est le lemme associé au nœud n (*i.e.* contenu du champ LEM),
n_K est la/les clé(s) associée(s) au nœud n (*i.e.* contenu du champ KEYS).

L'ensemble $\mathbb{K}_{STOP}$ a été établi de manière empirique, et regroupe essentiellement les mots grammaticaux les plus fréquents : $\mathbb{K}_{STOP}$ = {QUEST, THIS, TOBE, THESYSTEM, A, OF, AKOPUNCTUATION, $A, TOCAN, THEUSER, $QUE, IT, UNDEFPRON, LITERAL, TO-HAVE, THE, NEG, INORDERTO, THAT, TOEXECUTE}.

À partir de cette matrice, il est alors possible de déterminer le score associé à chaque clé sémantique k_i possible pour un lemme $l \in \mathbb{L}_G$ en appliquant la fonction wsdSem décrite par l'algorithme 4. La clé sémantique choisie est alors celle ayant le score le plus élevé dans le vecteur de pondération retourné.

Algorithme 4: WSD utilisant toutes les clés de la phrase : wsdSem

Input : Un lemme ambigü l de $\mathbb{L}_G$ et la phrase P de N nœuds lemmatisée par `GRASP` dont l est extrait

Output : Un vecteur w de p poids ($p \in [\![2, 4]\!]$ est le nombre de clés possibles pour l)

```
KP = {};
foreach nœud n ∈ N do                    // Extraction des nœuds discriminants de P
    if n ∉ K_STOP then KP = KP ∪ {n}
end
w = {};
foreach clé k_l associée à l dans L_G do
    score = 0;
    foreach clé de la phrase k_p ∈ KP do
        score = score + M_K [k_p, (l, k_l)];
    end
    w = w ∪ score;
end
return w;
```

Utilisation des lemmes précédents ($\mathbf{WSD}_{L_2}$). Comme pour WSD_{K_*}, il est tout d'abord nécessaire de construire une matrice M_L^- (resp. M_L^+) de listes qui associent à chaque couple (lemme ambigu l, clé associée k) $\in \mathbb{A}_{\mathrm{EK}}$ la fréquence des deux lemmes précédant (resp. suivant) les différentes occurrences de l au sens de k au sein des phrases du corpus *Daft*. Les deux matrices ont donc le même format :

$$M_L^- = \begin{pmatrix} L_{l_1,k_{1,1}} \\ L_{l_1,k_{1,2}} \\ \vdots \\ L_{l_{287},k_{287,i}} \end{pmatrix}$$

$$= \begin{pmatrix} \{((l'_{1,1,1}, l''_{1,1,1}), w_{1,1,1}), ((l'_{1,1,2}, l''_{1,1,2}), w_{1,1,2}), \dots ((l'_{1,1,n}, l''_{1,1,n}), w_{1,1,n})\} \\ \{((l'_{1,2,1}, l''_{1,2,1}), w_{1,2,1}), ((l'_{1,2,2}, l''_{1,2,2}), w_{1,2,2}), \dots ((l'_{1,2,m}, l''_{1,2,m}), w_{1,2,m})\} \\ \vdots \\ \{((l'_{287,i,1}, l''_{287,i,1}), w_{287,i,1}), ((l'_{287,i,2}, l''_{287,i,2}), w_{287,i,2}), \dots ((l'_{287,i,p}, l''_{287,i,p}), w_{287,i,p})\} \end{pmatrix}$$

où : l' est pour M_L^- (resp. M_L^-) au lemme situé 2 positions avant (resp. juste après) l,
l'' est pour M_L^- (resp. M_L^+) au lemme situé juste avant (resp. 2 positions après) l,
w est une pondération associée au couple de lemmes le précédant.

Comme la taille d'une liste $L_{l,k}$ dépend des bigrammes précédents (resp. suivants) observés dans le corpus *Daft*, en règle générale on a $|L_{l,k}| \neq |L_{l',k'}|$ (et même $|L_{l,k}| \neq |L_{l,k'}|$).

Afin de normaliser les observations faites pour chacun des différents sens d'un lemme l, on définit un coefficient de pondération $r_{l,k}$ tel que :

$$\forall (l,k) \in \mathbb{A}_{\text{EK}}, r_{l,k} = \frac{|r, r \in \mathit{Daft} \wedge (\exists n \in \mathcal{G}(r), n_L = l \wedge n_K = k)|}{|r, r \in \mathit{Daft} \wedge (\exists n \in \mathcal{G}(r), n_L = l)|}$$

Chaque liste de la matrice M_L^- peut se construire en appliquant la formule suivante[4] :

$$\forall (l,k) \in \mathbb{A}_{\text{EK}}, L_{l,k} = \biguplus_{r \in \mathit{Daft}} \biguplus_{\substack{n \in \mathcal{G}(r) \\ n_L = l \\ n_K = k}} \{((n_{L-2}, n_{L-1}), r_{l,k})\}$$

où pour une liste $L = \bigcup_{i=1}^{N} \{((l_i, k_i), w_i)\}$ et un couple $e = \{((l,k), w)\}$, l'opération $\uplus$ est définie comme :

$$L \uplus e = \begin{cases} \bigcup_{i=1}^{N} \{((l_i,k_i),w_i)\} \cup \{((l,k),w)\} & \text{si } \forall i, (l_i,k_i) \neq (l,k) \\ \bigcup_{i=1}^{p-1} \{((l_i,k_i),w_i)\} \cup \{((l,k),w_p+w)\} \cup \bigcup_{j=p+1}^{N} \{((l_j,k_j),w_j)\} & \text{si } l_p = l \wedge k_p = k \end{cases}$$

et n_{L+i} (resp. n_{L-i}) signifie le lemme du i$^{\text{e}}$ nœud après (resp. avant) le nœud n de lemme L.

Algorithme 5: WSD utilisant le bigramme de lemmes précédents et suivants : wsdLem

Input : Un lemme ambigu l de $\mathbb{L}_G$ et la phrase P de N nœuds lemmatisée par `GRASP` dont l est extrait

Output : Un vecteur w de p poids ($p \in [\![2,4]\!]$ est le nombre de clés possibles pour l)

$w = \{\}$;
n_L = `getNodeFromLemma(`N, l`)` ; `// On récupère le nœud ambigu dans la phrase`
Lbef = `getListFromMatrix(`M_L^-, l, k`)`;
Laft = `getListFromMatrix(`M_L^+, l, k`)`;
foreach *clé* k_l *associée à* l *dans* $\mathbb{L}_G$ **do**
  if $\{(\mathsf{n}_{L-2}, \mathsf{n}_{L-1}), p\_\} \in$ Lbef **then**
    $w = w \cup p$;
  else
    $w = w \cup 0$;
  end
  if $\{(\mathsf{n}_{L+2}, \mathsf{n}_{L+1}), p\_\} \in$ Laft **then**
    $w[-1] = w[-1] + p$;
  end
end
return w;

Dans ces conditions, il est alors possible de déterminer pour une nouvelle requête le score de chaque clé sémantique k_i possible pour un lemme $l \in \mathbb{L}_G$ en appliquant wsdLem, fonction décrite par l'algorithme 5. Comme pour wsdSem, la clé sémantique choisie est celle ayant le score le plus élevé dans le vecteur retourné.

[4] Pour M_L^+, il suffit de remplacer (n_{L-2}, n_{L-1}) par (n_{L+1}, n_{L+2}).

Algorithme de WSD final. La comparaison des résultats obtenus sur les lemmes de nœuds ambigus des phrases du corpus *Daft* avec les deux méthodes (wsdSem et wsdLem) montre que ni l'une ni l'autre n'est en mesure de fournir des résultats parfaits : sur les 287 lemmes ambigus, seulement 198 (69,0%) sont systématiquement correctement classifiés par WSD_{K_*}. Les performances sont un peu meilleures avec la WSD_{L_2} qui en classifie correctement 239 (83,3%). Toutefois, dans les deux cas, le problème semble venir de la taille trop restreinte du corpus *Daft* qui fait que, dans une nouvelle phrase à analyser, soit les bigrammes de lemmes précédents ou suivants ne sont pas déjà recensés dans le corpus, soit toutes les clés cooccurrentes se retrouvent toutes dans $\mathbb{K}_{STOP}$ et donc aucun résultat ne peut être trouvé.

Algorithme 6: WSD d'une phrase par l'utilisation conjointe de 3 méthodes

Input : Une phrase P de N nœuds, préalablement lemmatisée par `GRASP`
Output : Une phrase P' de N nœuds, sémantiquement désambigüisée

```
foreach nœud n de P do
    if |n.KEYS| > 1 then
        poids = wsdLem(n.LEMS,P);
        if poids == Null then
            poids = wsdSem(n.LEMS,P);
        else
            poids = poids + wsdSem(n.LEMS,P);
        end
        if poids == Null then poids = getMostFrequentKeyForLemma(n.LEMS)
        n' = getHighestScoredKey(poids );
        n ← n';
    end
end
return P;
```

L'algorithme employé (*cf.* algorithme 6) pour effectuer la WSD globale (notée WSD_G) repose donc sur l'utilisation conjointe de trois méthodes : d'une part les deux précédentes (en favorisant la `wsdLem` en premier lieu puisqu'elle fournit les meilleurs résultats), et, d'autre part, lorsque `wsdLem` et WSD_{K_*} échouent toutes les deux, en ayant recours à l'utilisation du sens le plus fréquent dans le corpus.

4.3.2.3 Résultats et perspectives d'amélioration

Si l'on considère l'ensemble des 8 474 nœuds ambigus existants dans le corpus *Daft*, l'utilisation de WSD_G permet d'identifier le sens correct dans 8 259 cas (soit 97,5%). Les 215 erreurs sont réparties sur 48 lemmes différents selon une loi de Zipf (*cf.* tableau 4.9).

Afin d'améliorer les performances à partir des méthodes employées, trois approches (non mutuellement exclusives) semblent possibles :

– recueillir de nouvelles requêtes pour augmenter la représentativité du corpus *Daft*, et donc diminuer les chances que les méthodes de WSD_{K_*} et `wsdLem` ne renvoient aucun

Rang	Lemme	Nombre d'occurrences dans *Daft*	Nombre d'occurrences mal désambigüisées
1	mettre	167	41 (24,5%)
2	pour	805	39 (4,8%)
3	chercher	57	15 (26,3%)
4	voir	143	12 (8,4%)
5	dire	460	10 (2,2%)
6	alors	48	8 (16,6%)
7	surG	595	8 (1,3%)
8	en	589	7 (1,2%)
9	plus	157	6 (3,8%)
10	tout	271	4 (1,4%)

Tableau 4.9 Les 10 lemmes les plus mal désambigüisés (en nombre)

résultat ;

- redéfinir certaines clés de $\mathbb{K}_G$, par exemple en fusionnant des clés au sens assez proche (*e.g.* TOSEE et TOCONSULT comme sens possibles associés au lemme "voir", parmi les plus mal désambigüisés) ;
- définir des règles grammaticales et sémantiques ad hoc pour gérer les cas non traitables par la WSD_G statistique.

L'extension de la taille du corpus est, de manière générale, dans les approches statistiques, la solution par excellence. Toutefois, on a vu les difficultés liées à la constitution d'un corpus dans le chapitre 3, et pour obtenir une amélioration significative, il faudrait probablement une augmentation d'au moins un ordre de grandeur (passage de 10 000 à 100 000 requêtes). En ce qui concerne la redéfinition des clés, bien que possible, elle nous semble devoir se faire selon des impératifs liés à l'utilisation pratique de leur sens par l'agent rationnel : il n'y aurait en effet pas grand sens à fusionner deux sens pour améliorer le score de la WSD, si on se rend compte par la suite que leur distinction est utile dans la définition d'une réaction. Par conséquent, la dernière méthode, qui revient à mêler l'approche statistique jusqu'ici employée à des méthodes plus classiques fondées sur des règles linguistiques, semblerait la plus intéressante à explorer : certaines des règles mentionnées dans la section 4.4 suivante visent d'ailleurs à accomplir ceci.

4.4 Analyse grammaticale des constituants

4.4.1 Principe

Dans cette section, nous supposons que l'identification du lemme, du POS et de la clé sémantique liés à un nœud a été correctement effectuée lors des étapes précédentes. En pratique, dans le cas des phrases du corpus, celles-ci ayant été manuellement annotées, cela signifie donc que l'on fait appel à ces annotations dans les cas où des erreurs existaient lors de l'identification automatique.

Le dernier sous-module de `GRASP` a pour objectif d'effectuer une analyse grammaticale de manière à rattacher les uns aux autres les différents nœuds qui constituent la phrase, par

exemple “le petit bouton rouge” est un syntagme nominal et on peut donc rattacher les nœuds liés aux lemmes “le”, “petit” et “rouge” au nœud de “bouton”. Formellement, un nœud peut alors contenir un sous-arbre de nœuds liés au lemme principal du constituant.
Soulignons que l'on ne prend pas ici en compte la sémantique de la phrase (qui sera traitée par le module `DIG` dans le chapitre 6) mais bien uniquement l'organisation grammaticale - même si les deux éléments sont partiellement corrélés.

4.4.2 Règles

Afin d'établir des liens entre les différents nœuds, nous avons recours à un ensemble de règles de réécriture, que nous appellerons $\mathbb{R}_G$. Une règle $r \in \mathbb{R}_G$ est une fonction de transformation d'un ensemble de k nœuds N en un ensemble de k' nœuds N' (où k' peut être différent de k). L'ensemble des règles de $\mathbb{R}_G$ est appliqué séquentiellement à toute requête.

4.4.2.1 Syntaxe d'une règle

Une règle est constituée de trois éléments :

1. Un *identifiant* unique, représenté par une chaîne de caractères ;
2. Un *schéma*, plus ou moins complexe, définissant la séquence de nœuds à laquelle la règle peut s'appliquer (pouvant faire appel à n'importe quel attribut d'un noeud recensé dans le tableau 4.3) ;
3. Une *réaction*, formée d'une séquence d'actions basiques (créer (`§GNnew`), supprimer (`§GNdel`), ajouter un champ (`§GNadd`), *etc.*) à effectuer sur les différents nœuds d'une requête qui correspondrait au schéma.

```
§GRdefrule[
  "Ndjnj",                                  Identifiant de la règle
  {d:DD|CD, j1:JJ, n:NN, j2:JJ},            Schéma : un déterminant ou un cardinal, suivi
  {                                         d'un adjectif, un nom et un second adjectif
    new = §GNnew[POS[JJ]];                  On crée un nœud de type adjectif
    §GNadd[new, COMP, {j1, j2}];            contenant les deux adjectifs
    §GNsetup[new, BAG, {j1, j2}];           au sein de ce nom composé.
    §GNadd[n, DET, d];                      On lie au nom le nœud du déterminant
    §GNadd[n, ADJ, new];                    et celui de l'adjectif composé.
    §GNsetup[n, BAG, {d, j1, n, j2}];       Le nœud du nom contient alors un sous-arbre
    n]                                      de 4 éléments, que l'on retourne
}];
```

Figure 4.7 Exemple de règle d'analyse grammaticale de $\mathbb{R}_G$: *Ndjnj*

Un exemple de règle est donné dans la figure 4.7, permettant de traiter le syntagme nominal “le petit bouton rouge”, et l'analyse par `GRASP` avant et après application de cette règle est représentée sur la figure 4.8.

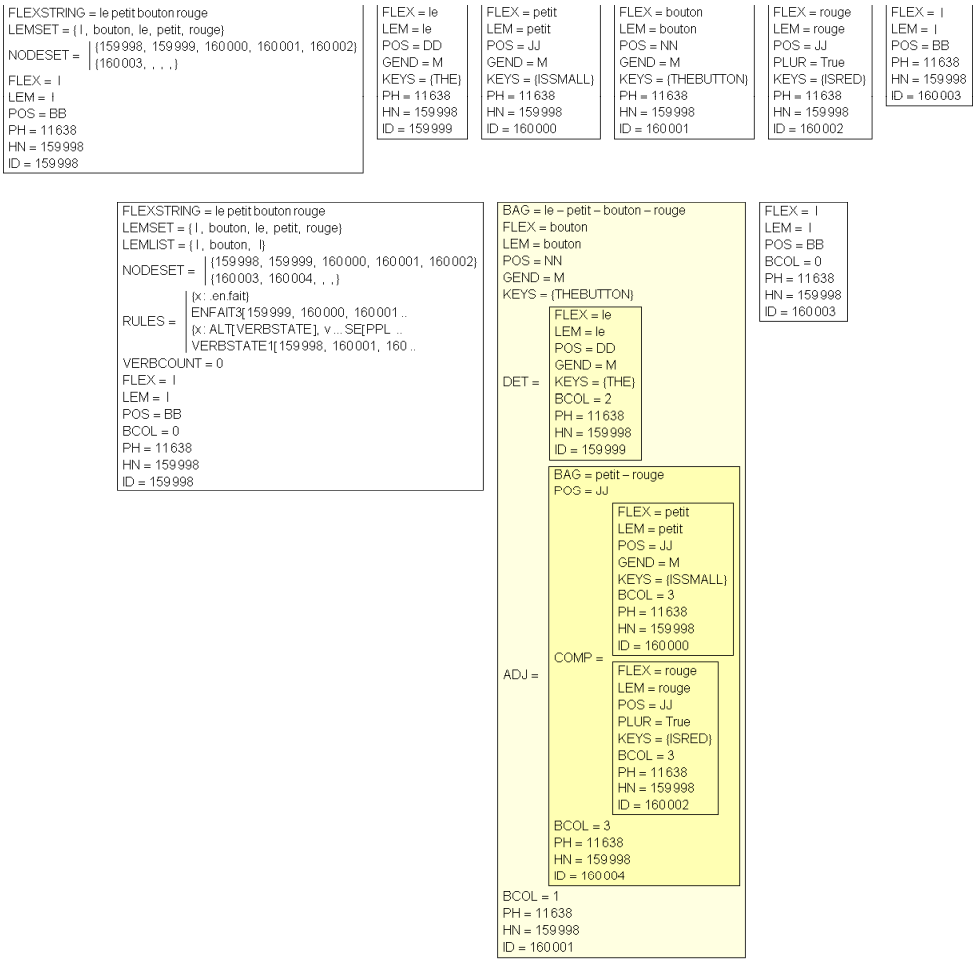


Figure 4.8 Analyse du syntagme nominal “le petit bouton rouge” par `GRASP`, avant (en haut) et après (en bas) application de la règle *Ndjnj* de la figure 4.7

4.4.2.2 Définition des règles

La définition des règles est fondée sur les observations faites à partir du corpus *Daft* : lorsqu'un phénomène est observé sur une phrase (par exemple le schéma déterminant-adjectif-nom-adjectif), on extrait toutes les autres occurrences de celui-ci au sein du corpus. À partir de là, on définit un ensemble d'actions permettant d'aboutir au regroupement de nœuds voulu. 240 règles ont ainsi été définies manuellement, traitant de phénomènes variés comme :

- les entités nommées : Jean Martin, monsieur Martin, ...
- les noms composés : la ligne de la liste, le bouton de gauche, ...
- les locutions complexes : de telles, peut-être, à mon avis, par exemple, ...
- les locutions verbales : mettre en évidence, vouloir dire, avoir l'idée de, ...
- les formes négatives : ne [VV] pas, aucun [NN] ne [VV], rien ne [VV], ...
- les formes interrogatives : à quoi [VV], [VV]-[PP], que [VV]-[PP], ...
- les verbes réflexifs : se bouger, se revoir, se passer, ...

L'ensemble de ces règles $\mathbb{R}_G$ figure en annexe D.

4.4.3 Résultats

Après application de l'ensemble des 240 règles à chacune des phrases du corpus, on obtient une réduction de près de la moitié (46,3%) du nombre de nœuds, passant de 101 726[5] à 47 110. Dans le cas de l'exemple de la figure 4.3, on est ainsi passé de 8 à 4 nœuds comme le montre la phrase complètement analysée sur la figure 4.9.

4.5 Conclusion

`GRASP` est un analyseur syntaxico-sémantique dédié aux requêtes d'assistance puisque ses ressources linguistiques sont fondées sur le corpus *Daft*. Pour toute requête r en langue naturelle fournie en entrée, il produit en sortie une forme $\mathcal{G}(r)$ structurée grammaticalement à l'aide des règles de $\mathbb{R}_G$, dans laquelle les différents constituants se sont vus associer une clé sémantique parmi les 1 294 de l'ensemble $\mathbb{K}_G$.
Notons également que `GRASP` est compatible avec une utilisation pratique au sein d'un agent conversationnel, puisque le temps de traitement est dépendant de manière linéaire de la longueur de la phrase en langue naturelle, et qu'en pratique, l'analyse de toutes les phrases du corpus *Daft* montre que le temps moyen de traitement d'une phrase est de l'ordre de 0,4 s.

[5]En fait, l'analyse complète du corpus produit 124 976 nœuds, mais pour chaque phrase on ne compte pas les nœuds de début et de fin (ayant pour identifiants 159 998 et 160 003 sur la figure 4.8) qui n'ont pas la même valeur et ne peut pas, par définition, être regroupés avec les autres. Ainsi, une phrase totalement analysée (tous les constituants ont été rattachés à un élément à la racine) aura tout de même trois nœuds.

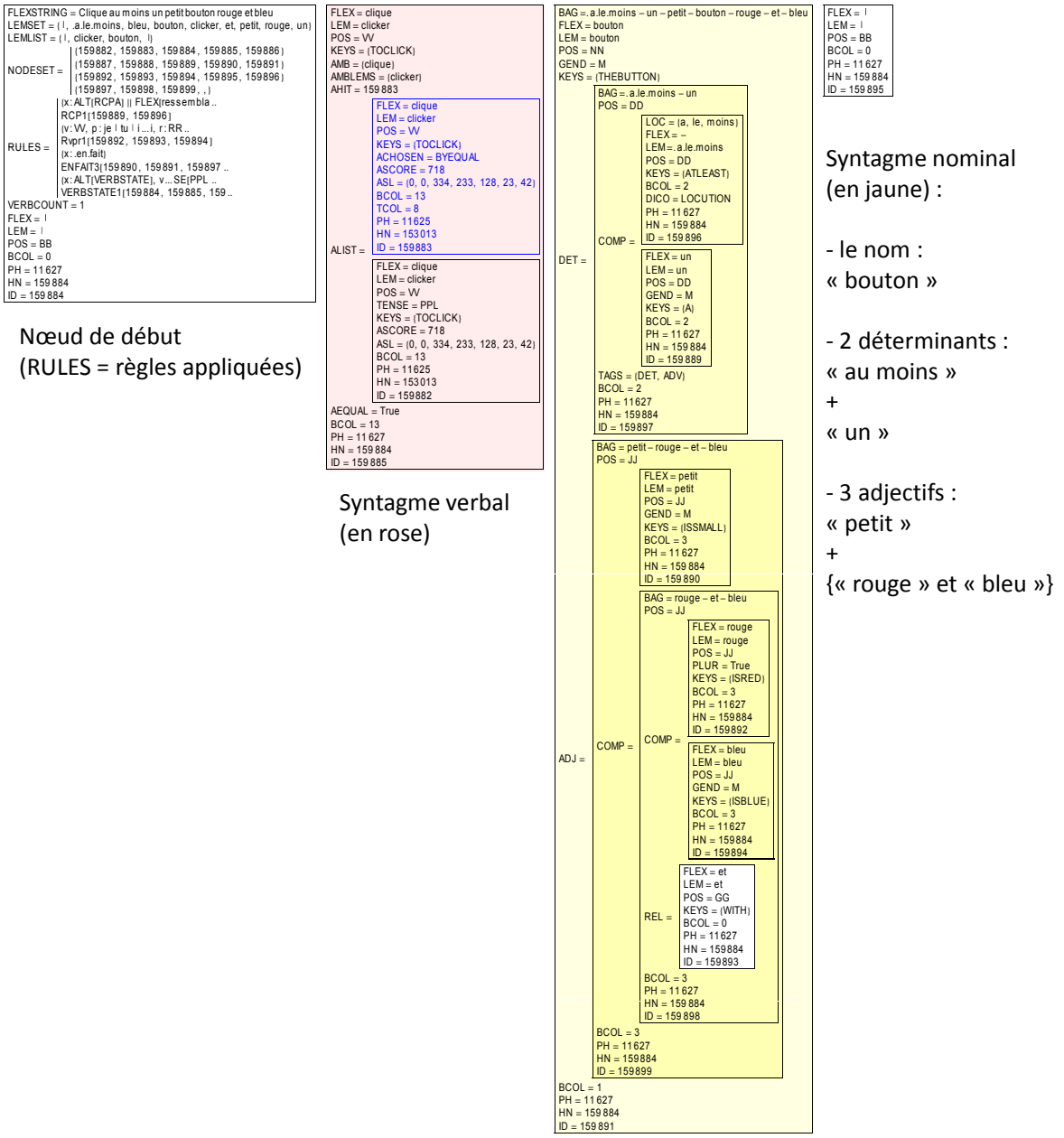


Figure 4.9 Exemple d'analyse en constituants par `GRASP` de la requête "Clique au moins un petit bouton rouge et bleu"

Chapitre 5

Langage formel de requêtes d'assistance

Sommaire

Afin de pouvoir réagir aux requêtes recueillies dans le corpus *Daft* introduit précédemment, l'agent doit avoir une représentation formelle de celles-ci à partir de laquelle un raisonnement peut être effectué, et c'est à cette représentation que nous allons nous intéresser dans ce chapitre. Nous présenterons donc dans un premier temps la méthodologie qui a été suivie pour annoter et extraire manuellement les éléments du langage formel et la façon dont ils se structurent les uns vis-à-vis des autres. Nous introduirons ensuite, de manière plus formelle, la syntaxe et la sémantique du langage *DAFT* issues de cette analyse empirique, et finalement nous évaluerons a posteriori ce langage par rapport au corpus *Daft*.

5.1 Méthodologie : un langage issu du corpus

L'objectif de cette section n'est pas d'introduire le vocabulaire du langage *DAFT* mais les principes ayant mené aux choix qui ont été faits, ceci à partir de quelques analyses de phrases-type :

- en termes de contenu : que souhaite-t-on représenter d'un point de vue sémantique ?
- en termes de syntaxe : comment souhaite-t-on représenter les concepts choisis ?

5.1.1 Exploitation de ressources déjà existantes

Nous avons préféré ne pas repartir de ressources existantes proches (comme par exemple Volem [Fernandez *et al.* , 2002] décrit dans la section 1.2.2.3) car ceci aurait requis un travail préalable de présélection des frames pertinentes de manière à augmenter la précision en supprimant les frames inutiles dans notre contexte. Or cette préselection prend du temps, et au final, elle permet de récupérer seulement les frames les plus triviales (*i.e.* fréquentes et donc faciles à extraire du corpus). Nous pensons donc que si elle peut être pertinente, il est préférable que cette comparaison soit plutôt faite a posteriori qu'a priori.

5.1.2 Choix des éléments à représenter

Concevoir un langage formel permettant de modéliser un sous-ensemble de la langue naturelle revient à faire ressortir les différents éléments qui constituent une requête et à expliciter les relations entre ceux-ci. La question qui se pose alors le plus souvent est de savoir le degré de finesse sémantique approprié : faut-il définir un élément spécifique du langage pour représenter la sémantique particulière propre au mot X ? Ou est-il préférable d'assimiler X à X' ayant un sens relativement proche ?
La réponse à apporter à ces questions est forcément sujette à débat :

- d'un côté, une spécialisation extrême va à l'encontre de l'abstraction nécessaire à toute compréhension[1] : si on considère comme éléments discriminants la taille, la couleur et la position de deux boutons portant le label "Quitter", on manque le fait vraisemblablement important, à savoir qu'ils ont la même fonction.
- d'un autre côté, une généralisation trop poussée empêche toute considération utile : les deux boutons "Quitter" susmentionnés peuvent être vus seulement comme deux boutons, voire comme deux composants de l'application, mais une telle définition s'applique également à un grand nombre d'autres éléments de l'application qui n'ont pas du tout la même fonction.

Dans le cas présent, comme on le voit dans les deux exemples ci-dessus, c'est le contexte qui est le nôtre qui nous permet de déterminer la réponse à ces questions. En effet, la distinction entre deux éléments dépend à la fois :

[1] "Penser, c'est oublier les différences, c'est généraliser, abstraire" [Borges, 1951].

– des requêtes que l'agent recevra en entrée (et donc du corpus qui est représentatif de celles-ci), car il n'est pas très utile de distinguer deux éléments proches si l'un des deux n'apparaît quasiment jamais en situation réelle d'utilisation ;
– des réactions attendues de la part de l'agent rationnel qui devra exploiter la requête formelle, car une fois deux éléments proches regroupés, la nuance entre les deux est définitivement perdue et aucune réaction ne pourra se faire à partir de celle-ci.

5.1.3 Choix du mode de représentation

On l'a vu dans le chapitre 2, le langage *DAFT* que nous souhaitons définir ici fait suite à une première version (*DAFT 1.0*), décrite dans la section 2.2.3.1, et qui exploitait un formalisme de type "frames". Ce type de représentation de la sémantique des requêtes est aussi celui mis en avant par Allen [1995], qui l'a mis en œuvre avec succès au sein du projet TRIPS [Allen *et al.* , 2000] qui demeure aujourd'hui encore une référence dans le domaine des systèmes de dialogues orientés tâche.
Bien qu'un changement complet du mode de représentation soit théoriquement envisageable (*cf.* l'état de l'art fait dans la section 1.2.2.3 pour voir les autres modes existants), une préférence existe donc pour repartir sur un langage utilisant un formalisme similaire. En l'absence d'arguments forts pour abandonner celui-ci, nous avons donc choisi de conserver ce mode de représentation, tout en étendant sa syntaxe par rapport à *DAFT 1.0* (*cf.* section 2.2.4.1).

5.2 Spécifications du langage *DAFT*

5.2.1 Syntaxe formelle

L'élément de base de la syntaxe du langage formel *DAFT* est appelé une **entité**. Une entité E est utilisée pour réifier toute notion sémantique et est formellement définie par :

$$\begin{aligned} E &= t_s : s\,[g, t_1 : a_1 = v_1, \ldots, t_n : a_n = v_n] \\ &= t_s : s\left[g, \bigcup_{i=1}^{n} t_i : a_i = v_i\right] \end{aligned}$$

où : $t_s \in \mathbb{T}_E$ est le *type* de l'entité E,
$s \in \mathbb{S}$ est un symbole représentant l'*identifiant* de l'entité E,
g est une chaîne de caractères représentant la glose définissant, en langue naturelle, la signification de l'entité, éventuellement à l'aide d'exemples, à la manière des synsets de WordNet [Fellbaum, 1998],
$t_i \in \mathbb{T}$ est le *type* du i^{e} attribut de l'entité E,
a_i est l'*identifiant* du i^{e} attribut (un symbole),

$v_i \in \delta_{t_i}$ est la *valeur* du i^{e} attribut, choisie au sein du domaine de validité défini pour le type t_i.

On nomme $\mathbb{E}$ l'ensemble (infini) des entités. Le contenu et la signification des différents ensembles évoqués ici ($\mathbb{T}$, $\mathbb{S}$, *etc.*) sont donnés dans la section 5.2.2.

Pour faire référence à l'identifiant, au type ou à la glose d'une entité E, on utilise respectivement la notation id[E], type[E] et gloss[E]. Le nombre n de triplets (t_i, a_i, v_i) est quant à lui noté $|E|$.

Exemple : une entité E_{ex_1} représentant le verbe "cliquer"

E_{ex_1} = {act} : Click[
"Une personne clique sur un objet d'une certaine manière",
{*person*} : clicker = ∅,
{*person*} : clicked = ∅,
{*manner*} : manner = ∅,
]

où id[E_{ex_1}] = Click, type[E_{ex_1}] = act et les trois attributs ont pour nom clicker, clicked et manner d'où $|E_{ex_1}| = 3$.

5.2.1.1 Identifiant et schème

L'identifiant, utilisé comme en-tête de l'entité, est ce qui définit les autres éléments (à l'exception des v_i). Autrement dit, si deux entités ont le même identifiant, seules leur valeurs peuvent être différentes :

$$\begin{aligned}\forall (E_1, E_2) \in \mathbb{E}^2, \text{id}[E_1] = \text{id}[E_2] \Rightarrow (&\text{gloss}[E_1] = \text{gloss}[E_2] \wedge \\ &\text{type}[E_1] = \text{type}[E_2] \wedge \\ &|E_1| = |E_2| \wedge \\ &\forall i \in [\![1, |E_1|]\!], t_i[E_1] = t_i[E_2] \wedge a_i[E_1] = a_i[E_2])\end{aligned}$$

Par conséquent, lorsqu'on ne s'intéresse pas aux valeurs prises par ses attributs, on peut faire référence à une entité de manière abrégée en mentionnant simplement son identifiant selon la notation : id[E][...] (*e.g.* E_{ex_1} = Click[...]).

L'association entre d'un côté un identifiant, et de l'autre un type, une glose, un ensemble d'identifiants d'attributs et des types associés est fait par la définition d'un *schème* pour chaque identifiant. Autrement dit, un schème peut être vu comme l'ensemble des éléments invariants entre deux entités ayant le même identifiant.

5.2.1.2 Type

Le type d'un attribut t_i peut être une combinaison de un ou plusieurs éléments parmi les suivants :

1. un symbole (appartenant à l'ensemble fini des symboles de $\mathbb{T}_E$),
2. une valeur choisie parmi un ensemble fini et discret écrit $[\![val_1, val_2, \ldots, val_n]\!]$,
3. une valeur choisie parmi un invervalle continu écrit $[borne_{inf}, borne_{sup}]$,
4. un type au sens informatique classique : *Boolean* ou *String*.

5.2.1.3 Attribut

Chaque identifiant d'attribut a_i a un nom unique dans le contexte de l'entité E à laquelle il appartient, c'est-à-dire que :

$$\forall E \in \mathbb{E}, \forall i, j \in [\![1, |E|]\!], i \neq j \Rightarrow a_i \neq a_j$$

Inversement, deux entités avec des identifiants différents peuvent avoir des identifiants d'attributs identiques :

$$(\forall (E_1, E_2) \in \mathbb{E}^2, (\forall i \in \mathbb{N}, a_i[E_1] = a_i[E_2])) \nRightarrow \mathrm{id}[E_1] \neq \mathrm{id}[E_2]$$

De plus, il n'y a pas de bijection entre un identifiant d'attribut et un type :

$$(\forall (E_1, E_2) \in \mathbb{E}^2, \mathrm{id}[E_1] \neq \mathrm{id}[E_2]), (i, j) \in \mathbb{N}^2, a_i[E_1] = a_j[E_2] \nRightarrow t_i[E_1] = t_j[E_2]$$

Les attributs associés à un schème ont été définis d'après ce qui a pu être observé au sein du corpus *Daft*. Cela signifie que les attributs d'une entité sont les paramètres qui sont statistiquement les plus souvent associés à la notion décrite dans la glose ; on ne vise donc pas à l'exhaustivité au niveau de la langue générale, mais seulement par rapport au corpus *Daft*.

5.2.1.4 Valeur

Une valeur v_i peut correspondre à :

1. une entité, si son type associé t_i est un symbole,
2. un ensemble fini d'entités, s'il s'agit d'un attribut à valeurs multiples (identifiable par le suffixe $*$ dans le nom de l'attribut),
3. une valeur terminale, si le type est tout sauf au symbole,
4. vide (noté $\emptyset$), dans tous les cas, quand aucune valeur n'est associée à l'attribut a_i.

Par défaut, les valeurs sont vides. Cela signifie qu'une entité n'a pas besoin d'avoir une valeur associée à chaque attribut pour être valide (*cf.* E_{ex_1}).

5.2.2 Sémantique opérationnelle

5.2.2.1 Classes d'éléments

Les entités et les schèmes de *DAFT* peuvent appartenir à quatre grandes catégories différentes : M (Modalités), A (Actions), R (Références) et P (Propriétés). Nous nommons ces catégories des *classes*. Les classes se distinguent tant d'un point de vue sémantique que par leur fonctionnement en termes d'association d'entités les unes avec les autres. On nomme $\Gamma = \{M, A, R, P\}$ l'ensemble des classes.
Un schème ne pouvant appartenir qu'à une seule classe, on peut partitionner l'espace des identifiants $\mathbb{S}$ en quatre sous-ensembles $\mathbb{S}_M$, $\mathbb{S}_A$, $\mathbb{S}_R$, $\mathbb{S}_P$ tels que :

$$\mathbb{S} = \mathbb{S}_M \cup \mathbb{S}_A \cup \mathbb{S}_R \cup \mathbb{S}_P$$
$$\forall (i, j) \in \Gamma^2, i \neq j, \mathbb{S}_i \cap \mathbb{S}_j = \emptyset$$

Dans la mesure où la classe d'une entité n'apparaît pas explicitement dans sa définition, nous avons recours à la notation suivante permettant de distinguer la classe d'une entité selon la façon dont nous écrirons son nom dans la suite :

- les modalités sont en majuscules : `WAY`, `LIKE`, `WILL`, *etc.*
- les actions sont en minuscules, avec une majuscule pour le début de chaque mot : `Go`, `Modify`, `SayBye`, *etc.*
- les références et propriétés sont en minuscules uniquement, avec éventuellement des tirets pour séparer différents mots : `object`, `name`, `location-absolute-vertical`, *etc.*

Pour chaque classe, nous donnons l'ensemble des identifiants de schèmes associés, tels qu'ils ont été définis suite à l'analyse du corpus. D'un point de vue quantitatif, le langage *DAFT 2.0* se compose de 44 modalités, 66 actions, 3 références et 124 propriétés, soit $|\mathbb{S}| = 237$.
Plusieurs exemples de schèmes avec le détail de leurs différents attributs sont donnés en annexe E.

Les modalités (M). Il s'agit d'éléments structurants de haut niveau, englobant notamment tout ce qui correspond à des actes de dialogues au sens large. On peut les regrouper dans des grandes catégories, comme le montrent les tableaux 5.1 et 5.2 qui les listent, mais ces distinctions établies a posteriori ne sont pas prises en compte au niveau du langage *DAFT* et sont purement classificatoires.

Les actions (A). Ce sont les éléments du langage qui représentent à la fois les verbes d'opération (*e.g.* "modifier", "déplacer", "montrer"...) et de prédication (*e.g.* "être prêt", "être visible"...). Ils sont listés dans les tableaux 5.3 et 5.4, regroupées en catégories thématiques qui, comme pour les modalités, n'ont pas vocation à être exploitées par *DAFT*.

Catégorie	Nom du schème	Glose
Informations structurelles et fonctionnelles	`META`	Schème permettant de conserver une information - ne servant pas à l'imbrication
”	`INFOS`	Informations à propos d'un *sujet*
”	`MEANING`	Signification d'un *élément* d'après *quelqu'un*
”	`VALUE`	La valeur prise par une *propriété* particulière d'un *élément*
”	`FUNCTION`	La fonction réalisant une *action* donnée sur un *élément* situé à un *emplacement* donné et disposant de *propriétés* particulières
”	`ROLE`	Le rôle d'un *élément*, dans le cadre d'un *référentiel* particulier
”	`FUNCTIONING`	Le fonctionnement d'un *élément*
”	`LIMIT`	Limite ou limitation liée à un *élément* ou d'une de ses *propriétés*
Capacités, droits et devoirs	`POSSIBILITY`	La possibilité offerte à *quelqu'un* par une *autorité* de faire une *action* ou d'agir sur un *élément*
”	`KNOWLEDGE`	La (*croyance* | *capacité*), avec un certain *degré de certitude*, de (*quelqu'un* | *quelque chose*) (à réaliser une *action* | au sujet de *quelque chose*)
”	`OBLIGATION`	L'obligation de *quelqu'un* par une *personne* (d'accomplir une *action* | vis-à-vis d'un *élément*)
”	`AUTHORIZATION`	L'autorisation accordée par *quelqu'un* à *quelqu'un d'autre* d'accomplir une *action*
”	`WILL`	La volonté de *quelqu'un* (de voir une *action* accomplie par *quelqu'un* | au sujet d'un *élément*)
”	`PROBABILITY`	La probabilité de *quelque chose* avec un certain *degré*
Identification d'entités	`EXISTENCE`	L'existence d'un *élément* dans un *emplacement* donné
”	`OTHER`	Un *élément* autre que celui considéré
Relations inter-entités	`LINK`	Un lien entre un *élément* et une *propriété*
”	`UNION`	L'union de plusieurs *éléments*
”	`DIFFERENCE`	La/les *N* différence(s) entre un/des *élément(s)* et un/des autres(s) *élément(s)* d'après un *critère*
”	`ORDER`	Un ensemble d'*actions* à accomplir dans le cadre d'un plan, la dernière correspondant au but final

Tableau 5.1 Liste des 44 schèmes de type Modalité de *DAFT 2.0*

Catégorie	Nom du schème	Glose
Sentiments	LIKE	Une *personne* aime (*quelqu'un* | *quelque chose*)
”	FEAR	Une *personne* craint (*quelqu'un* | *quelque chose*)
”	BOTHER	Une *personne* est ennuyée par *quelque chose* d'embarrassant
”	DOUBT	Une *personne* doute au sujet de *quelque chose*
”	SURPRISE	Une *personne* est surprise par *quelque chose*
”	REGRET	Une *personne* a des regrets au sujet de *quelque chose*
”	HAPPY	Une *personne* est heureuse de *quelque chose*
”	SAD	Une *personne* est malheureuse de *quelque chose*
Circonstancielles	WAY	La façon d'atteindre un *but*
”	EFFECT	L'effet d'(une *action* | un *élément*) est une *action* donnée
”	REASON	La cause d'(une *action* | un *élément*)
”	REFERENTIAL	Le lien par rapport à un *contexte* abstrait (pas un lieu)
”	PLACE	L'emplacement (d'un *élément* | où exécuter une *action*)
”	INSTANT	Le moment où (est effectuée une *action* | un *événement* a lieu)
”	CARDINAL	Le nombre d'*éléments* dans un *emplacement* donné
Assistance	PROBLEM	Quelque chose ne se passe pas comme prévu : une *personne* a un problème avec un *élément* et/ou pour réaliser une *action*
”	MISTAKE	Une *personne* est responsable d'une erreur dans l'emploi d'un *élément* ou la réalisation d'une *action*
”	HELP	Une *personne* requiert l'aide d'une autre *personne* (à propos d'un *élément* | pour réaliser une *action*) s'inscrivant dans un *but* général
Autres	TELL	Une *personne* s'adresse à un *interlocuteur* pour lui quelque chose sur (un *élément* | une *action*)
”	PING	On vérifie la présence d'une *personne* (phatique)
”	SELECTION	On selectionne certains éléments d'après un *critère*
Marqueurs	CHECK	Marqueur de vérification : qu'une action a bien été effectuée, que les deux interlocuteurs sont bien d'accord ou qu'un objet est bien dans l'état attendu.
”	ASK	Marqueur d'interrogation : donne une valeur interrogative à son contenu propositionnel
”	NEG	Marqueur de négation : inverse la valeur de son contenu propositionnel

Tableau 5.2 Liste des 44 schèmes de type Modalité de *DAFT 2.0* – suite

Catégorie	Nom du schème	Glose
Générique	`Act`	Action générique
”	`Try`	Essayer de faire agir un *élément* ou d'effectuer une *action*
Communication	`Contact`	Contacter une *personne*
”	`Dismiss`	Congédier une *personne*
”	`Give`	Donner un *objet* à une *personne*, de la part d'une autre *personne*
”	`Greet`	Accueillir une *personne* dans un *lieu*
”	`Interact`	Interagir avec *quelqu'un*
”	`Listen`	Ecouter un *locuteur*
”	`Recommend`	Recommander un *élément* ou une *personne*
”	`Repeat`	Recommencer une *action* liée à un *élément*
”	`SayBye`	Dire au revoir à une *personne*
”	`SayHello`	Dire bonjour à une *personne*
Réaction	`Accept`	Accepter une *situation* ou l'exécution d'une *action*
”	`Cancel`	Annuler une *action* précédente
Opération	`Activate`	Activer un *élément*
”	`Execute`	Effectuer une *action*
”	`Operate`	Actionner un *élément* d'une certaine *manière*, par un *actionneur*
”	`Use`	Utiliser un *élément* pour faire quelque chose
Processus dynamique	`Become`	Devenir *quelqu'un* ou *quelque chose*
”	`Count`	Compter d'une certaine *manière*
”	`End`	Finir (à interpréter en fonction du contexte)
”	`Happen`	Se dérouler, à un *instant* donné
”	`Pause`	Mettre en pause une *action* ou un *processus*
”	`Restart`	Redémarrer un *élément*
”	`Resume`	Reprendre une *action*
”	`Start`	Démarrer un *élément*
”	`Stop`	Arrêter (de faire une *action* | un *élément*) lorsque se produit un *événement* donné, et pour une certaine *durée*
”	`Update`	Mettre à jour un *élément* par *quelqu'un*
”	`Quit`	Quitter un *élément*
Données statiques	`Create`	Créer un *élément*
”	`Modify`	Modifier un *élément* ou une *propriété* de celui-ci, d'une certaine *manière*, en lui donnant une *valeur*
”	`Recreate`	Recréer un *élément*
”	`Load`	Charger un *élément*
”	`Save`	Enregistrer un *élément* dans un *emplacement*

Tableau 5.3 Liste des 66 schèmes de type Action de *DAFT 2.0*

Catégorie	Nom du schème	Glose
Ensembles de données	`Belong`	Faire partie d'un *conteneur*
”	`Choose`	Choisir un *objet*
”	`Join`	Adhérer à *quelque chose*, pour une *personne*
”	`Add`	Ajouter un *élément* dans un *emplacement* donné, d'une certaine *manière*
”	`Delete`	Effacer un *élément* d'une certaine *manière*
”	`Replace`	Remplacer un *élément* par un autre *élément*
”	`Swap`	Intervertir un *élément* avec un autre *élément*
”	`Sort`	Trier un *élément* d'une certaine *manière*
”	`Split`	Diviser un *groupe d'éléments* d'une certaine *manière*
Organisation	`Adapt`	Adapter/Ajuster un *élément* d'un certaine *manière*
”	`Gather`	Rassembler des *éléments* ensemble
”	`Link`	Lier un *élément* à un *référentiel*
Surveillance	`Control`	Contrôler (une *propriété* d')un *objet*
”	`Limit`	Limiter un *élément* à une *valeur*, par une *personne*
”	`Exceed`	Dépasser une *valeur*, pour un *objet* ou une de ses *propriétés*
”	`Restore`	Restaurer un *objet* à une *valeur* donnée
Visibilité	`See`	Voir un *élément*, par une *personne*
”	`Show`	Montrer à une *audience*, par un *présenteur*, d'une certaine *manière* et dans un *emplacement* donné
”	`Hide`	Cacher un *élément*
Dynamique spatiale	`Move`	Déplacer un *élément* ou une *personne* d'un *lieu* à un *autre lieu* (ou au sein d'un lieu), d'une certaine *manière*
”	`Go`	Aller à un *emplacement* donné, à un *moment* donné, d'une certaine *manière*
”	`Open`	Ouvrir un *élément*
”	`Close`	Fermer un *élément*
Interface graphique	`Bip`	Bipper
”	`Click`	Cliquer sur un *élément* d'une certaine *manière*
”	`Scroll`	Faire défiler un *élément*
”	`Read`	Lire un *texte* dans un *emplacement* donné
”	`Write`	Écrire le *texte* dans un *emplacement* donné
Jeu	`Play`	Jouer un *coup* (dans un jeu) pour une *personne*, par une autre *personne*, à un *instant* donné et d'une certaine *manière*
”	`Win`	Gagner (dans un jeu) d'une certaine *manière*
”	`Lose`	Perdre (dans un jeu) d'une certaine *manière*
”	`Solve`	Résoudre un *problème*
”	`Cheat`	Tricher d'une certaine *manière*

Tableau 5.4 Liste des 66 schèmes de type Action de *DAFT 2.0* – suite

Les références (R). Elles correspondent généralement aux syntagmes nominaux et supposent qu'il existe un élément concret ou abstrait y correspondant dans le monde réel ou au moins dans l'esprit de l'utilisateur. À partir de l'analyse faite du corpus, nous avons été amenés à distinguer trois sous-catégories :

1. les références d'**objet** (`object`) : qui correspondent aux éléments existants dans le monde réel (*e.g.* "la souris") ou dans l'environnement virtuel défini par l'application assistée (*e.g.* "le bouton rouge").
2. les références de **concept** (`concept`) : qui, par opposition aux objets, réfèrent à des éléments non concrets du monde comme des idées (*e.g.* "la vitesse", "le déplacement", "ce que je veux faire"...) et sont donc manipulées de manière sensiblement différentes. Par exemple, elles ne sont pas concernées par les mêmes verbes d'action.
3. les références de **personne** (`person`) : bien qu'elles puissent être vues comme un sous-cas des objets, puisqu'il s'agit également d'entités existant dans le monde physique, leurs possibilités d'interaction avec le monde sont généralement différentes, étant davantage des acteurs à l'origine des actions plutôt que des objets les subissant. L'agent est considéré comme une personne. Notons d'ailleurs que dans plus de 99% des cas, une entité étant soit l'utilisateur lui-même, soit l'agent, on se contente de modéliser trois types de personnes : l'agent/le système (`id="system"`), l'utilisateur (`id="user"`) et les autres (`id="other"`).

Les schèmes utilisés pour ces trois références sont de la même forme et sont constitués de quatre attributs :

- `identifier` qui correspond à un identifiant textuel de l'élément parmi une liste de valeurs possibles (*e.g.* "agent", "utilisateur", "application"...),
- `property*`[2] qui est un attribut multiple listant, sans ordre particulier, toutes les propriétés (au sens ci-dessous) associées à la référence en question (*e.g.* "de type bouton", "de couleur rouge", "de petite taille"...),
- `doing` qui définit la référence par l'action qu'elle permet d'effectuer (*e.g.* "l'assistant" est "la personne qui effectue l'action aider", "l'addition" est "le concept correspondant à l'exécution de l'action ajouter", *etc.*),
- `subject-of` qui définit la référence par une action dont elle est l'objet (*e.g.* "l'assisté" est "celui qui reçoit le résultat de l'action aider", *etc.*).

Les propriétés (P). Elles permettent de qualifier les références (et recouvrent donc une bonne part des adjectifs). Nous sommes partis de l'hypothèse, en accord avec les maximes conversationnelles de Grice [1975], que les utilisateurs définissent une référence en fonction d'un ensemble de propriétés (plus ou moins précises) jugées suffisantes pour permettre son identification dans le contexte d'énonciation.
Parmi les propriétés les plus courantes d'une référence, on peut mentionner :

- `type` : son type intrinsèque ("bouton", "compteur", "clavier"...),

[2] On rappelle que l'astérisque $*$ signifie qu'il s'agit d'un attribut à valeurs éventuellement multiples.

- `quantity` : généralement lié au déterminant, et qui correspond au nombre ("le", "un des", "quelques"...),
- `color` : portée par un adjectif, représente la couleur de la référence ("un des boutons *rouges*"),
- `position` : l'emplacement dans l'espace ("le champ texte *en bas de la page*"),
- `name` : ("*Coco* le compteur"),

Les trois types de références mentionnées précédemment prennent donc leur sens en fonction des valeurs associées à leur attribut (principal) `property*` au sein duquel toutes les propriétés définissant la référence sont mises à plat et sans ordre particulier a priori. Autrement dit, nous ne donnons la priorité à aucune propriété dans la représentation (*e.g.* la quantification est au même niveau que la couleur ou le type intrinsèque de l'objet lorsqu'on représente la référence objet "le bouton rouge"). Dale & Reiter [1995] se sont penchés sur la question de l'ordre d'application des propriétés au sein de référence mais dans le cas de la génération de texte, mais les travaux les plus proches ici sont à mettre au crédit de Pitel [2004, p.97] dans le cadre du projet antérieur Interviews, et de Landragin [2003] dans sa thèse sur les choix langagiers effectués pour désigner les objets du monde. Ce choix de modélisation et les conséquences qui en découlent seront discutés dans la section 5.4.
D'un point de vue représentationnel, bien que l'on conserve l'homogénéité du formalisme, les propriétés n'ont quant à elles en pratique qu'un seul attribut `val`. Le type associé à cet attribut peut être un intervalle discret ou continu de valeurs, mais ne peut pas être un symbole correspondant à une autre entité ; autrement dit, d'autres entités ne peuvent pas être imbriquées dans des propriétés. Formellement, cela donne donc :

$$(\forall E \in \mathbb{E}, \mathrm{id}[E] \in \mathbb{S}_P) \Rightarrow |E| = 1 \wedge a_1 = \texttt{val} \wedge t_1 \notin \mathbb{T}_E$$

Par exemple, `location-horizontal-absolute` a pour valeur associée un nombre appartenant à l'intervalle $[-1, 1]$ définissant à quel point l'élément référé se situe à gauche ou à droite d'un conteneur :

"au milieu" → `location-absolute-horizontal[val = 0]`
"vers la gauche" → `location-absolute-horizontal[val = -0.4]`
"tout à droite" → `location-absolute-horizontal[val = 1]`

Le `type` d'un élément a quant à lui pour valeur associée un des types référencés, d'après ce qui a pu être répertorié dans le corpus : `window`, `webpage`, `document`, `title`, `menu`, `list`, `label`, `row`, *etc.*

L'intégralité des 124 schèmes de propriétés figure dans le tableau 5.5. Les entités sont organisés selon 3 niveaux de spécialisation, où les propriétés correspondent uniquement aux feuilles de l'arbre ainsi formé. Par exemple, cela signifie que `location`, `location-absolute` ou `speed` ne sont pas des propriétés, contrairement à `location-absolute-horizontal` ou `speed-change`. La liste donnée sur la dernière ligne correspond donc à des propriétés de niveau 1.

Propriété de niveau 1	Niveau 2	Niveau 3
location	absolute	horizontal, vertical, depth, path
"	relative	distance, inclusion, navigation, superposition
time	instant	process
"	duration	long, season, month, weekday, dayperiod
manner	add	
size	length, width, height, thickness	
visual	color, dark, shape, transparent	
device	in, out	
able	acceptable, activable, dialogical, editable, loadable, listenable, movable, operable, restorable, saveable, undoable, visible	
trait	cautious, extraverted, friendly, gender, honest, lost, patient, proud, satisfied, talkative, thankful, upset	
lingua	formalspeak, language	
data	dynamic	
"	datatype	textelement, numberdigit
"	text	font, bold, italic, underlined
"	number	even, positive, real
speed	change	
comparison		
ordinality		
accurate, arbitrary, attending, attribute, available, boring, caring, correct, compatible, competent, difficult, direction, disposition, straight, disturbing, direct, domain, efficient, enabled, encouraging, filled, funny, general, genuine, good, intensity, insulting, infocus, main, mandatory, name, open, ownedby, part, pleasant, private, new, quantity, random, processstatus, significant, safe, true, type, usual, value, working, worthy		

Tableau 5.5 Liste des 124 schèmes de type Propriété de *DAFT 2.0* (ordonnés par catégories)

5.2.2.2 Types : restrictions sur les valeurs

Comme on l'a vu lors de la définition syntaxique du langage, un type est associé à :

1. chaque attribut d'une entité, définissant une restriction sur les valeurs pouvant être associées à l'attribut correspondant. Par exemple, l'attribut `clicker` de `Click` ne peut avoir pour valeur associée qu'une entité ayant pour type *person*.
2. chaque entité elle-même, définissant une restriction sur les attributs d'autres entités auxquels elle peut être associée. Par exemple, une entité `Click` peut devenir la valeur d'un attribut d'une entité dont le type serait une action (*act*).

À chaque type t est associé un ensemble (fini ou non) de valeurs possibles pouvant être associé à un attribut ayant pour type t : on nomme cet ensemble *le domaine de validité* du type t, noté δ_t.
Dans la mesure où il est possible d'associer à un attribut a d'une entité E un ou plusieurs types (*e.g.* le sujet principal de l'action `Move` peut être un objet ou une personne), on peut définir de manière similaire le domaine de validité de l'attribut a de E, noté $\delta_{a[E]}$. Logiquement, ce domaine de validité de l'attribut est défini comme l'union des domaines de validité des types associés à l'attribut, soit pour N types associés : $\delta_{a[E]} = \bigcup_{i=1}^{N} \delta_{t_i}$

Catégories de types. Nous distinguons deux catégories de types : les types élémentaires et les types terminaux. L'ensemble des types $\mathbb{T}$ est donc l'union de l'ensemble des types élémentaires (noté $\mathbb{T}_E$) et de l'ensemble des types terminaux (noté $\mathbb{T}_T$) : $\mathbb{T} = \mathbb{T}_E \cup \mathbb{T}_T$.

Types élémentaires. Les types élémentaires sont organisés sous forme d'un arbre (*cf.* figure 5.1) dont les trois branches principales partent de trois types que l'on nommera les types élémentaires primaires : *act*, *ref* et *ppt* (correspondant respectivement à action, référence et propriété).
L'ensemble des types élémentaires ($\mathbb{T}_E$) peut être formellement défini comme l'union des types élémentaires primaires avec les types de toutes les entité de classe Référence et Propriété, *i.e.* :

$$\begin{aligned}\mathbb{T}_E &= \{\texttt{act}, \texttt{ref}, \texttt{ppt}\} \cup \{\text{type}[E], \forall E \in \mathbb{E} | \text{id}[E] \in (\mathbb{S}_R \cup \mathbb{S}_P)\} \\ &= \{\texttt{act}, \texttt{ref}, \texttt{object}, \texttt{concept}, \texttt{person}, \texttt{ppt}, \texttt{name}, \texttt{color}\ldots\}\end{aligned}$$

Cette définition provient du fait que pour chaque schème de Référence et de Propriété, il existe un type élémentaire portant le même nom que l'identifiant, et d'ailleurs, ce type élémentaire en question est le type de toute entité générée à partir de ce schème. D'un point de vue notationnel, un type élémentaire sera donc toujours écrit en minuscules (puisque, pour rappel, c'est le cas des identifiants des entités de Référence et de Propriété).

Le domaine de validité associé à un type élémentaire $t \in \mathbb{T}_E$ est défini par l'ensemble des

entités dont le type associé est t ou un descendant de t dans l'arbre des types :

$$\forall t \in \mathbb{T}_E, E \in \delta_t \Leftrightarrow (\text{type}[E] = t \vee t \in \text{ancestors}[\text{type}[E]])$$

Exemples de domaines de validité de types élémentaires :

$\delta_{ref} = \{$*ref*, *person*, *object*, *concept*$\}$

$\delta_{location-absolute} = \{$*location-absolute*, *location-absolute-path*, *location-absolute-depth*, *location-absolute-vertical*, *location-absolute-horizontal*$\}$

$\delta_{color} = \{$*color*$\}$

→ **act**
→ **ref**
  → *person*
  → *object*
  → *concept*
→ **ppt**
  → *comparison, ordinality, type, ...*
  → *location*
    → *location-absolute*
    ... (*cf.* tableau 5.5 pour la liste détaillée des Propriétés correspondant à un type)

Figure 5.1 Arbre réduit des 130 types élémentaires de *DAFT 2.0* (en gras, les types élémentaires primaires)

Types terminaux. On peut distinguer quatre types terminaux, chacun étant lié à un domaine de validité différent :

1. un invervalle continu de valeurs, noté $[borne_{inf}, borne_{sup}]$:

$$v \in \delta_t \Leftrightarrow v \in [borne_{inf}, borne_{sup}]$$

2. un ensemble fini de valeurs, noté $[\![val_1, val_2, \ldots, val_n]\!]$:

$$v \in \delta_t \Leftrightarrow v \in \{val_1, val_2, \ldots, val_n\}$$

3. un booléen, qui n'est en fait qu'un ensemble fini particulier :

$$v \in \delta_t \Leftrightarrow v \in \{True, False\}$$

4. une chaîne de caractères :

$$v \in \delta_t \Leftrightarrow v = \text{“}[\hat{}\text{”}] * \text{”}$$

(avec la notation classique pour les expressions régulières)

D'après cette définition, $\mathbb{T}_T$ est clairement un ensemble infini (contrairement à $\mathbb{T}_E$).

Exemples de domaines de validité de types terminaux :
$\delta_{val[location-absolute-horizontal[...]} = [-1, 1]$ (où -1 signifie "gauche" et 1 signifie "droite")
$\delta_{val[color[...]]} = \{$"black", "white", "gray", "red", "blue", "green", "yellow", ... $\}$
$\delta_{val[visible[...]} = \{True, False\}$
$\delta_{val[name[...]} = \{.*\}$

Types associés à un attribut. Un attribut peut avoir comme type associé n'importe quel type $t \in \mathbb{T}$: un type élémentaire, un ensemble de types élémentaires ou un type terminal.

Exemples de types associés à des attributs :
type[`val[location-absolute-horizontal[...]]`] $= [-1, 1]$
type[`val[color[...]]`] $= [\![$ "black", "white", "gray", "red", "blue", "green", "yellow" $]\!]$
type[`val[visible[...]]`] $= Boolean$
type[`val[name[...]]`] $= String$
type[`clicker[Click[...]]`] $= \{$`person`$\}$
type[`about[ASK[...]]`] $= \{$`act`, `ref`$\}$

Types associés à une entité. Le type associé à une entité est nécessairement un type élémentaire ou un ensemble de types élémentaires. Les valeurs possibles pour le type associé à une entité dépend de la classe de cette entité :

$$\forall E \in \mathbb{E}, \mathrm{id}[E] \in \mathbb{S}_M \Rightarrow \mathrm{type}[E] \in \{\texttt{act}, \texttt{ref}, \texttt{object}, \texttt{concept}, \texttt{person}\}$$
$$\forall E \in \mathbb{E}, \mathrm{id}[E] \in \mathbb{S}_A \Rightarrow \mathrm{type}[E] = \texttt{act}$$
$$\forall E \in \mathbb{E}, \mathrm{id}[E] \in (\mathbb{S}_R \cup \mathbb{S}_P) \Rightarrow \mathrm{type}[E] = \mathrm{id}[E]$$

Une entité E peut également se voir associer un *type transparent*, noté $\S i$ tel que $i \in [\![1, |E|]\!]$, ce qui signifie que le type associé à l'entité est le même que celui de la valeur associée au i^{e} attribut de E. Cela permet donc d'avoir une forme de typage dynamique qui peut être utile pour certaines modalités (*cf.* le cas `ASK` dans la requête d'exemple R_{ex_2} donnée plus loin). Toutefois, pour faire sens cela suppose que ce i^{e} attribut :

1. puisse avoir une valeur associée de plusieurs types différents : autrement cela revient à dire que le type associé à E est fixe ;
2. soit forcément rempli : sinon le type associé à E ne peut pas être déterminé.

Exemples de types associés à des entités :
type[KNOWLEDGE[...]] = {*act*, *concept*}
type[MEANING[...]] = {*concept*}
type[Click[...]] = {*act*}
type[person[...]] = {*person*}
type[color[...]] = {*color*}
type[ASK[...]] = {§1}

5.2.2.3 Requêtes : association d'entités

Principes. Une requête d'utilisateur est représentée en *DAFT 2.0* sous la forme d'un arbre d'entités. Normalement, si une requête est bien formée (c'est-à-dire en pratique, si le langage permet de l'exprimer et qu'elle a pu être correctement construite de manière automatique), il y a une entité unique formant la racine de la requête. Toutes les autres entités sont rattachées à l'entité racine, de manière directe ou à travers d'autres entités elles-mêmes directement rattachées à celle-ci.

Typiquement, les modalités tendent à être les entités situées le plus à l'extérieur de la requête, auxquelles se rattachent les actions et références comme valeurs de leurs attributs. Les actions prennent elles-mêmes le plus souvent des références comme valeurs associées à leurs attributs, tandis que ces références sont principalement définies par l'ensemble des propriétés qui leurs sont attachées. Par conséquent, la structure typique d'une requête *DAFT 2.0* est de la forme :

$$M_1[\dots M_n[A_1[R_1[P_1[\dots], \dots, P_q[\dots]], \dots, R_p[\dots]], \dots, A_m[\dots]], \dots]$$

où $M_1 - M_n$ sont des modalités, $A_1 - A_m$ sont des actions, $R_1 - R_p$ sont des références et $P_1 - P_q$ sont des propriétés.

Exemples. Pour illustrer le principe de construction d'une requête, nous proposons de considérer trois exemples qui réfèrent à une même action (cliquer le bouton rouge), utilisée dans différentes tournures de phrases, qui correspondent ainsi à des activités conversationnelles différentes (au sens qui sera introduit en 7.2) : le premier exemple R_{ex_1} est une requête de contrôle, le deuxième R_{ex_2} est une requête d'assistance directe et enfin le troisième R_{ex_3} correspond à une requête d'assistance indirecte.

Exemple de représentation d'une requête de contrôle R_{ex_1} : "Clique le bouton rouge"

```
{act} : Click[
  {person}   : clicker     = ∅,
  {object}   : clicked     = object[
    {ppt}*     : property*   = {
      {quantity} : quantity[[...]] : val = 1 ]
      {color}    : color[[...]] : val = "red" ]
      {type}     : type[[...]] : val = "button" ]
    },
    {act}      : doing       = ∅,
    {act}      : subject-of  = ∅
  ],
  {manner}  : manner      = ∅
]
```

Six notions sémantiques apparaissent dans cette requête :

1. l'ordre de l'utilisateur pour que l'agent accomplisse une action,
2. l'action de cliquer sur un élément de l'interface graphique,
3. l'objet sur lequel est exécuté le clic,
4. le fait que l'objet cliqué est unique,
5. le fait que l'objet cliqué est de couleur rouge,
6. le fait l'objet cliqué correspond est de type bouton.

Nous allons donc détailler la prise en compte de chacune de ces notions :

1 et 2 : On a déjà vu dans l'exemple donné dans la section 5.2.1 l'entité correspondant au verbe "cliquer" : pour représenter R_{ex_1}, il suffit donc d'associer une valeur à l'attribut `clicked`. Pour montrer qu'il s'agit d'un ordre, on pourrait utiliser une modalité spéciale (par exemple EXEC) à associer à `Click`. Toutefois, nous avons choisi de considérer qu'une action à la racine d'une requête a implicitement valeur d'ordre. La requête `Click[...]` sera donc interprétée par l'agent comme "effectue l'action 'cliquer' sur...".

3 : Le schème utilisé pour représenter une référence existant dans le monde physique de l'application est `object`. On crée donc une entité :

```
{act} : object[
  [[ "window", ...]]   : identifier   = ∅,
  {ppt}*               : property*    = ∅,
  {act}                : doing        = ∅,
  {act}                : subject-of   = ∅
]
```

L'attribut `identifier` est associé à un type terminal : pour alléger la notation des exemples, par la suite, nous ne donnerons généralement pas la liste de toutes les valeurs possibles contenues dans une énumération et abrégerons donc la notation par [[...]].
L'attribut `property` (*ppt*) peut se voir associé plusieurs valeurs (*cf.* la présence de l'astérisque en suffixe).

4, 5 et 6 : Trois entités de classe Propriété sont nécessaires afin de représenter le fait qu'il s'agit d'une référence qui est unique ("le" est un défini – *cf.* [Corblin, 1987]), ayant une couleur particulière ("rouge") et dont la nature est d'être un composant d'application particulier ("bouton") :

```
{quantity} : quantity[[...]] : val = 1 ]
{color} : color[[...]] : val = "red" ]
{type} : type[[...]] : val = "button" ]
```

On retrouve d'ailleurs là les principaux composants présents dans un énoncé en langue naturelle d'après Landragin *et al.* [2002].

Exemple de représentation d'une requête d'assistance directe R_{ex_2} : "Puis-je cliquer le bouton rouge ?"

```
{concept}  : ASK[
  {act,ref}   : about       = POSSIBILITY [
    {person}    : of          = person[[...]] : identifier = "user"]
    {act}       : concerning  = Click [
      {person}    : clicker     = ∅,
      {object}    : clicked     = object[
        {ppt}*      : property*   = {
          {quantity}  : quantity[[...]] : val = 1 ],
          {color}     : color[[...]] : val = "red" ],
          {type}      : type[[...]] : val = "button" ]
}]]]]
```

Note : pour alléger l'écriture, certains des attributs n'ayant pas de valeur associée (*i.e.* dont la valeur associée est ∅) ne sont pas représentés ici.

Par rapport à la requête R_{ex_1}, cette requête R_{ex_2} met en jeu trois notions sémantiques supplémentaires :

1. la personne accomplissant l'action est l'**utilisateur**,
2. on considère la **possibilité** d'exécuter cette action,
3. la requête n'est plus un ordre (par défaut) mais une **interrogation**.

Pour le premier point, il est nécessaire d'associer une valeur à l'attribut `clicker`, qui doit être une entité de personne dont la valeur terminale de son identifiant correspond à l'utilisateur :

```
{person} : person[
  [[ ...]]    : identifier   = "user",
  {ppt}*      : property*    = ∅,
  {act}       : doing        = ∅,
  {act}       : subject-of   = ∅
]
```

En ce qui concerne le deuxième point, on fait appel à une entité de modalité `POSSIBILITY`, définie comme suit :

```
{concept} : POSSIBILITY[
  {concept}           : of          = ∅,
  {person}*           : offered-by  = ∅,
  {act, object, concept} : concerning  = ∅
]
```

Enfin, le troisième point signifie qu'il est nécessaire d'encapsuler toute la requête avec un marqueur d'interrogation, ce qui est le rôle de la modalité ASK :

{§1} : ASK[{act, ref} : about = ∅]

Pour rappel, le type transparent *§1* signifie que l'entité ASK aura un type qui sera égal au type associé à son premier (et ici unique) attribut. Le type associé à POSSIBILITY étant *concept*, il appartient au domaine de validité de l'attribut about de l'entité ASK qui peut donc encapsuler la requête. Le type associé à l'entité ASK devient alors également *concept*, ce qui rend l'utilisation de la modalité ASK "transparente" : en effet, la requête peut être imbriquée au sein d'une autre de la même manière que si ASK n'était pas présent, comme on va le voir dans l'exemple suivant (avec NEG, dont le fonctionnement est identique).

Exemple de représentation d'une requête d'assistance indirecte R_{ex_3} : "Dommage de ne pas pouvoir cliquer le bouton rouge"

```
{concept}  : REGRET[
  {concept}  : about  = NEG [
    {concept} : about   = POSSIBILITY [
      {act}     : concerning  = Click [
        {object} : clicked     = object[
          {ppt}*   : property*   = {
            {quantity} quantity[[...]] : val = 1 ],
            {color}  : color[[...]] : val = "red" ],
            {type}   : type[[...]] : val = "button" ]
}]]]]
```

Note : les attributs sans valeur associée (*i.e.* valant ∅) ne sont pas représentés ici.

Par rapport à R_{ex_1}, on a une nouvelle fois dans cette requête R_{ex_3} trois notions sémantiques supplémentaires :

1. on considère la **possibilité** d'exécuter cette action (*cf.* le deuxième exemple),
2. cette possibilité est **niée**,
3. l'idée selon laquelle cette situation d'ensemble est **déplorable**.

Nous ne revenons pas sur le traitement du premier point qui nécessite l'utilisation d'une entité POSSIBILITY. Pour le deuxième, on peut faire appel à un marqueur de négation, NEG, qui fonctionne exactement comme ASK dans R_{ex_2}, à savoir avec un type associé transparent :

{§1} : NEG[{act, ref} : about = ∅]

Enfin, en ce qui concerne le troisième point, il suffit d'inclure l'ensemble de la requête dans une entité REGRET (*cf.* tableau 5.2) au format suivant :

```
{concept} : REGRET [
  {person}              : from          = ∅,
  {act, person, concept}: about         = ∅
]
```

À noter qu'il s'agit d'un regret au sens large : on ne fait pas à ce niveau la différence entre remords (un acte a été accompli que l'on aurait préféré de pas faire) et regret (on aurait préféré accomplir l'acte en question). Bien que pertinente, celle-ci sera éventuellement à prendre en compte par l'agent rationnel, qui sera lui en mesure de vérifier si l'action ainsi modalisée a été accomplie par le passé.

5.3 Évaluation de *DAFT 2.0* par rapport au corpus *Daft*

Afin d'évaluer l'adéquation du langage formel défini dans la section 5.2 aux requêtes d'utilisateurs qu'il doit modéliser au sein du système `DAFT 2.0`, il est nécessaire de le confronter au corpus de requêtes, afin de :

1. déterminer la couverture offerte par ce langage, *i.e.* la part de requêtes pouvant être correctement et entièrement écrites sous forme formelle en *DAFT 2.0*,
2. mener une étude fréquentielle d'utilisation des différents schèmes.

5.3.1 Pré-évaluation à partir d'une annotation manuelle

Avant de concevoir le module `DIG` (décrit dans le chapitre 6) qui permet d'automatiser la génération de requêtes en *DAFT*, nous avons essayé de l'appliquer à une partie du corpus recueillie ($Daft_r$) en annotant manuellement les requêtes.
Nous allons considérer dans les sous-sections suivantes les deux sous-corpus de requêtes, $Daft_{sub_1}$ et $Daft_{sub_2}$, tels que définis en section 3.3.3.2.

5.3.1.1 Estimation de la robustesse de la structure du langage

Annotation initiale. Étant donné cette structure de langage, nous avons procédé à l'annotation du sous-ensemble de requêtes $Daft_{sub_1}$. Les schèmes nécessaires étaient définis au fur et à mesure des besoins, avec pour seule contrainte le respect de la structure fixée, de manière à voir les contraintes induites par celle-ci.

Correction de l'annotation. Nous avons ensuite effectué une correction des phrases initialement annotées en deux temps. Tout d'abord, nous avons cherché à "stabiliser" l'ensemble des schèmes utilisés en ayant un seul schème (ou structure de schèmes) par structure sémantique : par exemple de manière à ne pas représenter le sens de "cacher" une fois par une structure

de la forme `Modify[property=visibility[val=False]]` et une autre fois par une action unique comme `Hide[]`.

Ensuite, nous avons repris individuellement chacun des schèmes définis en considérant chacune de ses occurrences, de manière à s'assurer de disposer pour chacun d'une structure unique, aussi simple que possible mais non ambigüe. Par exemple, le schème `Control`, qui représente le fait de gérer/contrôler quelque chose, peut s'appliquer à :

- un élément concret : "je ne vois comment gérer la carte du site"
- un concept à valeur de propriété : "qu'est-ce qui contrôle la vitesse ?"

Dans les deux cas, lors de la première annotation, "carte du site" et "vitesse" avaient été associés à un attribut unique "of". Nous avons préféré le remplacer par deux attributs séparés ("object" et "property"), de manière à pouvoir représenter la partie interne d'une requête de type "qu'est-ce qui contrôle la vitesse du compteur ?" comme `Control[property = speed[], object = object[doing = Count[]]]`.

Résultats. À l'issue de ce travail, nous avons ainsi exhibé 39 modalités et 62 actions, soit un sous-ensemble relativement important des 44 modalités et 66 actions définies dans les tableaux 5.1,5.2, 5.3 et 5.4 (*cf.* la section 5.3.1.2 pour les schèmes restants). De plus, il a été possible de modéliser la grande majorité des requêtes de contrôle et d'assistance (*cf.* le tableau 5.6).

Sous-corpus	Contrôle	Assistance directe	Assistance indirecte
Couverture	95,8%	98,3%	92,2%

Tableau 5.6 Couverture d'un sous-ensemble du corpus $Daft_r$ par le langage *DAFT 2.0* en fonction des activités conversationnelles

Analyse. Si l'on s'intéresse de plus près aux phrases qui n'ont pas pu être modélisées, on peut distinguer certains cas typiques :

- les requêtes de contrôle sans verbe, pour lesquelles la modélisation est réalisable en principe, mais qui supposerait pour être correcte de connaître le verbe associé "par défaut" à la référence.
  Exemples de requêtes : "carte du site", "description", "membres"...
- les requêtes de contrôle sans référence, où le problème est similaire à celui mentionné ci-dessus puisqu'il faudrait ici connaître soit la référence associée par défaut, soit la référence ayant actuellement le focus de l'utilisateur.
  Exemples de requêtes : "go", "fais le"...
- les requêtes de contrôle mettant en jeu une référence complexe. *Exemples de requêtes :* "parle-moi *même si tu n'as rien à dire*", "exécute *la même chose qu'avant*"...
- les requêtes d'assistance directe au sujet d'une action imprécise (verbe "faire") ou qui sont liées à une requête précédente.

Exemples de requêtes : “que fais-tu ?”, “pourquoi tu le fais pas ?”, “si tu ne veux pas faire ce que je te dis, très bien, mais dis moi quoi faire”, ...

Dans tous les cas, une modélisation n'est donc pas impossible d'un point de vue strictement structurel (on peut à chaque fois construire une requête respectant la syntaxe du langage *DAFT*) mais elle serait en fait simplement partielle en l'absence d'informations complémentaires.
On peut en conclure que la syntaxe du langage *DAFT 2.0* est suffisamment robuste et adaptable pour nos objectifs.

5.3.1.2 Estimation de la couverture du langage

Une fois admise l'adéquation de la structure du langage, la seconde question qui se pose est celle de la couverture. Autrement dit, il faut que le langage que l'on définisse soit suffisamment générique pour pouvoir considérer que l'ensemble des schèmes donnés est suffisant pour modéliser toute requête que l'agent assistant sera amené à traiter, c'est-à-dire des requêtes nouvelles (non présentes dans le corpus *Daft*).

Première annotation. Afin d'évaluer ceci, nous avons commencé par procéder à une annotation d'un nouveau sous-ensemble de requêtes, $Daft_{sub_2}$. Lors de cette formalisation manuelle des requêtes, seuls pouvaient être utilisés les schèmes définis en 5.3.1.1.

Seconde annotation. Dans un second temps, nous avons repris les phrases de $Daft_{sub_2}$ qui n'avaient pas pu être formalisées lors de l'annotation initiale, en levant cette fois-ci la contrainte de non ajout de nouveaux schèmes.

Résultats. Lors de la seconde annotation, nous avons pu annoter une vingtaine de phrases supplémentaires en ajoutant 4 actions (`Choose`, `Execute`, `Interact` et `Link`) et 2 modalités (`REFERENTIAL` et `UNION`). On obtient ainsi des taux de phrases annotés similaires à ceux obtenus dans la section 5.3.1.1 (*cf.* tableau 5.6).

Analyse. Nous avons ainsi mis en évidence qu'une augmentation de 100% de la taille du corpus considéré pour définir les schèmes de *DAFT 2.0* ($Daft_{sub_1}$ puis $Daft_{sub_1} \cup Daft_{sub_2}$) n'entraîne qu'une augmentation de 5% des schèmes requis. Avec un corpus dix fois plus important ($Daft_r$), on peut donc raisonnablement penser être en mesure d'obtenir un langage couvrant bien le champ de l'assistance aux usagers novices, au vocabulaire spécifique à l'application près qui devrait pouvoir être ajouté par le concepteur de l'application (par exemple, “compter” n'est pas forcément en soi une action générique, mais elle est saillante dans le cas de l'application “Coco le compteur” definie dans la table 3.3).

5.4 Discussion : choix de modélisation et conséquences

5.4.1 Domaine de validité des propriétés : éléments individuels ou larges ensembles ?

Il est possible d'imaginer différents niveaux de granularité dans la définition des schèmes de classe Propriété, c'est-à-dire différentes partitions de l'espace des propriétés. Considérons par exemple la représentation de la couleur d'une référence. Pour simplifier, nous admettons que seules trois couleurs sont possibles (rouge, vert, bleu). Au moins trois manières de représenter cette notion sont disponibles et résumées dans le tableau 5.7.

Représentation	1	2	3
Nombre de propriétés	1 : `color`	3 : `red`, `blue`, `green`	3 : `red`, `blue`, `green`
Domaine de validité	δ_{color} = {"red", "blue", "green"}	Pour chacune : δ = {True, False}	Pour chacune : $\delta = [\![0, 1]\!]$ (*e.g.* couleurs 16 ou 32 bits)
Pouvoir expressif	Faible	Moyen	Fort
Informations implicites sur le domaine	Importante	Faible	Faible

Tableau 5.7 Trois représentations possibles pour la couleur d'une référence

En termes de pouvoir expressif, c'est-à-dire la capacité à représenter des éléments complexes du monde, nous avons $1 < 2 < 3$, dans la mesure où :

- dans le cas 1, un objet a une et une seule couleur,
- dans le cas 2, un objet est ou n'est pas de chacune des couleurs possibles,
- dans le cas 3, un objet peut être plus ou moins de chaque couleur.

Au contraire, en termes de connaissances du domaine ou de capacité de généralisation, en l'absence d'informations supplémentaires, $1 > 2 = 3$ puisqu'il est alors implicite que le fait qu'un objet soit rouge l'empêche d'être également bleu et vert.
Par conséquent, le choix d'un sur-ensemble (cas 1) ou de plusieurs sous-ensembles (cas 2 et 3) implique que l'on facilitera soit l'expressivité, soit les capacités de déduction de l'agent rationnel. En effet, l'arbre des types n'est pas seulement utile pour la représentation des requêtes, mais peut aussi être utilisé pour raisonner sur le monde.

En pratique, plutôt que d'opter de manière arbitraire pour l'une ou l'autre des solutions, nous nous laissons donc guider par l'utilisation que l'agent rationnel $\mathtt{A}_R$ est amené à faire de ces informations (*cf.* chapitre 8). En effet, ce n'est pas tant la représentation interne que l'agent se fait de la requête que la manière dont il réagira à celle-ci qui compte. En règle générale, nous

avons choisi d'utiliser des propriétés individuelles, sauf quand il apparaît que ces propriétés sont réellement mutuellement exclusives. C'est par exemple le cas de la propriété `type` : si un élément est un bouton, il ne peut pas être aussi une fenêtre. Cela permet de fournir un peu de "sens commun" à l'agent, puisqu'il n'est alors pas nécessaire d'encoder explicitement dans l'agent rationnel des règles du type "un bouton n'est pas une fenêtre". Néanmoins, il est clair que chacun de ces choix amène à une restriction de la vision du monde que l'agent peut avoir.

5.4.2 Actions génériques ou spécifiques ?

Dans le même esprit que ce qui a été vu pour les propriétés dans la section 5.4.1, on peut se demander si pour atteindre une certaine généricité du langage, il n'est pas préférable de restreindre au maximum le nombre de schèmes disponibles pour les actions sur le monde, en optant plutôt pour des actions génériques dotées de nombreux attributs. Le tableau 5.8 montre le type de représentation possible pour quelques verbes d'actions. En optant pour la

Verbe d'action	Représentation spécifique	Représentation générique
Cacher	`Hide[...]`	`Modify[ppt*={visibility[val=False]}]`
Donner	`Give[...]`	`Modify[ppt*={owner[]}]`
Aller	`Go[...]`	`Modify[ppt*={location[]}]`
Complexifier	`Complexify[...]`	`Modify[ppt*={difficult[val=+]}]`

Tableau 5.8 Représentations générique et spécifique de quelques verbes d'action

représentation générique, on peut diminuer de manière très conséquente le nombre de entités de classe Action nécessaire dans le langage (4 contre 1 dans le cas des verbes du tableau 5.8) et définir une sémantique très générique. Toutefois, on perd alors la structure fonctionnelle spécifique à chaque verbe. C'est-à-dire que dans les exemples considérés ici, l'entité `Modify` devrait disposer d'un certain nombre d'attributs vagues qui n'auraient pas le même sens en fonction des propriétés liées à `Modify`. Ainsi, dans les requêtes de contrôle "cache la fenêtre", "donne la valeur" et "va à la page d'accueil", "fenêtre", "valeur" et "page d'accueil" jouant le même rôle d'objet sur lequel s'applique l'action seraient à rattacher à un même attribut de `Modify`. De plus, on ne pourrait pas préciser le type de cet attribut qui devrait alors correspondre soit à un *object* ("fenêtre"), soit à un *concept* ("valeur") ou encore à un emplacement *location* ("la page d'accueil") : on perd donc une partie de l'intérêt du typage des attributs comme aide à la construction d'une requête.
Une solution alternative pourrait être de faire une construction de la requête en deux étapes, en passant d'abord par la représentation spécifique pour permettre l'utilisation des types, puis une conversion de la représentation spécifique vers la représentation générique équivalente. Toutefois, c'est encore une fois le contexte dans lequel s'inscrit l'utilisation de ce langage qui permet de choisir : nous ne nous intéressons en effet pas tant à la représentation sémantique formelle pour elle-même qu'à la manière dont elle sera utilisée par l'agent rationnel par la suite. La question à se poser est donc de savoir si une représentation plus générique permet

ou non de faciliter la synthèse de réactions associées au sein des heuristiques de l'agent. Or si l'on veut réagir à "donner" et "cacher" différemment, il importe peu de savoir si la réaction est associée à une requête écrite de manière spécifique ou générique : c'est donc un faux problème. Afin de faciliter la lisibilité des actions qui nous intéressent, nous avons donc préféré de manière générale une représentation spécifique, ce qui donne un langage plutôt riche en actions mais plus clair à comprendre.

5.4.3 Gestion de la position spatiale des références

Pour rappel, nous avons choisi de considérer toutes les propriétés comme étant d'importance égale par rapport à une référence. À ce titre, la position spatiale doit être agrégée dans les valeurs de l'attribut à valeurs multiples `property*`. Toutefois, les expressions spatiales présentent certaines spécificités nous obligeant à les traiter de manière particulière. On peut ainsi distinguer deux types d'expressions de position :

- les positions absolues : ici, en haut, à droite, *etc.*
- les positions relatives : à droite de X, dans la page, sur le disque, en dessous de Y, *etc.*

Si les positions absolues peuvent être représentées de la même manière que n'importe quelles autres propriétés, il n'en va pas de même des positions relatives : en effet, créer des entités différentes pour "en-dessous du bouton", "à droite du bouton" ou "sous le bouton" reviendrait à nier le fait qu'il s'agit d'une référence à un même bouton. De plus, on devrait alors avoir une liste d'emplacements extrêmement importante. Nous préférons donc définir les positions relatives comme étant la combinaison d'éléments de position ("dans", "sur", "en-dessous de") avec d'autres références. À cet effet, nous avons donc défini une modalité particulière, `PLACE`, dont le type associé est *`location-relative`*, et ayant un attribut prenant pour valeur l'élément de référence par rapport auquel on se situe (voir l'exemple de la figure 5.2 – en anticipant sur le mode de représentation des requêtes *DAFT 2.0* construites par l'outil `DIG`, décrit en 6.1.2).

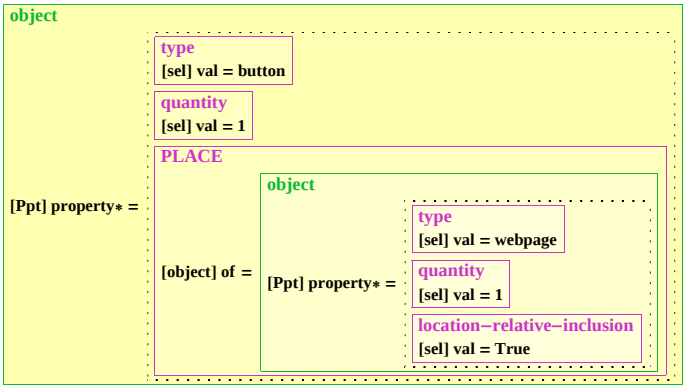


Figure 5.2 Représentation graphique de la requête "le bouton dans la page"

5.5 Conclusion

Nous avons introduit dans ce chapitre la syntaxe et la sémantique du langage *DAFT 2.0*, défini à partir du corpus *Daft*. Ce langage repose sur un ensemble $\mathbb{S}$ de 237 schèmes qui peuvent se voir associer des valeurs et être combinés entre eux de manière à modéliser les requêtes en langue naturelle pré-analysées par `GRASP`. Ayant montré que ce langage était suffisant pour couvrir la majorité des requêtes d'assistance issues du corpus, il reste maintenant à automatiser leur production, ce qui fait l'objet du chapitre qui suit.

Chapitre 6

Génération automatique de requêtes formelles : DIG

Sommaire

Comme on l'a vu dans le chapitre 4, l'outil GRASP fournit une analyse grammaticale des phrases qu'il prend en entrée, et produit des éléments individuels structurés et annotés individuellement d'un point de vue sémantique. Afin de pouvoir réagir à ces requêtes, il reste

encore à associer un sens à la requête prise dans son ensemble, en utilisant les liens entre les différents éléments et leur sens individuel pour passer à une représentation ayant la forme du langage *DAFT* introduit dans le chapitre 5. C'est le rôle de l'outil `DIG` (pour Daft Interpretation Generator), dont nous allons décrire ici le fonctionnement, après avoir brièvement rappelé la forme de ses entrées et sorties.

6.1 Entrées et sorties de `DIG`

6.1.1 Entrées et ressources

Pour produire une représentation sémantique d'une requête utilisateur, `DIG` dispose en entrée de quatre éléments :

1. **La représentation structurée de la requête utilisateur :** produite par `GRASP`, elle correspond formellement à une imbrication de listes (bien qu'on la visualise sous forme de "boîtes"). Idéalement, quand une requête est complètement analysée par `GRASP`, elle est constituée de 3 nœuds : le premier et le dernier nœud marquent les limites de la requêtes et conservent des informations sur l'historique des traitements effectués sur celles-ci, tandis que le nœud central contient l'analyse syntaxico-grammaticale à proprement parler). De même, les nœuds sont tous dotés d'une clé sémantique unique issue de $\mathbb{K}_G$ contenue dans le champ KEYS, ou dans le champ KEYWSD si l'algorithme de WSD (*cf.* section 4.3.2) a dû être appliqué (les autres sens sont donc encore accessibles si nécessaire).
2. **Un ensemble $\mathbb{S}$ de 237 schèmes :** ils définissent les éléments existants dans le langage *DAFT* et posent des contraintes sur la façon dont ceux-ci peuvent être associés les uns aux autres. Ils ont été décrits dans le chapitre 5.
3. **Un ensemble $\mathbb{K}_G$ de 1 294 clés sémantiques :** par rapport à la manière dont elles ont été définies dans le chapitre 4 (voir figure 4.4), il est nécessaire de leur adjoindre un sixième attribut afin de lier la clé à une ou plusieurs entité(s) *DAFT*. Cet attribut est décrit dans la section 6.2.1.
4. **Un ensemble de règles de combinaison $\mathbb{R}_{\mathcal{M}}$:** ces règles permettent d'associer entre eux les constituants de la phrase d'un point de vue *sémantique*, de la même manière que les règles de $\mathbb{R}_G$ permettaient de les associer d'un point de vue *grammatical*. Leur syntaxe est décrite en section 6.2.2.1 et leur utilisation pratique au sein du moteur de `DIG` en section 6.3.

6.1.2 Sortie et représentation graphique

`DIG` vise à produire des requêtes exprimée dans le formalisme *DAFT*, qui sont donc un ensemble d'entités imbriquées. Toutefois, on a vu sur les quelques exemples donnés dans le

chapitre 5 que ce mode de représentation, même correctement indenté, ne rend pas très intuitif la lecture de la phrase. C'est pourquoi `DIG` est doté d'une surcouche de visualisation permettant de représenter cette requête sous forme graphique à l'aide de "boîtes" imbriquées, à la manière de ce que produit `GRASP` pour l'analyse grammaticale. L'existence de cette sortie facilement compréhensible par un humain est essentielle afin de pouvoir réaliser l'évaluation par un annotateur humain décrite dans la section 6.5.
Dans cette représentation, dont un exemple qui va nous servir de fil rouge dans ce chapitre est donné par la figure 6.1, une boîte représente une entité et donc les associations entre entités sont représentées par l'imbrication des différentes boîtes. Le reste des équivalences entre les concepts *DAFT* et leur représentation graphique en sortie de `DIG` est résumé dans le tableau 6.1.

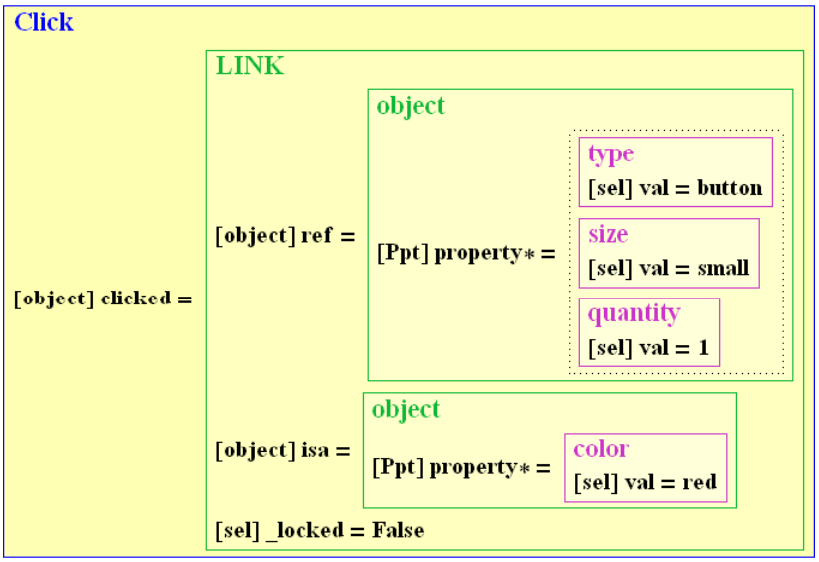


Figure 6.1 Représentation graphique de la requête "Clique le petit bouton qui est rouge" à la sortie de `DIG`

6.2 Ressources

6.2.1 Lien clés sémantiques et *DAFT*

Afin d'assurer la conversion des clés sémantiques issues des nœuds de `GRASP` en entités *DAFT*, il est nécessaire d'associer à chaque clé issue de $\mathbb{K}_G$ un ou plusieurs schèmes de $\mathbb{S}$. Cette opération ne peut se faire que par annotation manuelle, en ajoutant un champ DAFTINFO contenant la ou les entités *DAFT* à instancier. L'exemple de la clé TOCLICK, déjà vu sur la figure 4.4, est donc modifié comme indiqué sur la figure 6.2.

Toutefois, dans certains cas les entités peuvent ne pas seulement dépendre de la clé sémantique, comme dans l'exemple des couleurs comme "rouge", déjà évoqué en 4.2.1.1. S'il s'agit d'un adjectif (JJ), il suffit d'instancier une propriété tandis qu'il faut également instancier

Concept issu de *DAFT*	Représentation graphique en sortie de `DIG`
Une entité	Une boîte
Le nom d'une entité	Le champ en haut à gauche de la boîte
La classe de l'entité	La couleur du nom et de la bordure de la boîte
→ Modalité	→Rouge
→ Action	→Bleu
→ Référence	→Vert
→ Propriété	→Violet
→ Type transparent (*cf.* 5.2.2.2)	→Noir
Un triplet {type, attribut, valeur}	Une ligne dans la boîte
Le type d'un attribut	Premier élément d'une ligne, entre crochets
→ attribut élémentaire	→ Le nom du ou des types
→ attribut terminal	→ "sel" (pas de détails des valeurs possibles)
Le nom d'un attribut	Deuxième élément d'une ligne, suivi de =
→ sans valeur associée	→ Non représenté OU ligne en corps normal
→ avec valeur associée	→ Ligne en corps gras
La valeur d'un attribut	Troisième élément d'une ligne, suivant le =
→ Une valeur terminale	La valeur en toute lettres
→ Une entité	Une boîte imbriquée
→ Valeurs multiples (suffixe ∗)	Boîte en pointillé autour des boîtes des valeurs
Profondeur d'imbrication d'une entité	Fond plus clair lorsque la profondeur augmente

Tableau 6.1 Équivalence entre les éléments du langage *DAFT* et leur représentation graphique produite par `DIG`

```
{
  ID[TOCLICK],
  ONTO[‘¤INTERFACE’, ‘*BUTTON’],
  TYPE[VERB],
  COM[‘Denotes the action of clicking with the mouse’]]
  LEM[‘appuyer’, ‘clicker’, ‘enfoncer’]
  DAFTINFO[{{_, Click[]}}]
}
```

Figure 6.2 Exemple de champ DAFTINFO lié à une clé sémantique : TOCLICK

une entité de type *object* lorsque l'on a affaire à un nom (NN). Cela signifie qu'en pratique, le champ DAFTINFO contient un ensemble de couples (p_i, e_i), où $p \in \mathbb{P}_G$ et $e_i = \{e_{i,1}[\ldots], e_{i,2}[\ldots], \ldots e_{i,n}[\ldots]\}$ tel que $\forall j \in [\![1, n]\!]$, $e_{i,j} \in \mathbb{S}$. Lorsque le nœud associé à la clé dans l'analyse de GRASP a pour nature p_i, c'est l'ensemble des entités contenu dans e_i qui est instancié. L'exemple de "rouge" est donné par la figure 6.3. Si les entités à instancier ne dépendent pas de la nature du nœud GRASP lié à la clé, on utilise le caractère spécial _ (*cf.* figure 6.2).

```
{
  ID[ISRED],
  ...
  DAFTINFO[{
      {NN, {object[], color[val='red']}}
      {JJ, color[val='red']}
  }]
}
```

Figure 6.3 Exemple de champ DAFTINFO dépendant de la nature du nœud GRASP lié à une clé sémantique : ISRED

Sur les 1 294 clés, 697 ont été annotées avec un champ DAFTINFO soit 53,9%. Toutefois, ce nombre n'est pas forcément significatif en lui-même dans la mesure où certaines clés sont peu utilisées ou n'apportent que peu au sens général de la phrase, et que nous nous sommes donc focalisés sur les clés les plus fréquentes et porteuses de sens (par exemple, $QUE est très fréquente mais n'a pas de grande valeur sémantique intrinsèque). Cette annotation permet à près d'un tiers des requêtes (32,4%) d'avoir toutes leurs clés disposant d'un champ DAFTINFO. Il nous semble qu'il sera au contraire plus intéressant d'étudier la validité des requêtes produites en sortie par DIG (*cf.* section 6.5).

6.2.2 Règles de combinaison

Une règle de combinaison $r \in \mathbb{R}_{\mathcal{M}}$ permet de définir un ensemble d'opérations à effectuer sur certains nœuds de la requête fournie par GRASP. Formellement, une règle de combinaison r est donc une fonction de transformation permettant de passer d'un arbre de k entités à un arbre de k' entités, où k peut être différent de k' et où la structure même de l'arbre peut avoir été modifiée en combinant ensemble des entités. L'ensemble des règles de $\mathbb{R}_{\mathcal{M}}$ est appliqué séquentiellement.

Les règles de combinaison sont le pendant sémantique des règles grammaticales décrites en 4.4.2, s'appliquant à des entités au lieu de s'appliquer à des nœuds. La différence est l'existence d'une règle de combinaison dite "par défaut" (décrite en section 6.3.3.2) qui n'existait pas dans le cas de l'analyse grammaticale. Comme son nom le laisse deviner, il s'agit d'une règle de combinaison des entités entre elles, appliquée lorsque les conditions d'application des règles spécifiques ne sont pas remplies.

6.2.2.1 Syntaxe

Une règle de combinaison est constituée de trois éléments :

- Un *identifiant* unique, représenté par une chaîne de caractères, qui résume brièvement l'objectif de la règle ;
- Une en-tête définissant le *schéma* d'entités correspondant aux requêtes à traiter avec cette règle. Chaque constituant du schéma peut être référencé à travers un des quatre paramètres suivants des entités :
  - la classe $\gamma \in \Gamma = \{M, A, R, P\}$ de l'entité ;
  - l'identifiant $s \in \mathbb{S}$ de l'entité ;
  - la nature $p \in \mathbb{P}_G$ du nœud associé dans l'analyse par GRASP ;
  - la clé sémantique $k \in \mathbb{K}_G$ du nœud associé dans l'analyse par GRASP.
- Un corps définissant la *réaction*, composée de structures conditionelles de test (vérifier la valeur d'un champ (`§GRcontains`), l'existence d'un champ `§GRcontains`, l'ordre des éléments dans la requête (`§GRisbefore`), *etc.*) et d'actions élémentaires de modification de la requête (créer une entité (`§GRnew`), remplacer une entité (`§GRsubst`), combiner deux entités (`§GRmerge`), *etc.*).

6.2.2.2 Définition des règles de combinaison

À partir d'une part, d'un sous-ensemble de 1 075 phrases annotées manuellement dans le langage *DAFT* constituant la sortie de référence que l'on souhaiterait obtenir de manière automatique, et, d'autre part, de ces mêmes 1 075 phrases produites par `DIG` en ayant recours uniquement à la règle de combinaison par défaut décrite en section 6.3.3.2, il est possible d'identifier les problèmes de combinaison et d'améliorer les performances de manière incrémentale en définissant des règles de combinaison.
Nous avons ainsi été amenés à définir 24 règles de combinaison "génériques". Par "générique", il faut comprendre que nous ne sommes pas entrés dans les micro-phénomènes propres à un élément en particulier (par exemple pour corriger l'ordre des valeurs associées aux attributs d'un verbe donné). Il est donc évident que des règles supplémentaires seraient nécessaires. Néanmoins, l'ensemble ainsi défini, et donné en annexe F, permet non seulement de traiter les erreurs de combinaison les plus communes (apparaissant lorsqu'on se contente d'appliquer uniquement la règle de combinaison par défaut), mais aussi de valider les opérations élémentaires de modification de requête, qui pourront donc être reprises dans une version raffinée du système de règles de `DIG`.

6.3 Fonctionnement du système

Pour construire une requête *DAFT*, `DIG` doit successivement effectuer trois opérations sur la requête qui lui est fournie en entrée :

1. La **création** d'un arbre des clés sémantiques (notée $\mathcal{T}$), à partir de la requête, en fonction de l'analyse grammaticale ;
2. L'**instanciation** d'entités de $DAFT$ (notée $\mathcal{I}$), afin de les substituer aux clés sémantiques dans l'arbre précédemment construit ;
3. La **combinaison** des différentes entités entre elles (notée $\mathcal{M}$), de façon à obtenir un ensemble structuré selon les règles liées à la syntaxe de $DAFT$.

6.3.1 Création de l'arbre des clés sémantiques

Le principe de l'opération $\mathcal{T}$ est relativement simple puisqu'il consiste simplement à :

1. faire abstraction de la nature exacte des liens grammaticaux unissant les différents constituants d'une requête issue de `GRASP` pour ne conserver que l'organisation générale (*i.e.* quel constituant dépend de quel autre, uniquement d'un point de vue grammatical, puisque c'est ce que fournit `GRASP`) ;
2. ne conserver que le champ KEYS contenant la sémantique élémentaire des constituants.

Ainsi, pour la requête constituant notre exemple-type dans ce chapitre ("Clique le petit bouton qui est rouge"), on obtient l'arbre représenté par la figure 6.4.

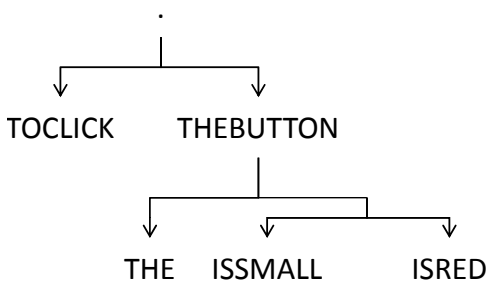


Figure 6.4 Représentation graphique de la requête "Clique le petit bouton qui est rouge" après l'opération $\mathcal{T}$

6.3.2 Instanciation d'entités sémantiques

Formellement, l'opération $\mathcal{I}$ d'instanciation d'une requête consiste à remplacer chaque clé sémantique k de l'arbre issu de $\mathcal{T}$ par la ou les entités qui lui est/sont associée(s) au sein de $\mathbb{K}_G$. Bien qu'il soit théoriquement possible d'associer les entités de $DAFT$ directement à un lemme de $\mathbb{L}_G$ pour obtenir un niveau plus fin de granularité sémantique, le besoin ne s'est pas fait sentir dans le cadre des requêtes du corpus traitées.
En complément de cette opération de base, quatre autres opérations complémentaires sont accomplies à ce moment :

– **l'ajout d'un attribut supplémentaire d'identification Ω :** qui a pour valeur associée le numéro d'identification du nœud d'origine au sein de la requête fournie par `GRASP`. Ceci permet de conserver le lien entre les deux représentations, d'une part pour suivre le processus de construction de la requête formelle dans la phase d'évaluation du

système, d'autre part pour pouvoir accéder si nécessaire à certaines informations comme le POS (*cf.* la troisième opération ci-dessous). Non visible sur la requête d'exemple de la figure 6.1, cet attribut apparaît en revanche sur la figure 6.5.

- **une première combinaison d'entités :** cette opération est effectuée lorsque deux entités sont liées à la même clé sémantique (voir la section 6.3.3 pour le fonctionnement). Ainsi, la clé THEBUTTON est associée à une instance d'un schème de référence (`object[]`) et d'un schème de propriété (`type[val="button"]`). Ces deux éléments sont donc encore plus proches que ne le sont TOCLICK et THEBUTTON dans la figure 6.4, et doivent donc être combinés prioritairement (s'ils peuvent l'être, ce qui est le cas dans cet exemple).
- **la gestion des schèmes dépendant de la nature du lemme associé :** dans certains cas, une même clé sémantique peut être associée à un lemme ayant plusieurs natures possibles, avec un impact sur l'entité à instancier. Par exemple, "rouge" peut avoir pour valeur un nom (NN) dans "je veux le rouge !" ou d'adjectif (JJ) dans "le bouton rouge". Quand il s'agit d'un nom, celui-ci réfère à un élément de l'application et doit donc conduire à l'utilisation d'une entité `object[]` et d'une propriété `color[val="red"]` qui seront combinées comme indiqué dans l'opération ci-dessus. En revanche, dans le cas où l'entité est un adjectif (comme dans l'exemple de la figure 6.1), seule la propriété `color[val="red"]` doit être instanciée.
- **une substitution dynamique de lemmes :** dans certains cas, nous avons choisi (*cf.* discussion dans la section 5.4.1) de regrouper sous une même clé sémantique plusieurs lemmes relevant d'une même catégorie. Ceci permet d'éviter une augmentation artificielle du nombre de clés très peu utilisées. Par exemple, la clé AKOPEOPLE décrit une personne en fonction de la langue qu'il parle au lieu de définir autant de clés que de langues à traiter (*e.g.* ISFRENCH pour "français", ISENGLISH pour "anglais", ISCHINESE pour "chinois"...). Il est cependant nécessaire pour l'agent rationnel $\mathtt{A}_R$ de disposer de la valeur réelle, et c'est pourquoi nous devons remplacer la valeur dynamiquement en fonction du lemme. En pratique, on a alors recours à un mot clé (`§Dlem`) dans la définition de l'entité à instancier. Ainsi, les entités associées à la clé AKOPEOPLE sont `person[]` et `language[val=§Dlem]`, ce qui donnera après l'étape $\mathcal{I}$: `person[property* = {val="francais"}]` pour "un français", `person[property* = {val="anglais"}]` pour "un anglais", *etc.*

6.3.3 Construction de la requête par combinaison

La troisième et dernière étape à accomplir par `DIG` est la combinaison des entités de manière à former une requête structurée, en partant des éléments les plus en profondeur dans l'arbre et en remontant vers la racine. Des trois opérations, il s'agit clairement de la plus complexe et coûteuse en temps de calcul. Dans un premier temps, nous allons formaliser la définition de

l'opération de combinaison $\mathcal{M}$, déjà illustrée sur des exemples dans la section 5.2.2.3, avant de voir son application pratique par DIG.

6.3.3.1 Définition formelle de l'opération de combinaison

La fusion de deux entités *DAFT* R_1 et R_2 en une entité unique R'_1 est réalisée par l'opération de combinaison $\mathcal{M}$ (pour "merging"). R_2 peut être fusionné dans R_1 (ou autrement dit, R_1 peut "absorber" R_2) par l'opération $\mathcal{M}$ si et seulement si il existe un attribut a de R_1 n'ayant pas de valeur associée (ou pouvant avoir plusieurs valeurs associées) et que le type de R_2 appartient au domaine de validité du type associé à a. Formellement, cela donne donc :

$$\mathcal{M}\left(t_{R_1} : R_1\left[\bigcup_{i=1}^{n_1} t_i : a_i = v_i\right], t_{R_2} : R_2\left[\bigcup_{i=1}^{n_2} t_i : a_i = v_i\right]\right)$$
$$\mapsto t_{R_1} : R_1\left[\bigcup_{i=1}^{k-1} t_i : a_i = v_i, t_k : a_k = R_2\left[\bigcup_{j=1}^{n_2} t_j : a_j = v_j\right], \bigcup_{i=k+1}^{n_1} t_i : a_i = v_i\right]$$
$$\text{ssi } t_{R_2} \in \delta_{a_k[R_1]} \wedge (v_k = \emptyset \vee v_k = \{\ldots\})$$

Exemple de combinaison de deux entités DAFT :

```
R1          = {object} object[
                 {ppt}*    : property*   = {
                    {type}   : type[[...]] : val = "button" ]
                 },
                 {act}     : doing       = ∅,
                 {act}     : subject-of  = ∅
              ]
R2          = {color}: color[[...]] : val = "red" ]
M(R1, R2)   = {object} object[
                 {ppt}*    : property*   = {
                    {type}   : type[[...]] : val = "button" ],
                    {color}  : color[[...]] : val = "red" ]
                 },
                 {act}     : doing       = ∅,
                 {act}     : subject-of  = ∅
              ]
```

car `color` $\in \delta_{\text{ppt}}$ (*cf.* l'arbre des types de la figure 5.1).

6.3.3.2 Application de la combinaison

L'algorithme faisant appel à l'opération de combinaison $\mathcal{M}$ pour l'appliquer à l'ensemble de la requête peut être décomposé en trois étapes principales qui constituent les paragraphes suivants.

$\mathcal{M}_1$ **Prétraitement : réordonnancement des entités de la requête** Lorsque deux entités e_1 et e_2 sont de même niveau hiérarchique dans l'arbre construit par $\mathcal{T}$ (par exemple ISSMALL et ISRED sur la figure 6.4), il faut définir une relation d'ordre entre elles pour déterminer si l'on doit commencer par essayer d'associer e_1 à un attribut de e_2 ou plutôt e_2 à un attribut de e_1. De manière générale, au sein des requêtes annotées manuellement pour arriver à la définition du langage *DAFT* (*cf.* section 5.1), on a pu constater qu'une requête était structurée de telle façon que les entités de classe Modalité étaient les plus à l'extérieur, suivi des Actions et des Références, tandis que les Propriétés étaient plutôt en profondeur dans la requête construite (*cf.* section 5.2.2.3). Par conséquent, nous avons choisi de nous baser sur les classes des entités pour les ordonner, tel que : $M \succ A \succ R \succ P$, où $x \succ y$ signifie "y se combine de préférence avec un attribut de x".
Dans le mesure où dans l'exemple de la figure 6.4 nous n'avons pas plus de deux entités au même niveau hiérarchique, il n'est pas très adapté pour illustrer ce réordonnancement. Nous proposons donc, uniquement pour l'illustrer, de considérer le syntagme "le petit bouton qui est rouge" mis à plat, c'est-à-dire tel qu'il serait s'il n'avait pas été préalablement structuré en forme d'arbre par `GRASP` (*cf.* figure 6.5).

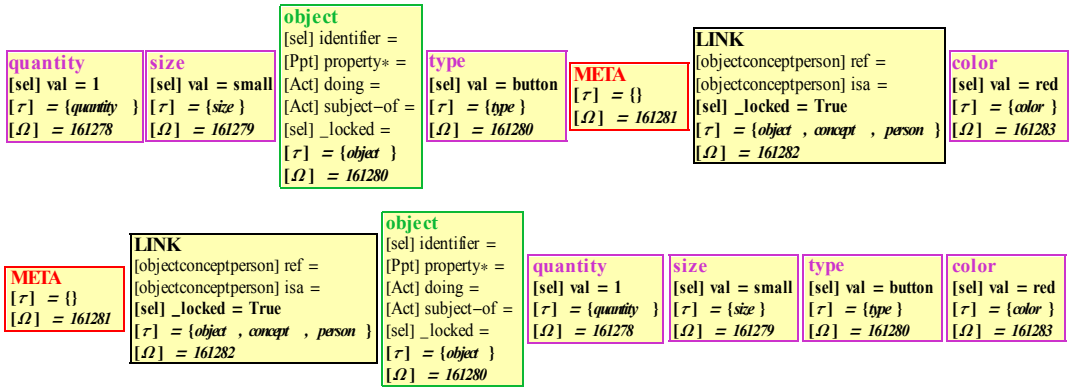


Figure 6.5 Requête partielle "le petit bouton qui est rouge" mise à plat - avant (en haut) et après (en bas) réordonnancement par $\mathcal{M}_1$

Cette étape est en outre également utile pour l'application ultérieure des règles de combinaison spécifiques. En effet, celles-ci étant basées sur un filtrage par motif en fonction des entités *DAFT* de la requête, travailler sur une version "normalisée" de la requête permet de réduire le nombre de cas à prendre en compte : ainsi, on sait qu'on peut rencontrer la séquence (`object` ; `color`) mais pas (`color` ; `object`) (`object` étant de classe R et `color` de classe P).

$\mathcal{M}_2$ **Application des règles spécifiques** Tant que la requête est modifiée par l'application d'au moins une règle, `DIG` passe en revue une à une l'ensemble $\mathbb{R}_{\mathcal{M}}$ des règles spécifiques définies en 6.2.2, de la première à la dernière, et les applique si elles peuvent l'être.

$\mathcal{M}_3$ **Application de la règle de combinaison par défaut** Finalement, si à l'issue de $\mathcal{M}_2$ la requête est constituée de plus d'une seule entité à la racine, `DIG` tente de combiner

toutes les entités qui n'ont pu l'être en donnant la priorité aux entités situées à un même niveau hiérarchique dans l'arbre (*e.g.* si l'on considère l'arbre de la figure 6.4, on a { `Click[] object[...]`, { `quantity[...]`, { `size[...]`, `color[...]` }}}, et on va donc chercher à combiner `size` et `color` ensemble avant d'essayer de combiner `object` et `size`). Si plusieurs entités sont au même niveau dans une liste (`size` et `color` dans l'exemple précédent), on cherche à combiner prioritairement les éléments situés "à droite" à un attribut de libre des éléments situés "à gauche", en privilégiant les plus proches. La justification de ce choix pour la règle de combinaison par défaut est discuté dans la section 6.4.1.

6.4 Discussion : choix d'implémentation et conséquences

6.4.1 Choix relatifs à la règle de combinaison par défaut

Lorsque deux entités a et b sont au même niveau, il est évident que l'ordre de priorité qui va être suivi pour les combiner (*i.e.* a dans b d'abord ou l'inverse) va éventuellement produire des résultats différents, dans la mesure où il peut arriver que a et b aient tous les deux un attribut sans valeur associée et dont le domaine de validité est compatible avec le type de b et a respectivement. Le choix de l'ordre de combinaison mérite donc d'être étudié attentivement. Dans cette sous-section, nous allons considérer l'exemple d'une requête qui, après application des règles de combinaison spécifiques, se retrouve constituée de cinq entités[1] situées à un même niveau soit : $\{a, b, c, d, e\}$. Nous noterons également le fait de "considérer la possibilité d'associer l'entité b à un attribut libre de a", ou autrement dit "tenter d'appliquer l'opération $\mathcal{M}(a, b)$", sous une forme condensée $a[b]$ (pour "a absorbe b").

Dans le prolongement des discussions précédentes, nous pouvons faire ressortir trois principes pour nous aider à déterminer l'ordre de combinaison préférentiel :

1. Dans la mesure où les requêtes annotées manuellement sont plutôt de la forme $M[A[R[P]]]$ (d'après le classe des entités) et que les requêtes ont été prétraitées de manière à être ordonnées ainsi, il semble logique de considérer le principe (P_1) : *on essaye de combiner chaque entité au sein des entités situées à sa gauche avant le contraire.* Par exemple, $b[d]$ sera testé avant $d[b]$.
2. Si l'analyse grammaticale par `GRASP` a été correctement réalisée, les éléments dont les sens s'articulent directement les uns avec les autres sont censés être plutôt proches dans la requête réorganisée, d'où le principe (P_2) : *on essaye de combiner les entités avec leurs voisins avant de considérer les plus distants.* Par exemple, $b[c]$ (distance = 1) est à considérer avant $b[d]$ (distance = 2).
3. En appliquant les deux principes précédents, plusieurs ordres restent possibles, qui ne permettent pas de choisir entre traiter d'abord $a[b]$ (gauche-droite, distance = 1) ou

[1] Car l'on observe extrêmement rarement plus de cinq entités de même niveau hiérarchique dans le cadre des requêtes du corpus *Daft*.

$b[c]$ (gauche-droite, distance = 1). Supposons donc que soient possibles les opérations $\mathcal{M}(a,b)$ et $\mathcal{M}(b,c)$. Si l'on tente d'abord $a[b]$, b n'est plus accessible pour tenter $b[c]$ et on se retrouve avec la liste $\{a[b], c, d, e\}$. D'où le dernier principe (P_3) : *on essaye d'abord de combiner entre elles les entités situées en fin de liste.* Ici cela donnerait alors $\{a[b[c]], d, e\}$.

Une question demeure néanmoins ouverte quant à l'ordre de priorité entre ces trois principes, notamment entre P_2 et P_3. Ainsi, il s'agit de savoir si $a[e]$ doit être testé avant $a[b]$ (P_3 prioritaire sur P_2) ou l'inverse.

Comme d'après P_1 l'ordre de combinaison gauche[droite] prime, nous considérons pour le moment seulement les combinaisons de ce type. Concrètement, on peut alors distinguer 4 politiques Π_i ($i \in [\![1,4]\!]$) qui suivent les principes P_2 et P_3 :
Π_1 **($P_2 > P_3$) :** on essaye de combiner les entités séparées par une distance 1 en commençant par la fin, puis on augmente la distance :

$$\underbrace{d[e], c[d], b[c], a[b]}_{\text{d=1}}, \underbrace{c[e], b[d], a[c]}_{\text{d=2}}, \underbrace{b[e], a[d]}_{\text{d=3}}, \underbrace{a[e]}_{\text{d=4}} \text{ ordre G-D}$$

Π_2 **($P_3 > P_2$) :** on essaye de combiner la $k = N^{\text{e}}$ entité avec un attribut libre de toutes celles situées à sa gauche, en commençant par les plus proches, et on décrémente k jusqu'à 2 (la première entité n'en ayant pas à sa gauche) :

$$\underbrace{d[e], c[e], b[e], a[e]}_{\_[e]}, \underbrace{c[d], b[d], a[d]}_{\_[d]}, \underbrace{b[c], a[c]}_{\_[c]}, \underbrace{a[b]}_{\_[b]} \text{ ordre G-D}$$

Π_3 **($P_3 \gg P_2$) :** on essaye de combiner un attribut libre de la $k = (N-1)^{\text{e}}$ entité avec toutes les entités situées à sa droite, en commençant par les dernières, et on décrémente k jusqu'à 1 :

$$\underbrace{d[e]}_{d[\_]}, \underbrace{c[e], c[d]}_{c[\_]}, \underbrace{b[e], b[d], b[c]}_{b[\_]}, \underbrace{a[e], a[d], a[c], a[b]}_{a[\_]} \text{ ordre G-D}$$

Π_4 **($P_3 > P_2$) :** on essaye de combiner un attribut libre de la $k = (N-1)^{\text{e}}$ entité avec toutes les entités situées à sa droite, en commençant par les plus proches, et on décrémente k jusqu'à 1 :

$$\underbrace{d[e]}_{d[\_]}, \underbrace{c[d], c[e]}_{c[\_]}, \underbrace{b[c], b[d], b[e]}_{b[\_]}, \underbrace{a[b], a[c], a[d], a[e]}_{a[\_]} \text{ ordre G-D}$$

Π_4 présente un problème si $\mathcal{M}(c,b)$ est possible (*cf.* la définition de P_3) ; il reste donc Π_1, Π_2 et Π_3. Si l'on prend maintenant en compte le fait qu'il faut dans un second temps tester les imbrications de la droite vers la gauche, on peut encore générer deux possibilités pour chaque cas, selon que l'on reprenne les essais dans même ordre (noté $_m$) ou en ordre inversé (noté $_i$).

Ainsi, pour la politique Π_1, on peut avoir :

$$\Pi_{1m} : \underbrace{e[d], d[c], c[b], b[a]}_{\text{d=1}}, \underbrace{e[c], d[b], c[a]}_{\text{d=2}}, \underbrace{e[b], d[a]}_{\text{d=3}}, \underbrace{e[a]}_{\text{d=4}} \text{ ordre D-G = même ordre que G-D}$$

ou

$$\Pi_{1i} : \underbrace{e[a]}_{\text{d=4}}, \underbrace{d[a], e[b]}_{\text{d=3}}, \underbrace{c[a], d[b], e[c]}_{\text{d=2}}, \underbrace{b[a], c[b], d[c], e[d]}_{\text{d=1}} \text{ ordre D-G = ordre inverse de G-D}$$

Pour Π_1, l'ordre inverse Π_{1i} viole le principe P_2. Toutefois, pour Π_2 et Π_3, dans la mesure où ils commencent et finissent par des combinaisons d'éléments séparés par une distance de 1, la meilleure solution est moins triviale à déterminer. Il y a donc au final 5 politiques à envisager pour définir l'ordre à utiliser par la règle de combinaison par défaut : $\Pi_{1m}, \Pi_{2i}, \Pi_{2m}, \Pi_{3i}, \Pi_{3m}$. Ces 5 ordres ont été testés en les utilisant pour la règle de combinaison par défaut sur une centaine de phrases issues du corpus, en évaluant qualitativement la qualité de la représentation sémantique produite en sortie par `DIG`. Il est alors apparu qu'en pratique, les représentations produites en sortie dans les 5 cas étaient très proches. D'après nos observations, cette absence de différences, a priori surprenante, de différences se justifie par deux facteurs :

1. les cas où la structure de requête est complexe (4 entités de même niveau ou plus) sont souvent traités par une règle de combinaison spécifique, et donc le changement de politique concernant la règle de combinaison par défaut n'a pas d'effet visible.
2. si la structure a au moins été partiellement correctement traitée par `GRASP` lors de l'analyse des constituants, les listes produites sont plutôt courtes (3 entités maximum), et les différences entre politiques sont beaucoup moins sensibles, d'autant plus si les premiers essais de combinaison permettent de produire un résultat. Ainsi, si on considère $\{a, b, c\}$ pour l'ordre gauche-droite seulement, on voit que Π_2 et Π_3 ne sont plus discernables l'une de l'autre : $\Pi_1 \rightarrow \{b[c], a[b], a[c]\}$, $\Pi_2 \rightarrow \{b[c], a[c], a[b]\}$, $\Pi_3 \rightarrow \{b[c], a[c], a[b]\}$.

C'est pourquoi nous avons finalement choisi, arbitrairement, Π_{3m} comme politique d'ordre pour la règle de combinaison par défaut, soit sur notre exemple :

$$\underbrace{d[e]}_{d[\_]}, \underbrace{c[e], c[d]}_{c[\_]}, \underbrace{b[e], b[d], b[c]}_{b[\_]}, \underbrace{a[e], a[d], a[c], a[b]}_{a[\_]}, \underbrace{e[d]}_{\_[d]}, \underbrace{e[c], d[c]}_{\_[c]}, \underbrace{e[b], d[b], c[b]}_{\_[b]}, \underbrace{e[a], d[a], c[a], b[a]}_{\_[a]},$$

6.4.2 Quand a-t-on affaire à une référence ?

Nous avons expliqué, dans la définition du langage *DAFT*, que les références étaient considérées comme une collection de propriétés permettant de définir à proprement parler l'élément auquel on réfère. Cependant, du point de vue pratique de `DIG`, la question se pose alors de savoir dans quels cas une entité de classe R doit être instanciée. Ainsi, dans notre phrase d'exemple récurrente "Clique le bouton rouge", quel élément doit permettre l'instanciation d'une entité `object[]` ?

– “rouge” n’est pas un bon candidat, dans la mesure où “Clique le bouton” devrait également avoir une référence présente.

– “bouton” semble être la meilleure option, dans la mesure où il s’agit d’un type ayant implicitement une existence réelle dans le monde (de l’application). Toutefois, “clique le rouge” pourrait être une requête tout à fait valide, nécessitant également la présence d’une référence.

– “le”[2] est utilisé comme quantifieur de références, ce qui signifie donc que sa simple présence devrait permettre de supposer l’existence et l’unicité de celle-ci [Corblin, 1987]. Néanmoins, si “le” est obligatoire dans le cadre d’une phrase grammaticalement bien formée, on a vu dans les extraits de corpus présentés à plusieurs reprises (*cf.* annexe A) que les phrases recueillies dans le corpus *Daft* étaient loin d’être toutes grammaticalement correctes. Pour les requêtes de contrôle en particulier, on note beaucoup de phrases du type “Cliquer bouton rouge”, où l’idée de référence doit apparaître dans la requête finale, sans pour autant qu’il n’y ait de “le”.

Il apparaît donc que la notion de référence est “diffuse” et ne peut être directement associée à l’une des propriétés qui lui sont associées. Nous avons donc géré cette situation en associant la création d’une référence à la présence d’un nom dans la requête. Ceci explique en grande partie la nécessité de la prise en compte de la nature des mots pour l’instanciation de requête, tel qu’illustré sur la figure 6.3 où l’on instancie une référence pour “rouge” si et seulement si il s’agit d’un nom.

6.4.3 Gestion du focus d’une requête

Un type de requête d’assistance assez courant dans le corpus *Daft* consiste à demander la valeur actuelle de la propriété d’un objet (*e.g.* “est-ce que le champ texte est éditable ?”). Avec le mode de représentation choisi, les propriétés étant mises au même niveau, on ne distingue pas par défaut les propriétés définissant la référence (`quantity[val=1]` et `type[val="textfield"]`) de celle sur laquelle porte la question (`editable[val=True]`) et qu’on qualifiera comme étant le “focus” de la requête.

[2] On considère seulement ici l’article défini “le”, pas le pronom personnel servant d’anaphore par exemple dans “clique-le”.

Exemple de deux requêtes similaires avec un focus différent :

R_{f1} = "Coco le compteur est actif ?"

```
= ASK[about = VALUE[
    of = object[
       property* = {
          name[val = "Coco"],
          quantity[val = 1]}
       doing = Count[]],
    property = enabled[],
    value = True]
```

→ focus = valeur de `enabled`

R_{f2} = "Coco est le compteur actif ?"

```
= ASK[about = VALUE[
    of = object[
       property* = {
          enabled[val = True],
          quantity[val = 1]}
       doing = Count[]],
    property = name[],
    value = "Coco"]
```

→ focus = valeur de `name`

Bien que seule R_{f1} soit réellement issue du corpus, on voit qu'en utilisant le formalisme défini lors de l'annotation du corpus, la différence avec R_{f2} est non triviale à établir de manière automatique. En effet, il va falloir associer les trois propriétés de l'objet à deux champs (`property` et `property*`) ayant un même type associé (*ppt*).

Nous avons donc été amenés à modifier le formalisme de *DAFT* en introduisant une modalité spéciale, `LINK`, modélisant la sémantique de propositions de type : "l'objet X a une propriété P ayant pour valeur V". En langue naturelle, cela se traduit généralement par l'emploi de "être" ou "avoir", par opposition aux liens implicites faits lorsqu'on utilise une référence sous forme de syntagme nominal (*e.g.* "le petit bouton rouge") ou une apposition (*e.g.* "Coco le compteur").

Avec cette modalité, la question sur la valeur d'une propriété se modélise alors désormais sous la forme d'un `ASK[about=LINK[...]]`, qui signifie littéralement que l'on pose une question sur l'identification entre l'objet de référence et un autre objet dont une des propriétés a une valeur donnée. La représentation des requêtes R_{f1} et R_{f2} telle qu'obtenue en sortie de `DIG` est représentée sur la figure 6.6.

6.4.4 Champs bloqués

Avec certaines entités, l'utilisation de la règle de combinaison par défaut entraîne systématiquement la formation de requêtes n'ayant pas de sens sémantiquement parlant. C'est le cas par exemple de la modalité `LINK[]`, qui n'a de sens que si ses deux attributs (`ref` et `isa`) ont chacun une valeur associée, pour signifier que la première référence est liée à la seconde. Or il n'y a pas de moyen simple de forcer les deux attributs à se voir associer une valeur en même temps[3], et si l'utilisation d'une règle spécifique va s'avérer nécessaire pour produire la

[3]Sauf à utiliser un système d'unification à la manière de *Prolog*. Mais d'une part ce mécanisme n'existe pas par défaut sous *Mathematica*, et d'autre part, son implémentation aurait un impact sur le temps de calcul, d'où le mécanisme mis en place ici pour "simuler" ce fonctionnement.

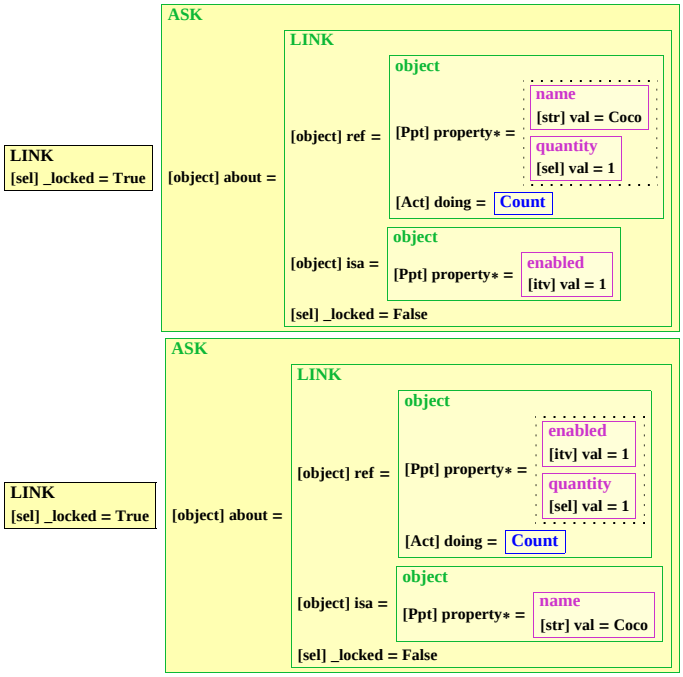


Figure 6.6 Représentation graphique des requêtes R_{f1} (en haut) et R_{f2} (en bas)

bonne représentation de la requête, il faut s'assurer qu'une autre référence n'aura pas déjà été associée à un des attributs auparavant.

Pour résoudre ce problème, l'idée consiste à empêcher l'application de la règle de combinaison par défaut dans certain cas, ce qui techniquement peut être implémenté d'au moins trois manières différentes :

1. en transformant chaque triplet $\{t_i, a_i, v_i\}$ en quadruplet $\{t_i, a_i, v_i, \lambda_i\}$, où λ_i serait un booléen indiquant si le champ est bloqué ou non ;
2. en définissant explicitement une 'stoplist' d'attributs de certains schèmes qui ne peuvent être utilisés par la règle de combinaison ;
3. en ajoutant un attribut spécial `_locked` aux schèmes qui ne peuvent être utilisés par la règle de combinaison.

La solution 1 est la plus flexible et générique. Toutefois, elle est aussi la plus lourde à implémenter, puisqu'elle suppose l'ajout d'un élément à tous les attributs de tous les schèmes, alors même que le nombre de cas particuliers que nous souhaitons traiter concerne en pratique environ une dizaine de clés sémantiques (sur 1 294 clés au total).
La solution 2 est plus ad hoc mais aussi plus facile à implémenter. Son principal défaut est toutefois lié au fait que la stoplist ainsi définie est statique : il n'est pas possible de prendre en compte le fait qu'on puisse vouloir "débloquer" l'accès par la règle de combinaison par défaut à certains attributs une fois que d'autres ont été correctement remplis.
La solution 3 est également plus légère que la première en termes d'implémentation (puisque le champ spécial n'est ajouté qu'aux schèmes qui en ont besoin). De plus, contrairement à la deuxième solution, le blocage est réversible puisqu'il suffit de modifier la valeur du booléen

lorsque la règle spécifique nécessaire a été appliquée. La seule restriction est l'impossibilité de bloquer uniquement certains champs d'une entité.

Au final, nous avons opté pour l'utilisation conjointe des solutions 2 et 3 : la deuxième lorsqu'un attribut doit être définitivement bloqué pour l'opération $\mathcal{M}$, la troisième lorsque le fait que certains attributs d'une entité se voient associer une valeur permet d'en débloquer d'autres (via l'opération `§DRunlock` – *cf.* annexe F).

6.4.5 Vers une association plus étroite avec l'agent rationnel

`DIG` produisant des représentations sémantiques sur lesquelles l'agent rationnel $\mathtt{A}_R$ doit se baser pour réagir, on peut imaginer qu'après la définition de celui-ci (évoquée en perspective dans le chapitre 8), le fonctionnement de `DIG` puisse être partiellement revu afin de produire une interprétation qui soit guidée à la fois par le bas (via l'analyse de corpus) et par le haut (en anticipant sur les heuristiques de traitement).

6.4.5.1 Gestion plus fine des ambigüités sémantiques

Dans certains cas, `DIG` doit être en mesure de gérer des ambigüités sémantiques qui ne peuvent être traitées au niveau de l'étape de WSD de `GRASP`. Ainsi, on peut considérer le cas de l'ambigüité entre `concept` et `object` dans la requête "Est-ce que tu aimes le rouge ?", pouvant signifier :

- "Est-ce que tu aimes [l'objet rouge le plus saillant dans l'application] ?", où la question porte sur un objet de l'application et donc d'une référence de type `object[...]`.
- "Est-ce que tu aimes [la couleur rouge en général] ?", où il s'agit d'une question sur les goûts de l'agent et d'une référence de type `concept[...]`.

`DIG`, pas plus que `GRASP`, n'est réellement en mesure de déterminer le sens exact de la requête dans la mesure où elle relève plus de la pragmatique que de la sémantique. Dans les cas comme celui-ci, nous avons choisi de nous baser sur la situation qui occurre le plus souvent d'après le corpus (pour cet exemple, la première interprétation) et de la choisir par défaut (d'où ici une instanciation de l'entité `object[ppt*=color[val="blue"]]`. Néanmoins, il serait préférable de pouvoir gérer de manière plus satisfaisante ces deux cas. Deux approches sont possibles :

Solution 1 : avoir recours à un schème spécial encapsulant les deux valeurs possibles (à la manière de la modalité `UNION`), *e.g.* `AMBIGUOUS[option1 = object[ppt* = color[val = "blue"]], option2 = concept[ppt* = color[val = "blue"]]]`. À partir de là, l'analyse ultérieure de la requête pourrait être séparée en autant de cas que nécessaire, exécutés de manière concurrente puis comparés :

- au niveau représentationnel : en comparant les requêtes finalement produites en sortie par `DIG` dans les deux cas, on pourrait par exemple choisir la représentation la plus structurée ou la plus proche de celles présentes dans le corpus *Daft*.

– au niveau réactionnel : en transmettant les différentes interprétations possibles à l'agent rationnel et en effectuant le choix entre les réactions engendrées par chacune des interprétations (actions à réaliser, réponse à prononcer, émotions à exprimer, *etc.*).

Solution 2 : avoir, parmi les heuristiques de réaction de l'agent rationnel (telles que décrites dans la section 8.2.2.4) une heuristique particulière de gestion des ambigüités, qui autoriserait de lancer une nouvelle analyse sémantique de la phrase par `DIG`. Ainsi, à partir de l'exemple traité ici, on pourrait lors de la première analyse, au choix :

1. traiter le cas le plus abstrait et hors contexte, *i.e.* ici l'instanciation d'un `concept[]` ;
2. considérer au contraire le contexte de l'assistance qui pousse à interpréter en priorité une requête comme faisant référence à l'application assistée, et choisir par défaut la représentation avec une référence `object[]`.

L'heuristique de gestion d'ambigüité aurait alors pour rôle de vérifier, à partir du modèle de l'application à sa disposition, l'existence d'éléments bleus saillants[4] :

– si elle trouve un élément suffisament saillant : dans le premier cas, elle relance une interprétation par `DIG` en forçant l'utilisation d'une référence `object[]`, dans le second, l'analyse de la requête se poursuit normalement en faisant désormais abstraction de l'ambigüité.
– si elle ne trouve pas d'élément saillant : dans le premier cas, elle poursuit l'analyse, dans le second, elle relance une interprétation par `DIG` en forçant l'utilisation d'une référence `concept[]`.

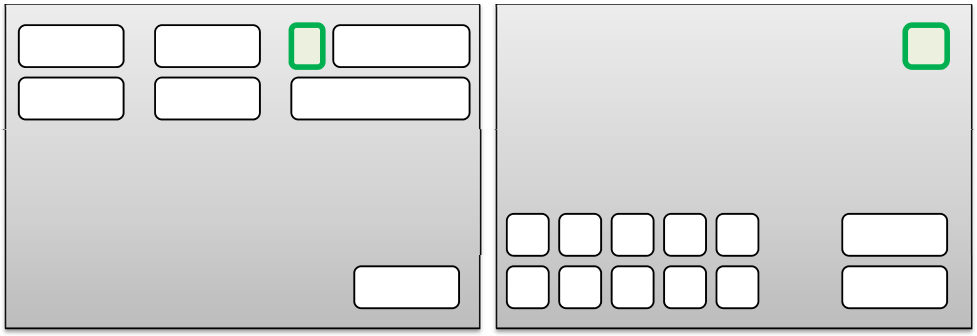
Si la première solution semble plus facile à implémenter, surtout si le choix doit se faire au niveau représentationnel, dans l'optique d'un agent rationnel et psychologique tel que décrit en 8.2, la seconde semble plus prometteuse. En effet, elle semble plus proche de la manière dont un humain traite une requête de ce type : en fonction de la situation, on privilégie dans un premier temps une des deux interprétations, avant éventuellement de reconsidérer ce choix si après réflexion l'autre interprétation semble finalement plus probable. Par ailleurs, cette capacité à revenir sur son interprétation originelle ou la tendance à opter pour une interprétation plutôt qu'une autre sont des comportements qui pourraient être dépendants de la personnalité ou de l'état émotionnel courant de l'agent (*cf.* section 8.2.1.1).

6.4.5.2 Ordre de traitement des propriétés

Nous avons fait le choix de faire apparaître toutes les entités de classe P au même niveau au sein d'une référence. Cependant, quand l'agent rationnel utilise la représentation d'un `object[...]` pour l'identifier au sein du modèle de l'application auquel il a accès, certaines propriétés sont sensibles à l'ordre dans lequel elles sont traitées. Nous allons illustrer le problème à l'aide de trois exemples de propriétés, sensibles ou non à l'ordre de traitement.

[4] Ce qui requiert d'être en mesure d'évaluer la saillance des éléments (au sens gestaltien), comme envisagé par Leray [2009]

Exemple 1 : une propriété non-sensible à l'ordre : `location`.
Bien que la position spatiale d'un élément soit mise en évidence dans la langue naturelle et dans le langage *DAFT* (*cf.* section 5.4.3) par rapport à d'autres propriétés, d'un point de vue rationnel, elle n'a ni plus ni moins d'importance qu'une autre propriété. Ainsi "le petit bouton en haut" peut tout autant être interprété comme "le petit bouton, qui se trouve en haut", "le bouton, qui est en haut et qui est aussi petit" ou "l'élément en haut, qui est un petit bouton". L'ordre de traitement est donc sans importance particulière, tout du moins a priori. En effet, dans des situations particulières, une propriété peut être plus discriminante qu'une autre et pourrait avoir intérêt à être prise en compte en premier lieu. Ainsi, dans l'exemple donné par la figure 6.7, il vaudrait mieux considérer la taille en premier dans l'application 1, à l'inverse de l'application 2 où la position est plus discriminante.



(a) Application 1 : un seul petit bouton (b) Application 2 : un seul bouton en haut

Figure 6.7 Exemple d'applications particulières où l'ordre des propriétés peut faciliter la recherche d'une référence (en vert épais : la référence recherchée)

Exemple 2 : une propriété sensible à l'ordre à considérer en dernier : `quantity`.
La quantité, dont le rôle est généralement supporté par le déterminant, permet d'effectuer une sélection au sein d'un ensemble déjà défini. Ainsi, si l'on recherche les éléments correspondant à la référence "plusieurs boutons rouges", on pourra faire :

1. rechercher tous les objets rouges,
2. rechercher les objets de type bouton,
3. faire l'intersection des deux ensembles obtenus en 1 et 2,
4. vérifier qu'il reste plusieurs éléments.

Si les étapes 1 et 2 peuvent être interverties, en revanche il est clair qu'effectuer la quatrième plus tôt ne serait pas logique et entraînerait un résultat erroné.

Exemple 3 : une propriété sensible à l'ordre à considérer en premier : `time`.
Dans le cas de la notion de temps, au contraire, il est essentiel de la prendre en compte le plus tôt possible car cela peut changer complètement l'espace dans lequel une référence doit être recherchée. Ainsi, la formulation au passé de la requête "C'était quoi le petit bouton rouge

en haut ?" sous-entend que l'objet en question n'est actuellement plus visible : le rechercher dans le modèle actuel de l'application serait donc sans intérêt, et il faudrait au contraire faire appel (s'il existe) à un modèle de l'application à instant antérieur (*cf.* les travaux de Sabouret & Sansonnet [2000] sur les chroniques).

6.5 Évaluation

Bien que l'évaluation du langage *DAFT* réalisée à partir d'une annotation manuelle du corpus tel que défini en 5.3 ait laissé penser que celui-ci était adapté à la représentation de la sémantique des requêtes, nous avons vu que la construction de manière automatique par `DIG` des requêtes en *DAFT* a rendu nécessaire un certain nombre d'ajustements. Par conséquent, il est important d'évaluer la qualité des représentations sémantiques produites par cet outil. De plus, dans la mesure où la détection d'erreurs (*i.e.* différence entre la représentation *DAFT* sortie de `DIG` et la représentation idéale telle qu'on peut la construire manuellement) requiert une connaissance de la sémantique réelle de la requête, cette étape ne peut être effectuée que par un annotateur humain.
Nous allons donc dans un premier temps proposer les critères d'évaluation que nous avons retenus pour annoter les requêtes et illustrer la pertinence de chacun d'eux sur quelques exemples ; ensuite, nous présenterons les résultats de l'annotation selon ces mêmes critères d'un sous-ensemble du corpus traité par `DIG`.

6.5.1 Critères d'analyse qualitative d'une représentation sémantique

Idéalement, si une requête est correctement représentée sous forme formelle *DAFT*, on doit pouvoir en "lisant" celle-ci reconstituer le sens voulu par le locuteur l'ayant émise. Ainsi, la requête de la figure d'exemple 6.1 peut se "lire" en analysant les boîtes : "Il y a un objet de type bouton, de petite taille et unique qui est de couleur rouge (contenu de `LINK`). Exécuter un clic sur cet objet."
Toutefois, de par la manière dont le corpus a été recueilli, nous n'avons pas accès aux intentions d'origine du locuteur et prendrons donc comme référence le sens tel qu'il peut être compris par un autre humain (l'annotateur) en dehors de tout contexte (puisqu'on a dit en 3.1.3.4 qu'on travaillait sur des requêtes isolées). L'évaluation de l'adéquation entre sens compris en lisant la requête en langue naturelle et sens compris en lisant la sortie de `DIG` peut être décomposée selon trois paramètres :

1. la **qualité** de la représentation : c'est-à-dire une évaluation de la facilité avec laquelle un humain peut donner un sens à la requête produite par `DIG`, ceci sans avoir sous les yeux la requête d'origine en langue naturelle.
2. la **représentativité** de la requête : c'est-à-dire le degré de corrélation existant entre le sens de la requête produite par `DIG` et le sens de la requête en langue naturelle d'origine.

3. la **complétude** de la requête : c'est-à-dire la proportion d'informations présentes dans la requête en langue naturelle que l'on retrouve dans la requête produite par `DIG`.

Représentativité et complétude peuvent être vus comme une forme de précision et de rappel. Pour chacun de ces paramètres, nous avons proposé d'attribuer une note sur une échelle de 4 points (entre 0 et 3), ce qui nous permet d'avoir trois évaluations notées $\mathcal{E}_Q$, $\mathcal{E}_R$ et $\mathcal{E}_C$. Ces évaluations peuvent être vues comme la réponse à trois questions associées :

$\mathcal{E}_Q$: La requête *DAFT* fournie par `DIG` a-t-elle un sens ?
- 3 : Oui
- 2 : Plutôt oui, et il y a soit :
  - une structure sémantique principale, et quelques éléments additionnels non reliés qui n'apportent pas de sens supplémentaire,
  - plusieurs éléments non reliés, mais dont le sens est compréhensible en les prenant dans l'ordre où ils se suivent.
- 1 : Plutôt non, il y a des îlots d'éléments qui font sens, mais il n'y a pas au moins un sens général se dégageant de manière certaine de la requête.
- 0 : Non, la requête est incompréhensible.

$\mathcal{E}_R$: La requête *DAFT* fournie par `DIG` a-t-elle le même sens que la requête en langue naturelle ?
- 3 : Oui
- 2 : Plutôt oui, en dépit de quelques imprécisions, le sens général est correct.
- 1 : Plutôt non, il y a des faux-sens importants.
- 0 : Non, il y a des contre-sens ou aucun rapport.

$\mathcal{E}_C$: La requête *DAFT* fournie par `DIG` contient-elle toutes les informations présentes dans la requête en langue naturelle ?
- 3 : Oui
- 2 : Plutôt oui, des détails non essentiels ont été perdus.
- 1 : Plutôt non, certains éléments essentiels ont été perdus.
- 0 : Non, les éléments essentiels ne sont pas présents.

Bien que complémentaires, ces critères ne sont pas décorrellés, puisque la représentativité $\mathcal{E}_R$ est liée aux deux autres paramètres :

- à la qualité $\mathcal{E}_Q$, car une requête incompréhensible ne peut pas vraiment être comparée à quoi que ce soit : $\mathcal{E}_Q$ faible $\Rightarrow$ $\mathcal{E}_R$ faible.
- à la complétude $\mathcal{E}_C$, car une requête dont les éléments essentiels sont absents n'aura forcément pas le même sens que celle d'origine : $\mathcal{E}_C$ faible $\Rightarrow$ $\mathcal{E}_R$ faible.

Inversement, une requête dont tous les blocs sémantiques sont présents et qui a un sens lorsqu'elle est prise dans son ensemble, peut avoir un sens différent de celui de la requête d'origine

s'ils ont été mal combinés : $(\mathcal{E}_Q \wedge \mathcal{E}_C)$ élevés $\not\Rightarrow \mathcal{E}_R$ élevé. Par conséquent, la représentativité demeure un paramètre d'évaluation important.

Dans la section suivante, nous nous référerons à ce système de notation sous le nom de $\mathcal{E}_{QRC}$. Logiquement, on dira alors qu'une requête ayant obtenu un score de qualité $\mathcal{E}_Q$ de 3, de représentativité $\mathcal{E}_R$ de 2 et de qualité $\mathcal{E}_C$ de 1 a un score $\mathcal{E}_{QRC}$ de 321.

6.5.2 Annotation des phrases

6.5.2.1 Définition des activités conversationnelles du corpus *Daft*

Nous avons réalisé différentes études du corpus *Daft* qui seront détaillées dans le chapitre 7, et avons besoin ici pour l'annotation des phrases issues de `DIG` de la notion de d'activité conversationnelle. Nous allons donc pour le moment poser ce concept, et renvoyons à la section 7.2 pour plus de détails.
Nous avons pu distinguer, au sein des phrases collectées dans le corpus *Daft* quatre grandes classes d'activités, représentant les attentes de l'utilisateur vis-à-vis du système :

1. le contrôle : où l'utilisateur se sert de l'agent comme médiateur pour effecteur des tâches au sein de l'environnement de l'application assistée ;
2. l'assistance directe : lorsque l'utilisateur fait explicitement appel à l'agent et à ses connaissances pour l'aider dans la réalisation d'une tâche ;
3. l'assistance indirecte : quand l'utilisateur masque son besoin d'aide en le formulant par exemple sous la forme d'un regret ("dommage qu'on ne puisse pas…") ;
4. la discussion : qui regroupe toutes les interactions davantage centrées sur le personnage de l'agent que sur l'application qu'il assiste, et où l'on retrouve toutes les requêtes typiquement recueillies par un agent conversationnel classique (non assistant).

La distribution des requêtes au sein de ces classes est donnée par le diagramme circulaire de la figure 6.8.

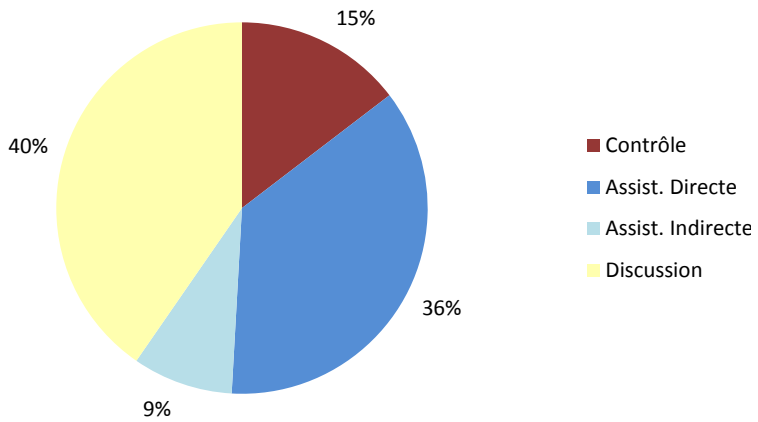


Figure 6.8 Répartition des principales activités conversationnelles du corpus *Daft*

6.5.2.2 Sous-ensemble de *Daft* annoté

Dans la mesure où les schèmes de *DAFT* ont été définis essentiellement pour traiter les requêtes d'utilisateurs dans le contexte de l'assistance, nous excluons de cette étude les requêtes relevant de la discussion qui requièrent de traiter une part beaucoup plus importante de la langue naturelle, nous faisant perdre l'intérêt de travailler dans le cadre d'un sous-langage. L'annotation a donc été réalisée sur les 640 phrases du sous-corpus $Daft_{sub}$ (défini en section 3.3.3.2) relevant du contrôle ou de l'assistance.

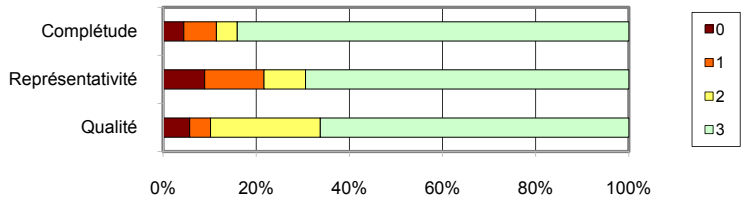
6.5.2.3 Résultats

Paramètres considérés indépendamment. Les résultats de l'annotation, paramètre par paramètre, sont synthétisés par la figure 6.9, montrent que :

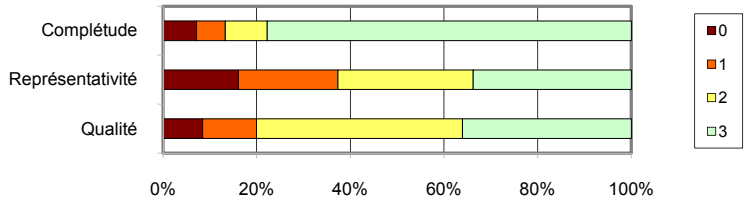
- la proportion de requêtes complètes (score de 2 ou 3 pour $\mathcal{E}_C$) est relativement constant, de l'ordre de 90%. Cela montre que bien qu'un certain nombre de clés sémantiques n'ont pour le moment pas eu de champ DAFTINFO associé, les éléments les plus utilisés ont bien un ou plusieurs schèmes associés.
- la proportion de requêtes ayant un sens (score de 2 ou 3 pour $\mathcal{E}_Q$) est de l'ordre de 90% pour les requêtes de contrôle, mais seulement de 80% dans le cas des requêtes d'assistance à proprement parler. Ce résultat était prévisible : les requêtes de contrôle étant généralement plus simples et plus courtes (*cf.* section 7.3.2) : elles mettent en jeu moins d'entités, et donc il y a moins de risques d'erreurs en les combinant entre elles.
- la proportion de requêtes dont le sens est bien représenté (score de 2 ou 3 pour $\mathcal{E}_R$) diffère sensiblement selon les activités conversationnelles considérées : 80% pour le contrôle, 65% pour l'assistance directe et 60% pour l'assistance indirecte. On retrouve ici l'illustration de l'idée intuitive selon laquelle les requêtes d'assistance directe, et encore plus indirecte, sont plus complexes en langue naturelle, donc plus complexes à représenter de manière formelle et en conséquence, plus difficiles à interpréter.

Non traitée ici, on peut toutefois imaginer l'allure qu'aurait l'histogramme pour l'activité de discussion. En effet, si toutes les clés du corpus s'étaient vu associer des entités, de manière à ce que le score de complétude soit du même ordre de grandeur (90%) pour les requêtes de discussion que pour les autres activités conversationnelles, on peut estimer que les scores de $\mathcal{E}_R$ variraient grandement d'une requête à l'autre dans la mesure où la complexité des requêtes varient grandement (*cf.* section 7.2.4).

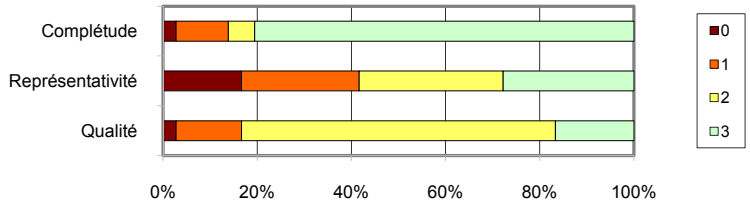
Scores de $\mathcal{E}_{QRC}$. Si on considère maintenant les 3 notes prises conjointement sous forme d'un triplet, sur les 64 valeurs théoriquement possibles, seules 39 apparaissent effectivement (selon une loi de Zipf, tel que représenté sur la figure 6.11), ce qui est logique, d'après les liens entre les paramètres qui ont été mentionnés précédemment. Les cinq valeurs les plus fréquentes sont :



(a) Contrôle



(b) Assistance directe



(c) Assistance indirecte

Figure 6.9 Évaluation des requêtes dans $\mathcal{E}_Q$, $\mathcal{E}_R$ et $\mathcal{E}_C$, en fonction de leur activité conversationnelle

- 333 : pour les requêtes parfaitement analysées,
- 223 : pour les requêtes pas complètement bien formées mais pour lesquelles les erreurs n'ont qu'un impact mineur sur le sens,
- 233 : pour les requêtes pas complètement bien formées mais sans que cela n'empêche d'extraire le sens général de la requête pour un annotateur humain,
- 213 : pour les requêtes pas complètement bien formées tel que cela a un impact significatif sur le sens de la requête par rapport à la requête en langue naturelle,
- 000 : pour les requêtes non traitées par `DIG` (faute de schèmes associés aux clés).

Si comme précédemment on cherche à distinguer ces scores en fonction des activités conversationnelles (*cf.* figure 6.10), on retrouve de manière assez nette les différences déjà évoquées rien que dans l'allure des courbes : la proportion de 333 est nettement majoritaire pour les requêtes de contrôle, légèrement majoritaire pour les requêtes d'assistance (suivi de près par 223) et seulement le quatrième score le plus fréquent pour les requêtes d'assistance indirecte.

6.6 Conclusion

Dans ce chapitre, nous avons introduit `DIG`, un outil permettant de produire de manière automatique des requêtes en *DAFT* de manière à les rendre exploitables par l'agent rationnel

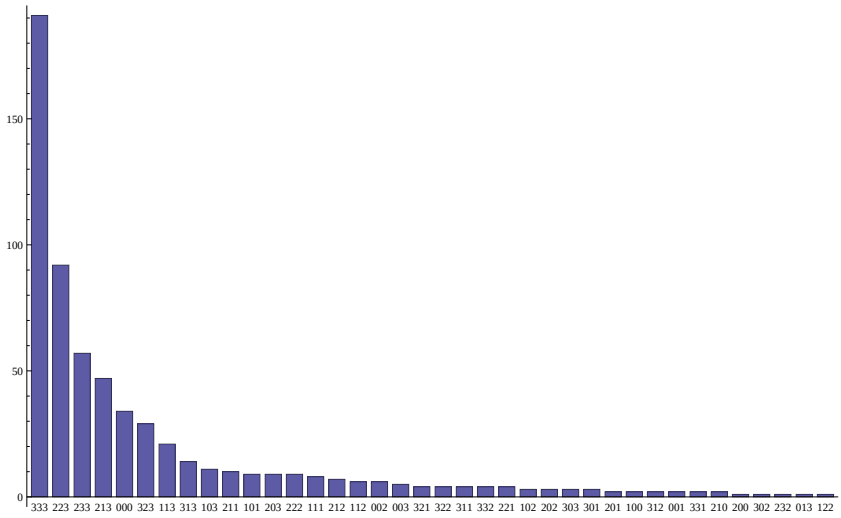
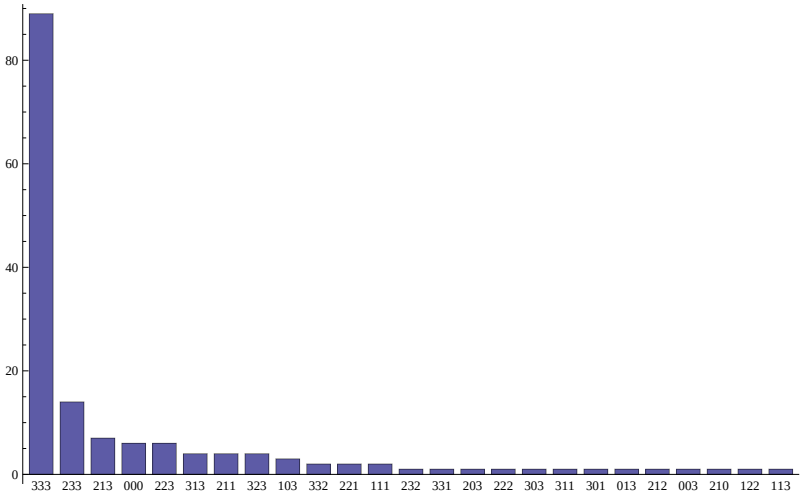


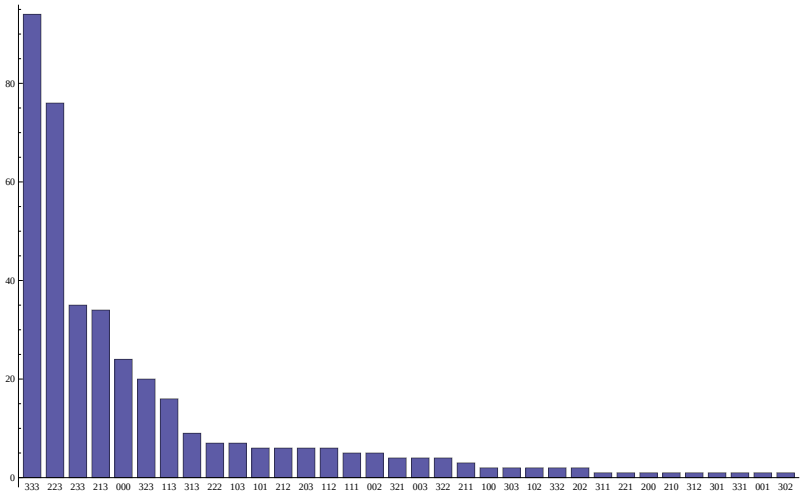
Figure 6.10 Évaluation des requêtes dans $\mathcal{E}_{QRC}$ (nombre de requêtes pour chaque score)

$\mathtt{A}_R$ (qui sera abordé dans le chapitre 8. Nous avons vu que cette automatisation n'était pas triviale, et que divers mécanismes (ajout de champs, règles de combinaisons spécifiques...) devaient être mis en œuvre afin d'atteindre le résultat voulu. Nous avons enfin proposé une méthodologie d'évaluation des requêtes produites, qui pourra être de nouveau appliquée pour évaluer l'impact de futures améliorations - notamment si l'on associe de nouveaux schèmes aux différentes clés sémantiques de $\mathbb{K}_G$.

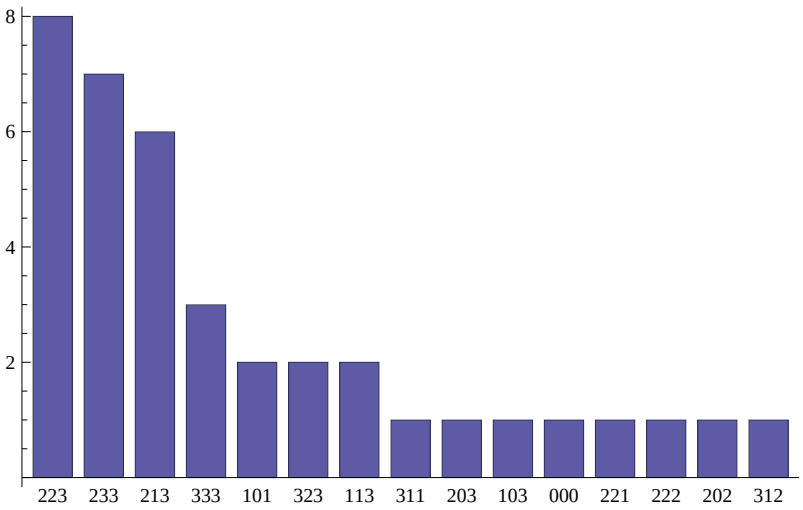
Notons qu'en ce qui concerne le temps d'exécution, `DIG` est à peu près comparable à `GRASP`, avec un temps moyen de 0,3 seconde d'analyse par requête du corpus *Daft*, soit une moyenne totale pour l'ensemble $\{\mathtt{GRASP} + \mathtt{DIG}\}$ de 0,7 seconde par requête. En sachant que la chaîne de traitement doit encore comporter le module correspondant à l'agent rationnel $\mathtt{A}_R$, ce temps d'analyse semble raisonnable pour permettre une utilisation en pratique du système `DAFT`.



(a) Contrôle



(b) Assistance directe



(c) Assistance indirecte

Figure 6.11 Évaluation des requêtes dans $\mathcal{E}_{QRC}$, selon leur activité conversationnelle

Troisième partie

Évaluations et expérimentations

Chapitre 7

Évaluation statistique des ressources linguistiques

Sommaire

Dans ce chapitre, nous nous intéressons dans un premier temps à la distinction entre le corpus *Daft* et d'autres corpus similaires afin de mettre en valeur les spécificités de *Daft* considéré dans son ensemble. Dans un second temps, nous nous tournons vers une analyse interne du corpus, de manière à distinguer différentes activités conversationnelles, et nous recherchons enfin des indices permettant d'automatiser leur identification en vue d'une éventuelle classification automatique.

7.1 Caractérisation du corpus de requêtes d'assistance

Suite à la constitution du corpus *Daft* selon les modalités définies dans le chapitre 3 afin d'étudier les aspects linguistiques liés à la Fonction d'Assistance, nous nous intéressons dans ce chapitre à l'étude de ses caractéristiques en analysant ce qui permet de le distinguer d'autres corpus existants : d'abord en le contrastant avec un corpus généraliste, puis en appliquant une méthode fondée sur les actes de dialogue pour le comparer à d'autres corpus a priori plus proches.

7.1.1 Comparaison à un corpus généraliste

Bien qu'il apparaisse évident qu'un corpus généraliste n'aurait pu se substituer au corpus *Daft* pour l'utilisation que nous souhaitons en faire, nous avons toutefois voulu évaluer comment se situait le corpus *Daft* par rapport à un tel corpus selon quelques facteurs quantitatifs. Nous avons également fait appel, pour certaines mesures quantitatives au moteur `GRASP`, introduit dans le chapitre 4, afin de procéder à l'analyse grammaticale de celles-ci.

7.1.1.1 Méthodologie 1 : comparaison quantitative

Choix des corpus. En ce qui concerne le choix du corpus généraliste, nous avons essentiellement été limités par la disponibilité restreinte de ceux-ci en langue française. Nous nous sommes finalement tournés vers le corpus MULTITAG, introduit dans la section 3.1.2.1, qui correspond tout à fait à la définition et qui était facilement accessible, étant issu d'un projet financé par le CNRS [Paroubek, 2000].
Du côté du corpus *Daft*, cette étude ayant été faite avant l'ajout des FAQ de Word et LaTeX, le corpus considéré correspond plus précisément à $Daft_{app} \cup Daft_{thé}$, c'est-à-dire au corpus collecté dans les expériences Coco, Hanoï et AMI, enrichi par l'usage de thésaurus.

Prétraitement du corpus MULTITAG. Pour que les comparaisons quantitatives soient significatives, il est indispensable de considérer un corpus qui soit de taille équivalente, ce qui n'est clairement pas le cas de MULTITAG pris dans son ensemble qui est constitué de 1 000 000 de mots, contre seulement 42 716 dans $Daft_{app} \cup Daft_{thé}$. Nous avons donc eu recours à deux critères différents afin de sélectionner aléatoirement un sous-ensemble de phrases de MULTITAG : le nombre de phrases et le nombre de flexions. Nous obtenons ainsi deux corpus :

- $Multitag_{phra}$: qui correspond à un sous-ensemble de MULTITAG similaire à $Daft_{app} \cup Daft_{thé}$ en nombre de phrases ;
- $Multitag_{flex}$: qui correspond à un sous-ensemble de MULTITAG similaire à $Daft_{app} \cup Daft_{thé}$ en nombre de flexions.

Le tableau 7.1 permet de comparer quantitativement le corpus $Daft_{app} \cup Daft_{thé}$ à ces deux sous-ensembles du corpus MULTITAG. Ce choix de distinguer deux sous-corpus s'explique par

la nature que l'on suppose très différente des phrases présentes dans MULTITAG par rapport à celles de $Daft_{app} \cup Daft_{thé}$.

Corpus	$Daft_{app} \cup Daft_{thé}$	$Daft_{web}$	$Multitag_{phra}$	$Multitag_{flex}$
Phrases	**5 159**	321	**5 005**	1 460
Caractères	198 555	7 063	781 572	232 706
Mots	42 716	1 645	155 667	46 363
Mots inconnus	554	94	24 190	6 890
Flexions	**47 839**	1 963	160 672	**47 823**
Flexions uniques	3 130	394	14 129	7 148
Flexions inconnues	352	63	8 272	3 620
Lemmes uniques (connus)	1 788	288	2 309	1 841
Lemmes inconnus	-	39	N/A	N/A
Caractères / mots	4,65	4,29	5,02	5,02
Mots / phrase	8,28	5,12	31,1	31,8
Mots inconnus / connus	1,3 %	5,7 %	15,5 %	14,8 %
Flexions inconnues / uniques	11,2 %	16,0 %	58,5 %	50,6 %

Tableau 7.1 Comparaison quantitative du corpus *Daft* au corpus MULTITAG, en fonction de l'analyse des requêtes faite par `GRASP`

7.1.1.2 Résultats : des phrases plus courtes et moins riches

Des phrases plus courtes. La différence entre les phrases issues de *Daft* et de MULTITAG apparait clairement ne serait-ce qu'en considérant un facteur tel que le nombre de mots moyen par phrases : seulement 8,3 pour $Daft_{app} \cup Daft_{thé}$ contre plus de 31 pour MULTITAG. Cela entraîne qu'à nombre de phrases égal, $Multitag_{phra}$ contient plus de trois fois plus de mots que $Daft_{app} \cup Daft_{thé}$. Ainsi, il suffit de 1 460 phrases issues de MULTITAG pour obtenir autant de mots que dans les plus de 5 000 de $Daft_{app} \cup Daft_{thé}$.
On peut également voir l'illustration de ce phénomène sur la figure 7.1 : si les courbes représentant le nombre de mots par phrase des corpus $Daft_{app} \cup Daft_{thé}$ et MULTITAG (dans son ensemble) ont une allure assez similaire (de type loi de Poisson), outre le facteur d'échelle lié à la taille supérieure de MULTITAG, la longueur de phrase la plus commune dans $Daft_{app} \cup Daft_{thé}$ n'est que de 6, quand elle est de 42 dans MULTITAG. En se concentrant sur le sous-ensemble $Multitag_{phra}$, on peut d'ailleurs montrer que si l'on trie les phrases en fonction de leur longueur (obtenant alors une courbe en forme de loi de Zipf), en appliquant un facteur de l'ordre de 3,7 au nombre de lemmes par phrase aux phrases du corpus $Daft_{app} \cup Daft_{thé}$, la courbe obtenue est très proche de celle montrant le nombre de lemmes par phrases pour les quelques 5 000 phrases de $Multitag_{phra}$.

Des phrases moins riches. La propension à utiliser des phrases plus longues se double d'une richesse de vocabulaire clairement plus importante dans MULTITAG : même en se ramenant à un nombre de flexions équivalent dans les deux corpus, le nombre de lemmes

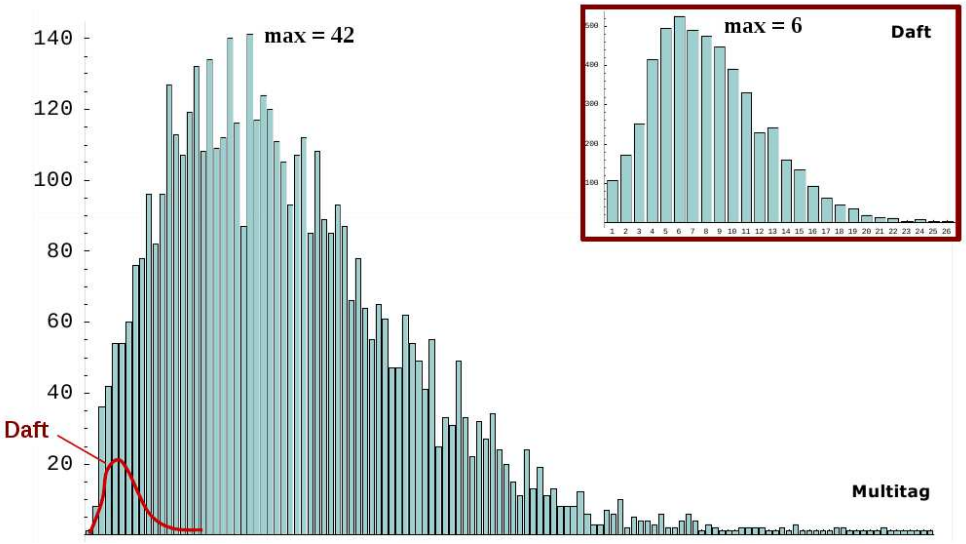


Figure 7.1 Répartition du nombre de mots par phrase dans les corpus *Daft* et MULTITAG

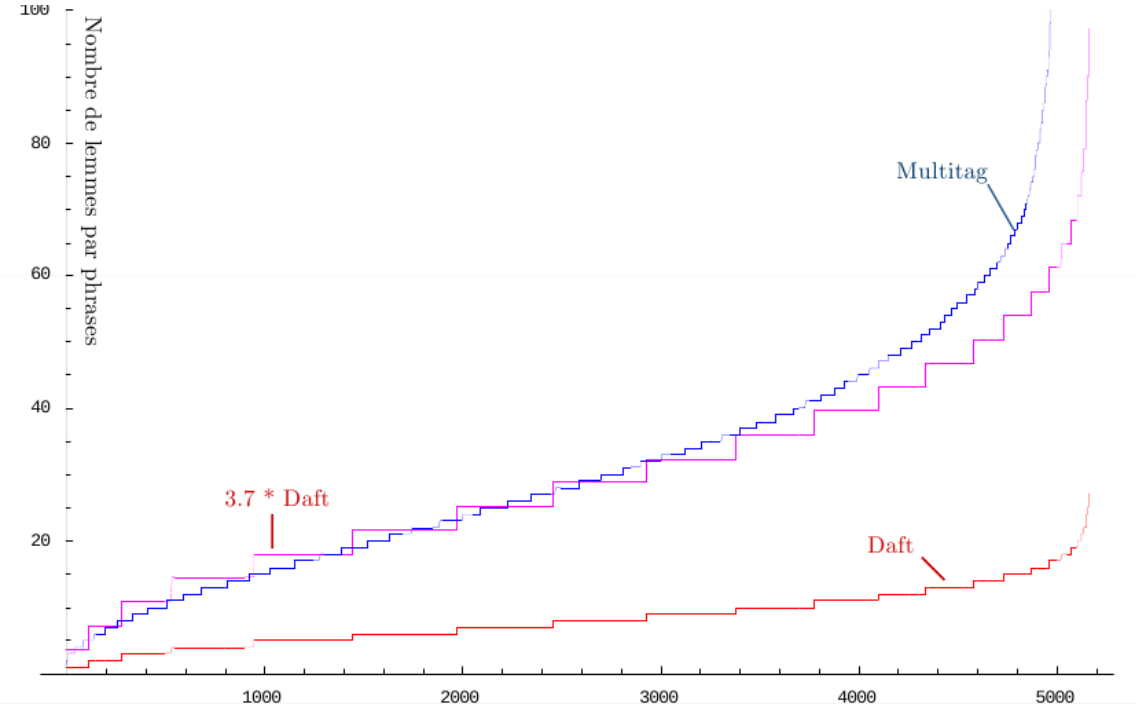


Figure 7.2 Nombre de lemmes par phrases dans les corpus *Daft* et $Multitag_{phra}$ (des plus courtes aux plus longues)

différents est en effet environ deux fois plus important dans MULTITAG (4 082 *vs* 2 006, soit +103%). L'estimation du nombre de lemmes inconnus est faite en considérant que le rapport était à peu près identique au rapport nombre de lemmes connus / nombre de lemmes inconnus mesuré avec le corpus $Daft_{web}$. Cette différence se confirme et est même encore plus importante quand on effectue la comparaison au niveau du nombre de flexions uniques (*i.e.* sans répétition) plutôt qu'au niveau des lemmes (7 148 *vs* 3 130, soit +128%) ce qui signifie que pour un lemme donné, il y a plus de flexions dans MULTITAG que dans *Daft*. Cela peut notamment s'expliquer au niveau des verbes par le fait que certains temps complexes sont beaucoup moins couramment employés dans les requêtes d'assistance que dans l'usage commun (souvent au présent de l'indicatif pour les questions ou à l'impératif pour les ordres).

De manière purement quantitative également, les mots dans MULTITAG sont en moyenne sensiblement plus longs (5,02 caractères contre 4,65 caractères) que ceux employés dans $Daft_{app} \cup Daft_{thé}$.

7.1.1.3 Méthodologie 2 : utilisation de `GRASP`

Une autre manière de traiter les données consiste à considérer la question suivante : en supposant que l'on souhaite que l'outil `GRASP` développé à partir du corpus $Daft_{app}$ soit en mesure de traiter également des requêtes issues d'un nouveau corpus constitué à partir d'une ou plusieurs nouvelles applications assistées, quelle serait la proportion du nombre de lemmes nouveaux ou de flexions nouvelles à introduire ?
C'est dans cette optique que nous avons considéré les conséquences pour `GRASP` de l'ajout, lors de sa constitution, du sous-corpus $Daft_{web}$ (constitué à partir d'une application), dont les statistiques similaires à celles considérées jusqu'à maintenant pour $Daft_{app} \cup Daft_{thé}$ et MULTITAG figurent également dans le tableau 7.1.

7.1.1.4 Résultats : un registre de langue limité ?

La proportion de mots inconnus à ajouter est modeste (5,7%). En termes d'extension du lexique déjà utilisé par `GRASP`, la conséquence est encore moins sensible puisqu'il suffit d'ajouter seulement 39 nouveaux lemmes (soit 2,1% par rapport au nombre de lemmes qui y figurent déjà). Au contraire, si l'on essaye de faire la même opération pour augmenter le lexique permettant de traiter les phrases issues de MULTITAG, il faudrait ajouter en proportion trois fois plus de mots (14,8%) ou lemmes (15,5%).
Cette observation est un premier indice indiquant que le corpus *Daft*, tel qu'il est constitué, par ajouts successifs de sous-corpus, tend déjà à saturer le domaine de langue que constituent les requêtes d'assistance, puisque la proportion de nouveaux mots est significativement plus faible en considérant de nouvelles requêtes d'assistance qu'en considérant des phrases quelconques.

7.1.2 Comparaison à des corpus orientés tâches

Nous avons émis l'hypothèse, dans le chapitre 3, que l'assistance était un domaine suffisamment particulier pour que son étude requière la constitution d'un corpus dédié. Si de toutes façons il est utile de disposer d'un corpus dont on connaît et maîtrise les conditions de recueil, il demeure utile a posteriori également de poser la question des spécificités propres à ce corpus. Ceci passe par la comparaison avec d'autres corpus qu'on peut estimer proches au niveau du type de requêtes qu'ils contiennent. En effet, si l'on parvient à trouver des différences, elles permettront de caractériser les requêtes d'assistance selon des critères qui pourront se révéler utiles lorsqu'il s'agira de les analyser. À l'inverse, si l'on met en évidence l'absence de différences entre le corpus *Daft* et d'autres corpus dialogiquement proches, cela nous autoriserait à fusionner les deux de manière à étendre encore la couverture de *Daft*.
Dans cette section, nous allons plus précisément travailler sur la base du corpus *Daft* tel qu'il était au moment de cette étude, c'est-à-dire constitué par $Daft_{app} \cup Daft_{web} \cup Daft_{thé}$.

7.1.2.1 Méthodologie

Pour comparer des corpus, il faut en premier lieu définir des critères de comparaison adaptés par rapport à l'utilisation que l'on envisage de faire des corpus en question. Ici, notre objectif étant à terme de produire une représentation formelle de la sémantique (voire de la pragmatique) des requêtes en vue d'un traitement par un ACA, une approche purement quantitative comme on l'a fait pour distinguer *Daft* de MULTITAG dans la section 7.1.1 n'est plus vraiment appropriée. La plupart des travaux dans le domaine de la comparaison de corpus se basent sur la présence de mots-clés similaires [Kilgarriff, 2001] ou plus généralement sur une analyse purement grammaticale du contenu des corpus [Rayson & Garside, 2000]. Une approche plus pertinente ici nous semble être de se concentrer sur une comparaison statistique en termes d'actes de dialogue, à la manière des travaux de Ripoche [2006] ou Twitchell *et al.* [2004] sur lesquels nous nous basons.

La notion de profils interactionnels. Le concept de profil interactionnel est défini par Ripoche comme "la distribution des actes de dialogue présents dans une unité d'interaction donnée" où "une unité [...] sera définie en fonction de l'objectif de l'analyse" [Ripoche, 2006, p. 110]. Une fois que l'unité d'interaction a été choisie, on calcule la proportion de chaque acte de dialogue (dans la taxonomie choisie) de manière à pouvoir présenter ledit profil sous la forme d'un histogramme de manière à disposer d'une vision synthétique associée à l'unité d'interaction. L'intérêt des profils interactionnels ne réside clairement pas tant dans leur valeur intrinsèque que dans la possibilité qu'ils offrent de comparer deux unités d'interaction (un tour de dialogue, un dialogue, un corpus de dialogues, *etc.*).

Cette approche n'est pas très éloignée du modèle développé par Twitchell *et al.* [2004], les principales différences résidant plutôt dans le contexte d'application que dans la manière

de l'appliquer. En effet, l'approche à base de profils interactionnels est :

- discrète plutôt que probabiliste : Twitchell *et al.* [2004] considèrent qu'un élément d'une unité peut être lié à plusieurs actes de dialogue avec une certaine probabilité associée, tandis qu'on préfère ici utiliser une identification des actes de dialogues réalisée :
  - par annotation manuelle : pour plus de précision et afin de ne pas introduire de bruit dans les données à cause d'erreurs liées aux processus d'identification automatique ;
  - unique : ce qui est en partie lié au point précédent, car il est plus facile d'identifier précisément et avec certitude un acte de dialogue lors d'une annotation par un humain.
- plus collective qu'individuelle : là où Twitchell *et al.* [2004] s'intéressent à l'étude des interactions d'une personne, les profils interactionnels étudient plutôt une tendance de comportement générale issue de l'interaction de plusieurs utilisateurs différents avec le système ;
- non normative : Twitchell *et al.* [2004] soustraient les profils qu'ils obtiennent d'un profil type global qui est supposé représenter une certaine "norme" pour évaluer la façon dont une interaction peut dévier du profil type.

Choix des corpus à comparer. Parmi les corpus disponibles et présentés dans la section 3.1.2.2 et la table 3.2 auxquels nous pouvons comparer le corpus *Daft*, nous nous sommes restreints à trois d'entre eux, qui présentent l'avantage d'avoir été annotés en terme d'actes de dialogue selon un codage précis :

- Switchboard de Godfrey & Holliman [1997] annoté par [Jurafsky *et al.* , 1998] : 1 155 retranscriptions de dialogues téléphoniques représentant environ 200 000 phrases.
- Map Task de Anderson *et al.* [1991] et annoté par [Carletta *et al.* , 1996] : 128 dialogues au cours desquels une personne doit reproduire un itinéraire sur une carte en suivant des instructions données par une personne dotée d'une carte similaire.
- Bugzilla, annotation automatique en termes d'actes de dialogues par Ripoche [2006] de 1 200 000 commentaires issus de 128 000 rapports de bogues créés au cours du développement de la suite logicielle de la fondation Mozilla.

Switchboard et MapTask proviennent d'interactions orales et sont donc naturellement plus riches en termes de mots qu'un corpus qui aurait été récolté par écrit dans des conditions par ailleurs identiques [Kelly & Chapanis, 1977], mais la proximité des activités nous est apparue plus importante pour cette comparaison que cette différence d'origine. En ce qui concerne la différence de langue (ces trois corpus étant en anglais), nous ne pensons pas qu'elle puisse être significative dans la mesure où nous plaçons, dans cette étude, au niveau des actes de dialogue, et que les intentions communicatives sont a priori comparables entre des francophones et des anglophones[1].

Enfin, d'autres corpus introduits en 3.1.2.2 seraient aussi intéressants à contraster avec *Daft*,

[1] L'étude des différences interculturelles entre utilisateurs novices sort du cadre de cette thèse.

tels que GOCAD ou TRAINS, tous les deux également recueillis en vue de concevoir un agent logiciel. Toutefois, ceux-ci n'étant pas annotés en termes d'actes de dialogue, il serait très coûteux de devoir les analyser au préalable afin, dans un premier temps, de définir un schéma de codage des actes de dialogues propre à ces corpus, puis, dans un second temps, de les annoter à l'aide de celui-ci.

La correspondance des taxonomies. L'unité d'interaction choisie pour notre étude est le corpus dans son ensemble. Toutefois, les trois corpus considérés ont chacun été annotés avec une taxonomie d'actes de dialogues qui leur est propre, et il est donc nécessaire de les ramener dans un premier temps à une taxonomie commune. Nous avons donc eu recours à la taxonomie formée de cinq actes définie par Searle [1969] (*cf.* section 1.2.3.2), qui est suffisamment générique pour servir de base de comparaison et que nous nommerons T_S. Il suffit dès lors d'établir un ensemble de règles de conversion pour être en mesure de disposer de trois corpus annotés avec cette taxonomie commune.

Switchboard : Annoté à l'origine selon quatre dimensions, la combinaison entre ces dimensions étant relativement limitée, il est en fait possible de distinguer 42 catégories principales dans la taxonomie DAMSL [Jurafsky *et al.* , 1998] utilisée pour annoter Switchboard, et ce sont celles que nous avons utilisé ici. Certains actes de dialogue sont toutefois très spécifiques (par exemple le "self-talk") et non triviaux à convertir dans une taxonomie générale telle que T_S.

MapTask : Ce corpus a été à l'origine annoté selon une taxonomie de 13 actes qui peuvent être relativement facilement associés aux actes de T_S. À titre d'exemple, la table 7.2 présente les règles utilisées pour effectuer cette conversion MapTask vers la taxonomie commune. On voit ici qu'il a toutefois été nécessaire d'ajouter à T_S une sixième catégorie, "Inconnus", qui correspond aux actes de dialogue des taxonomies originelles qui ne pouvaient pas être aisément convertis vers un des cinq actes de T_S.

$\mathbf{T}_S$	Assertifs	Promissifs	Directifs	Expressifs	Déclaratifs	Inconnus
MapTask	clarify explain reply-w	-	align check instruct query-w query-yn	acknowledge ready reply-n reply-y	-	uncodable

Tableau 7.2 Règles de conversion d'actes de dialogue de la taxonomie de MapTask vers la taxonomie searlienne T_S

Bugzilla : Initialement étudié par Ripoche [2006], le corpus Bugzilla avait été directement annoté dans la taxonomie T_S donc aucune conversion particulière n'a été nécessaire.

***Daft* :** En ce qui concerne le corpus *Daft* lui-même, nous avons procédé à une annotation directement dans la taxonomie commune T_S du sous-corpus $Daft_{sub}$ de 1 074 phrases, tel que défini en section 3.3.3.2.

7.1.2.2 Résultats

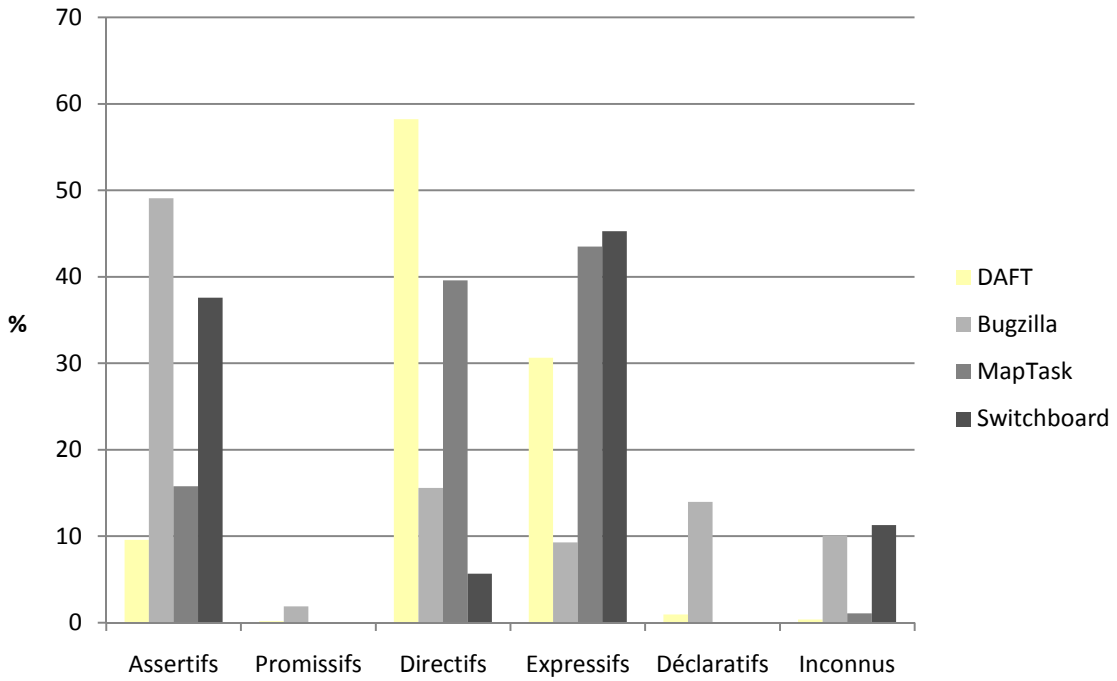


Figure 7.3 Profils interactionnels de quatre corpus dans la taxonomie T_S des actes de dialogue de Searle

De part les conversions forcément imparfaites et l'existence même d'une classe "Inconnus", quand bien même elle est de taille réduite (environ 4% en moyenne), les profils interactionnels que l'on peut ainsi obtenir et qui apparaissent sur la figure 7.3 sont forcément à considérer avec précaution pour ne pas surinterpréter certains micro-phénomènes. Néanmoins, des tendances assez nettes se dessinent qui permettent de distinguer le corpus *Daft* des trois autres corpus :

– *Daft* a une majorité de directifs (58%), qui s'explique par le grand nombre d'ordres (faire-faire et faire devoir dans la taxonomie de Caelen [2003]) et de questions (faire faire-savoir) à l'agent. Bien que MapTask et Bugzilla soient également des corpus orientés tâche, les interactions ont uniquement lieu entre des humains, et il semble donc que de s'adresser à un ordinateur, même au travers d'un agent conversationnel, rend les requêtes plus directes. On peut penser que cela est dû au fait que les utilisateurs font implicitement l'hypothèse que l'agent n'est pas en mesure de réaliser des inférences aussi complexes que des humains, et jugent donc plus logique d'user d'un mode d'expression plus direct

et explicite. C'est aussi un signe de l'hypothèse forte selon laquelle le système est là pour "servir" (au sens maître-esclave) l'utilisateur.

- la proportion d'assertifs dans *Daft* est relativement faible (9%), les utilisateurs préférant exprimer leur ressenti et leur état d'esprit (31%) vis-à-vis de la situation dans laquelle ils se trouvent plutôt que de présenter les faits de manière neutre et impersonnelle comme ils peuvent essayer de le faire lorsqu'il s'agit d'un rapport de bogue dans le corpus Bugzilla. Cela peut être lié au stress cognitif provoqué par le fait de se retrouver dans une situation où l'on ne sait pas comment effectuer la tâche à accomplir.
- très peu de promissifs ($< 1\%$) sont observés dans *Daft*, ce qui est sans nul doute lié à la nature même de la relation utilisateur-agent, où l'agent est implicitement considéré par l'utilisateur comme un subordonné. Par conséquent, il est logique que l'utilisateur ne se sente pas vraiment dans une situation où il doit s'engager à effectuer ce que l'agent lui suggère, et quand bien même il le fait, cet engagement est implicite et rarement exprimé de manière verbale (ex : "je te promets que je vais réfléchir avant de jouer").
- les déclaratifs sont quasiment absents ($< 1\%$) du corpus *Daft*, car en général l'utilisateur fait l'action sans juger nécessaire de l'annoncer pour l'effectue (par exemple, il arrête d'interagir avec l'agent plutôt que de lui dire comme on a pu le relever une fois "à partir de maintenant, je vais me débrouiller toute seule").

Il apparaît donc que notre hypothèse initiale est confirmée : le corpus *Daft* se distingue en termes de fréquence des différents actes de dialogue d'autres corpus proches, ce qui conforte l'idée que la constitution d'un corpus spécifique constituait une étape préalable indispensable.

7.2 Analyse conversationnelle manuelle du corpus *Daft*

7.2.1 Motivation de l'analyse

Pour rappel, lors de la phase de recueil du corpus *Daft*, les sujets humains étaient informés qu'ils devaient réaliser un certain nombre de tâches pour lesquelles ils pouvaient, si nécessaire, faire appel à un agent conversationnel intégré au programme capable de les assister. Les sujets demeuraient néanmoins entièrement libres pendant leur interaction avec le système d'avoir ou non recours à l'agent, et lorsqu'ils le faisaient, ils étaient également libres dans la formulation de leurs requêtes à celui-ci, tant sur le fond (contenu sémantique) que sur la forme (formulation employée).

Or des comportements d'interaction différents sont ainsi apparus, et une simple lecture de quelques phrases ainsi recueillies montre immédiatement que des utilisateurs ont même parfois (au moins temporairement) complètement abandonné leur tâche d'origine. Ainsi, il apparaît que dans de nombreux cas (*cf.* tableau 7.3), les phrases recueillies n'étaient pas vraiment liées au domaine de l'assistance qui nous intéresse prioritairement. C'est pour cette raison que nous nous sommes intéressés à qualifier et à quantifier ces différents types de requêtes apparaissant dans notre corpus.

7.2.2 Définition de la notion d'activité conversationnelle

Nous nommerons par la suite "activité conversationnelle" la représentation du premier degré de l'intentionnalité communicative de l'utilisateur, c'est-à-dire ce qui pourrait constituer le premier niveau d'actes de dialogue dans une classification à plusieurs niveaux. La définition de cette classification d'activités conversationnelles est clairement liée au contexte d'énonciation, qui correspond ici à une interaction avec un agent visant à assister l'utilisateur. Par ailleurs il s'agit d'une classification réalisée a posteriori, à partir du corpus de requêtes recueillies.

7.2.3 Méthodologie d'étude

7.2.3.1 Principes

Afin d'étudier les différentes activitiés conversationnelles du corpus *Daft*, nous avons travaillé à partir de deux sous-ensembles de phrases extraits aléatoirement du sous-corpus de phrases collectées ($Daft_r$), $Daft_{sub_1}$ et $Daft_{sub_2}$ (définis en section 3.3.3.2) dans la mesure où les phrases issues de $Daft_e$ augmentent la couverture en termes de requêtes d'assistance, mais faussent les proportions de celles-ci au sein du corpus total *Daft*.
Ces phrases ont été annotées manuellement par un annotateur unique de manière successive : tout d'abord, l'annotation de $Daft_{sub_1}$ a permis d'établir de manière empirique les distinctions entre requêtes, de manière à systématiser un protocole d'annotation (décrit dans la section 7.2.3.2). Ce protocole a ensuite été rigoureusement suivi pour procéder à l'annotation du deuxième sous-ensemble, $Daft_{sub_2}$.
Un extrait (non représentatif) de phrases issues de $Daft_{sub_1}$ et $Daft_{sub_2}$ est donné dans la table 7.3 dans le seul but de servir de base d'exemples pour illustrer les différentes activités conversationnelles décrites.

7.2.3.2 Protocole d'annotation

La première question que l'on est amené à se poser est de savoir si l'utilisateur demeure dans le contexte de l'accomplissement d'une tâche (a priori, celle donnée en objectif), ou si, au contraire, il rentre dans une discussion qui n'est plus tant centrée sur l'application que sur l'agent conversationnel lui-même. On établit donc un premier niveau de classification par rapport au contenu thématique de la requête :

- les **requêtes orientées tâche** : dans lesquelles l'utilisateur œuvre de manière plus ou moins directe à se rapprocher d'un état particulier de l'application qu'il souhaite atteindre, indépendamment de savoir si, se faisant, il parvient effectivement à se rapprocher de cet état ou non (ex : phrases 1 à 9 du tableau 7.3) ;
- les **requêtes orientées discussion** : où l'objectif de l'utilisateur n'est plus lié à l'application mais plutôt l'intérêt même d'une interaction avec l'agent en tant qu'entité

N°	Requête d'utilisateur
1	appuies sur le bouton quitter
2	clickersur le bouton back
3	bon, reviens à l apage d'accueil
4	a quoi sert cette fenêtre,
5	c quoi le GT ACA
6	le bouton "fermer" et le bouton "quitter" ont exactement le même fonctionnement ?
7	je ne vosi aucune page de demso ! !
8	j'ai été surprise qu'il manque une fonction d'annulation globale
9	ça serait mieux si on pouvait aller directement au début
10	ca marche :-)
11	non alors !
12	ah bon ?
13	je comprend rien a ce que t'a dit
14	réponds-moi stp
15	hello comment ça va ?
16	auf viedersen
17	marco ?
18	je n'ai plus besoin d'aide
19	et si on se disait vous ?
20	Quel genre de musique tu aimes ?
21	j'aime tes cheveux Léa
22	bon à rien !
23	ne me force pas à te tuer !
24	La présentation est élégante
25	je suis ce que je suis

Tableau 7.3 Sous-ensemble de requêtes réelles issues de $Daft_{sub}$, sélectionnées de manière à représenter différentes activités conversationnelles

supposée intelligente et dont on s'attend à ce qu'elle exhibe certains comportements humains, et en particulier une capacité à répondre à des questions personnelles ou de sens commun (ex : phrases 10 à 25).

Dans le contexte de l'étude de la fonction d'assistance, c'est clairement plutôt la première catégorie qui nous intéresse. Au sein de celle-ci, la seconde distinction à effectuer concerne donc le fait de savoir si la requête liée à la tâche vise à l'accomplissement direct de celle-ci ou si elle correspond à une utilisation de l'agent dans son rôle premier d'assistant. Plus précisément, on distingue :

- les **requêtes de contrôle** : où l'utilisateur fait appel à l'agent en tant que médiateur pour interagir avec l'application qu'il assiste. Ainsi, au lieu d'effectuer lui-même les actions à l'aide de la souris ou du clavier en faisant usage de l'interface graphique traditionnelle de l'application, l'utilisateur considère que la présence de l'agent implique la possibilité pour lui d'interagir de manière dialogique avec l'application qu'il assiste (et donc que l'agent a la possibilité de transmettre ce genre de requêtes à l'application – ex : phrases 1 à 3) ;
- les **requêtes d'assistance** : qui sont au cœur de notre étude et sont donc celles que l'on souhaite être capable de traiter en priorité (ex : phrases 4 à 9).

Au sein même des requêtes d'assistance, les demandes d'aide sont plus au moins directes, et peuvent donc parfois nécessiter un traitement plus complexe lorsqu'il est nécessaire d'expliciter les intentions réelles de l'utilisateur, parfois exprimées de manière implicite. Nous avons donc subdivisé celles-ci en :

- **requêtes d'assistance directes** : dans lesquelles une question est explicitement posée par rapport à un élément ou un processus de l'application (ex : phrases 4 à 6) ;
- **requêtes d'assistance indirectes** : qui regroupent des jugements d'utilisateurs au sujet de l'application qui seraient interprétés par des assistants humains comme un besoin d'assistance ; elles nécessitent donc une couche de prise en compte de la pragmatique au niveau du système pour détecter la signification implicite[2] (ex : phrases 7 à 9).

En ce qui concerne les requêtes orientées discussion, bien qu'elles soient moins en rapport avec notre objectif, devant leur importance quantitative au sein de $Daft_{sub_1}$ (*cf.* section 7.2.4), nous avons toutefois souhaité détailler leur subdivisions. Plus particulièrement, on distingue en fonction du thème de la discussion :

- les **réactions à une réponse** de l'agent : c'est-à-dire l'ensemble des façons d'exprimer :
  - son accord ou son désaccord (ex : phrases 10 et 11),
  - son incrédulité (ex : phrase 12),
  - son incompréhension (ex : phrase 13),
  - son insistance (ex : phrase 14).
- les **fonctions communicatives** : correspondant à l'ensemble des formules :

[2] Notons que dans ce type de requêtes, l'utilisateur va à l'encontre des maximes de Grice (*cf.* section 1.2.3.3) selon lesquelles son intention devrait être exprimée de manière claire et non ambigüe.

  - permettant d'initier ou de mettre fin à l'interaction avec l'agent (ex : phrases 15 et 16),
  - de type phatiques[3] (ex : phrase 17).
- le **dialogue avec l'agent** : qui regroupe des requêtes où l'agent est explicitement le centre d'intérêt de l'utilisateur. Il peut s'agir :
  - d'un **ordre à l'agent lui-même**, au sujet du comportement de l'agent, par opposition aux requêtes visant à contrôler l'application (ex : phrase 19) ;
  - d'une **question personnelle à l'agent** sur ses capacités, son histoire ou ses opinions (ex : phrase 20) ;
  - d'un **avis sur l'agent** : qui peut être connoté positivement, c'est alors un compliment (ex : phrase 21), ou négativement, et c'est alors une critique (ex : phrase 22) ;
  - d'une **menace**, visant à renforcer ou à réaffirmer l'autorité que l'utilisateur ressent naturellement sur l'agent (ex : phrase 23).
- les **commentaires sur l'application** : qui n'ont pas de valeur d'assistance, même indirecte (ex : phrase 24) ;
- les **autres requêtes de discussion** : qui sont un mélange de requêtes difficiles à interpréter sans une prise en compte précise du contexte (ex : phrases 25).

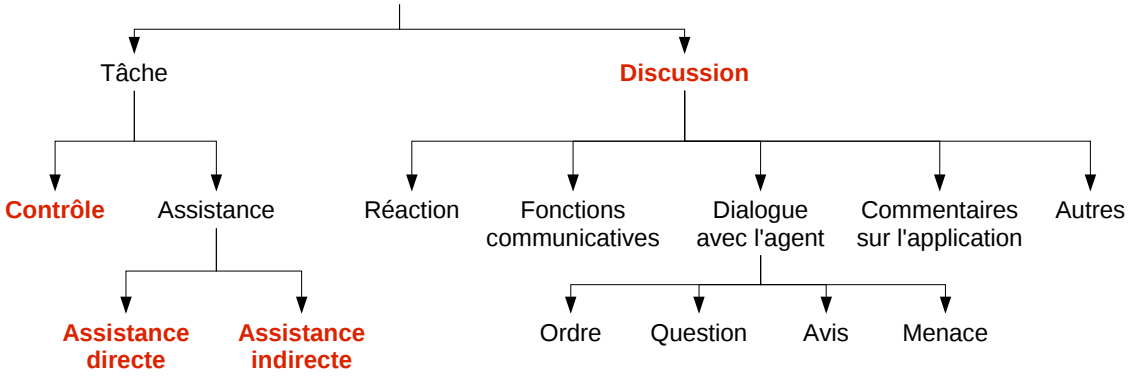


Figure 7.4 Classification des activités conversationnelles du corpus *Daft* (en gras et rouge : les activités conversationnelles majeures)

Ces subdivisions sont synthétisées dans la classification d'activités conversationnelles de requêtes donnée dans la figure 7.4. En associant à chaque nœud de l'arbre une question permettant de discriminer ses branches, on obtient le questionnaire de la figure 7.5 utilisé pour l'annotation en activités conversationnelles du sous-ensemble $Daft_{sub_2}$.
On retiendra en particulier quatre activités conversationnelles que l'on qualifiera de majeures par rapport à l'intérêt qu'on leur porte dans le cadre de cette thèse : le contrôle, l'assistance directe, l'assistance indirecte et la discussion ou clavardage (considérée de manière indistincte par la suite).

[3]Expression ayant une fonction de pure civilité et pas de valeur sémantique propre (*e.g.* s'il vous plaît, allô, *etc.*).

La requête permet de progresser dans la réalisation d'une tâche au sein de l'application ?
Oui
↪La requête permet l'accomplissement immédiat d'une action ?
Oui ⇒ **Contrôle**
Non
↪ La demande d'assistance est évidente et exprimée de manière directe ?
Oui ⇒ **Assistance directe**
Non ⇒ **Assistance indirecte**
Non ⇒ **Discussion**
↪ La requête a pour sujet l'application ?
Oui ⇒ Commentaire sur l'application
Non
↪La requête permet d'entamer, de clore ou de maintenir le contact avec l'agent ?
Oui ⇒ Communication
Non
↪La requête exprime une opinion par rapport à une requête antérieure ?
Oui ⇒ Réaction
Non
↪ La requête a pour sujet l'agent ?
Oui ⇒ Dialogue avec l'agent
Non ⇒ Autres

Figure 7.5 Questionnaire utilisé pour l'annotation en termes d'activités conversationnelles du sous-ensemble $Daft_{sub_2}$ du corpus *Daft*

7.2.4 Résultats

7.2.4.1 Répartition des différentes activités conversationnelles

	Activité conversationnelle				
Sous-corpus	Contrôle	Assistance directe	Assistance indirecte	Discussion	Total
$Daft_{sub_1}$	82	190	51	199	522
$Daft_{sub_2}$	75	200	43	234	552
Total	157	390	94	433	1 074

Tableau 7.4 Répartition des différentes activités conversationnelles au sein de $Daft_{sub}$

Les résultats de l'annotation des deux sous-ensembles sont donnés dans le tableau 7.4. Les deux sous-ensembles étant issus de tirages aléatoires sans remise de requêtes dans $Daft_r$, il est logique de penser que leur répartition est équivalente. Toutefois, comme la première annotation a été faite avant l'établissement du protocole, nous avons vérifié que c'était bien le cas en considérant l'hypothèse nulle H_0 supposant que les deux répartitions sont équivalentes. Il est alors possible de calculer les données permettant de vérifier cette hypothèse (*cf.* tableau 7.5) : la probabilité ainsi obtenue (3,34) étant inférieure à celle du χ^2 à 5% (7,81), on peut en déduire que l'hypothèse H_0 tient. Les répartitions sont bien équivalentes : il n'est donc pas nécessaire de procéder à une nouvelle annotation de $Daft_{sub_1}$ et on peut les fusionner pour obtenir la

	Activité conversationnelle				
Sous-corpus	Contrôle	Assistance directe	Assistance indirecte	Discussion	Total
$Daft_{sub_1}$	76,2	189,4	45,6	210,8	522,0
$Daft_{sub_2}$	80,8	200,6	48,4	222,2	552,0
Total	157,0	390,0	94,0	433,0	1 074,0

Tableau 7.5 Répartition théorique pour le calcul de l'hypothèse nulle H_0

répartition des requêtes dans $Daft_r$, donnée par la figure 7.6.

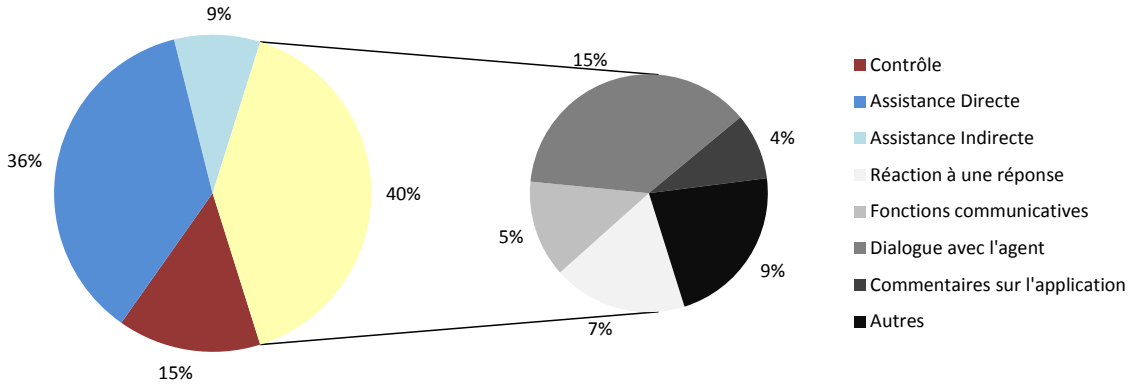


Figure 7.6 Répartition des activités conversationnelles au sein de $Daft_r$

7.2.4.2 Analyse et interprétation

Les résultats de la sous-section précédente montrent que non seulement l'utilisateur n'interagit pas seulement avec l'agent pour obtenir de l'assistance (encore une fois, en dépit des instructions), mais que l'assistance, bien que demeurant l'activité principale (45% des requêtes), ne représente même pas la moitié des requêtes. On peut émettre plusieurs hypothèses permettant d'interpréter ce phénomène.

Une faible motivation à accomplir les tâches. On peut penser qu'un biais existe de part le fait que les demandes d'assistance des utilisateurs ne révèlent peut-être pas tant un besoin profond qu'ils pourraient avoir naturellement pour réaliser une tâche qui leur tient personnellement à cœur, qu'une volonté de suivre les consignes. Il est donc probable qu'en situation de besoin critique d'assistance, un utilisateur novice aura moins tendance à partir dans une discussion annexe avec l'agent. Toutefois, les proportions sont telles qu'il semble extrêmement probable que les autres activités soient souvent présentes, et leur existence même mérite une discussion.

Un besoin d'un agent agissant sur l'application. L'usage de requêtes de contrôle démontre que l'utilisateur n'attend pas seulement de l'agent qu'il soit capable de l'assister dans

son utilisation de l'application, mais aussi qu'il désire également que l'agent soit capable d'agir sur cette même application.

Un besoin de communication hors tâche. De la même manière, la très forte présence de requêtes de discussion prouve que l'utilisateur souhaite aussi que l'agent fasse preuve d'un comportement de type "humain", en étant capable de réagir à des requêtes ou même de simples commentaires qui sortent du cadre restreint donné par l'application et l'ensemble fini de tâches qui peuvent être accomplies dans celle-ci.

Un effet de l'incarnation ? On peut estimer que le comportement décrit dans le paragraphe précédent est sans nul doute lié à l'usage d'un agent personnifié (c'est-à-dire représenté visuellement à l'écran par un personnage graphique). En effet, la présence tangible apportée par la représentation graphique qui fournit une incitation à considérer l'agent comme un être humain et donc à lui poser des questions que l'on ne penserait pas nécessairement à poser si on ne disposait que d'un simple champ texte pour saisir les requêtes. Ce phénomène est à relier avec le concept d'affordance proposé en psychologie par Gibson [1977].

Conséquences pratiques sur le projet DAFT. Bien qu'intéressant psychologiquement parlant, ce résultat pose un problème par rapport à notre démarche qui vise à restreindre la complexité du traitement de la langue naturelle en travaillant sur le sous-domaine particulier de l'assistance. En effet, pour être capable de traiter de manière probante les requêtes de discussion, il faut que l'agent constitue une véritable interface dialogique capable de gérer un vocabulaire très large, doté d'informations personnelles sur de nombreux domaines de la vie quotidienne, d'une opinion sur les divers sujets pouvant être abordés dans une conversation courante, *etc.*

D'un point de vue pratique, nous continuerons dans la suite de cette thèse à mettre l'accent sur le traitement des requêtes de contrôle et d'assistance (directe ou indirecte). Nous prendrons aussi en compte des requêtes témoignant d'un retour de l'utilisateur sur une réponse précédente de l'agent, dans la mesure où elles ne remettent pas réellement en cause le choix de traiter les requêtes de manière isolées, puisque la mémorisation d'un unique tour de dialogue suffit pour les traiter correctement.

7.3 Vers une classification automatique de requêtes selon leur activité conversationnelle

Nous nous proposons dans cette section d'explorer différentes approches en vue d'une identification automatique de l'activité conversationnelle d'une requête, de manière indépendante du sens de la requête telle qu'il peut être fourni par l'analyse sémantique de celle-ci. Tout au

long de cette section, les études réalisées l'ont été à partir du sous-ensemble $Daft_{sub}$ de 1 074 phrases du corpus *Daft*. Lorsque nous utiliserons des approches nécessitant de subdiviser notre ensemble d'étude en 10 ensembles de tailles égales, nous considérerons simplement 1 070 des 1 074 requêtes.

7.3.1 Motivation

L'intérêt de l'analyse du corpus en termes d'activités conversationnelles va au-delà de la comparaison de corpus. En effet, disposer de l'intention communicationnelle générale de l'utilisateur est une information très utile à l'agent conversationnel de manière à élaborer sa réponse : soit pour un prétraitement, de manière à diriger la requête entrante vers des modules différents lorsqu'il s'agit de traiter une requête de contrôle (un simple ordre à exécuter), d'assistance (nécessitant une analyse plus poussée des intentions de l'utilisateur) ou de discussion (où un module de type chatbot serait peut-être plus adapté qu'une analyse profonde de la requête).
Ce pourrait aussi être une information utile en dernier recours : en effet, une méthode classique dans les systèmes de chatbot et qui a fait le succès d'ELIZA [Weizenbaum, 1966] est le recours à une réponse dite "évasive", choisie aléatoirement parmi un ensemble de réponses vagues, et qui est destinée à masquer le fait que le contenu propositionnel de la requête utilisateur n'a pas pu être compris. Perçu comme un manque d'interêt ou un signe de distraction, le recours à ce type de réponse, s'il n'est pas trop fréquent, peut même renforcer l'impression d'avoir affaire à un être humain. Une analyse selon des critères non sémantiques de la requête utilisateur, si elle parvenait à déterminer suffisament efficacement la classe d'activité conversationnelle (selon la classification présentée en 7.2.3.2), permettrait d'affiner ce type de réponse évasive.

Exemples de réponses évasives de l'agent qui prennent en compte l'activité conversationnelle de l'utilisateur :

- À une requête de Contrôle : "Fais-le toi-même", "Je n'ai pas envie de faire ça pour l'instant", "Toujours des ordres. . . ", . . .
- À une requête d'Assistance : "J'aimerais pouvoir vous aider", "De l'aide pour faire quoi ?", "Que dois-je expliquer à ce sujet ?", . . .
- À une requête de Discussion : "Je préfèrerais qu'on reste dans le cadre de l'application", "Restons concentrés sur l'application", . . .

7.3.2 Classification en fonction de paramètres globaux : la requête comme un tout

Dans cette section, nous nous proposons d'étudier la requête langagière en tant qu'ensemble de mots ou d'entités exprimées en *DAFT 2.0*, dont on peut mesurer quelques paramètres statistiques simples. Nous explorons également la possibilité de combiner ces différents

paramètres, et faisons de nouveau appel aux profils interactionnels introduits précédemment dans ce chapitre.

7.3.2.1 Longueur des requêtes en langue naturelle

Un des facteurs les plus simples à considérer dans une requête en langue naturelle pour établir des statistiques est le nombre de mots, avant même qu'elle ne soit analysée par `GRASP`. Une rapide observation de la distribution de ces requêtes selon leur activité conversationnelle (*cf.* tableau 7.6) révèle une différence sensible de longueur (en mots). En particulier, les requêtes de contrôle contiennent globalement moins de mots que les requêtes d'assistance. Toutefois, si les distributions correspondantes peuvent être approximées par des distributions normales (test d'adéquation du χ^2 avec un seuil de 1%), les écarts-types ($\sigma \approx 3.5$) sont trop importants pour que cette mesure puisse, seule, fournir un moyen de discrimination suffisamment précis et fiable pour permettre la classification des requêtes.

Activité	Contrôle	Assistance directe	Assistance indirecte	Discussion
Moyenne $\overline{x}$	5,44	8,01	9,90	6,01
Écart-type σ	3,36	3,54	3,30	3,62

Tableau 7.6 Nombre de mots par requête en fonction de son activité conversationnelle

7.3.2.2 Paramètres basiques des requêtes formelles

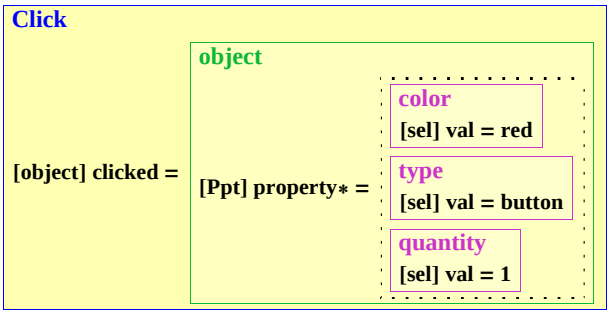


Figure 7.7 Représentation formelle *DAFT* de la requête R_{ex_1} "Clique le bouton rouge"

Une fois les requêtes mises sous forme formelle par `DIG`, il est possible de considérer des paramètres supplémentaires par rapport à lorsqu'elles se trouvaient sous leur forme en langue naturelle. Ainsi, dans leur représentation formelle en *DAFT* on peut notamment mesurer :

– leur **longueur** : le nombre d'entités au plus haut niveau de la requête formelle ; *e.g.* sur la figure 7.7, il y a un unique élément à la racine (`Click`) donc la longueur de la requête formelle est de 1.

– leur **profondeur** : le nombre maximum d'entités imbriquées les unes dans les autres ; *e.g.* sur la figure 7.7, la profondeur est de 3 (`Click[...object ...[ ...type ...] ...]`).
– le **nombre d'entités** : indépendamment de la façon dont elles sont structurées ; *e.g.* sur la figure 7.7, on compte 5 entités (`Click`, `object`, `color`, `type`, `quantity`).

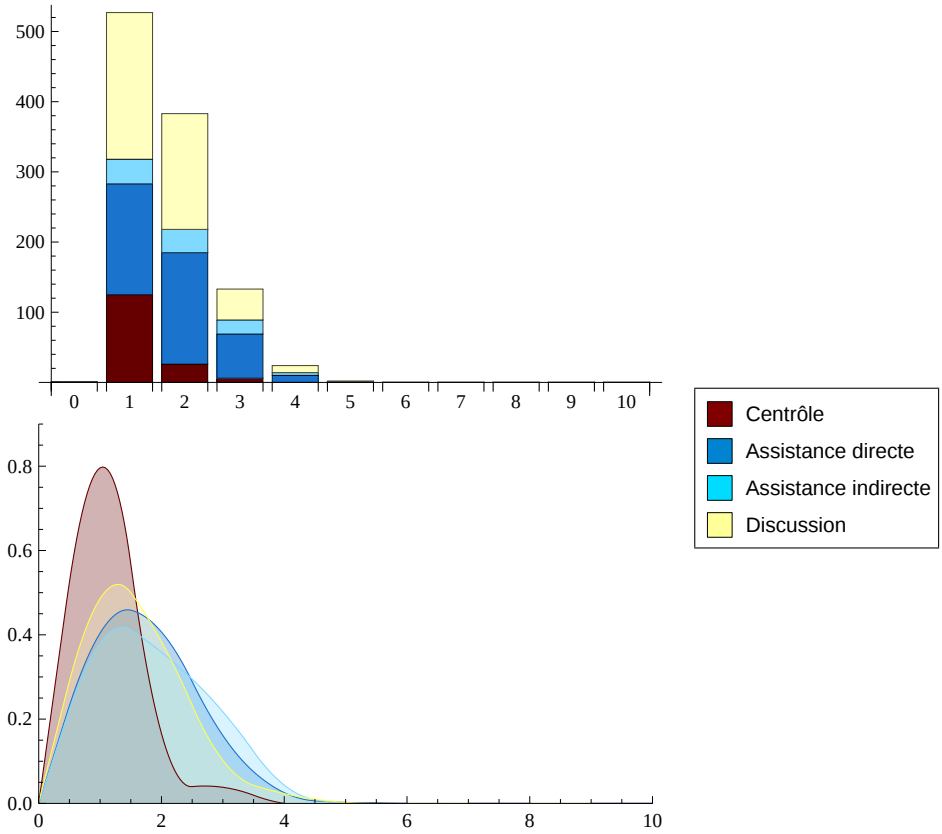


Figure 7.8 Nombre (haut) / Proportion (bas) de requêtes pour chaque activité conversationnelle en fonction de la longueur de la requête formelle (nombre d'entités à la racine)

Analyse de la longueur Bien qu'en théorie la longueur d'une requête formelle devrait toujours être de 1 (comme sur l'exemple de la figure 7.7), ce qui est le cas lorsque celle-ci est complètement analysée, de nombreuses requêtes ont en fait plusieurs éléments au niveau de la racine qui n'ont pas pu être regroupés par l'analyseur sémantique `DIG`. C'est notamment le cas lorsque les requêtes sont formées de deux propositions indépendantes. Ainsi, près de la moitié des requêtes ont une longueur comprise entre 2 et 4.
La figure 7.8 montre le nombre et la proportion d'activités conversationnelles associées à chaque longueur. On remarque par exemple que si seulement la moitié des requêtes sont de longueur 1, cette proportion atteint 80% dans le cas des requêtes de contrôle, et qu'une requête d'une longueur supérieure à 2 a très peu de chances d'être une requête de contrôle. Cette observation est cohérente avec celle déjà faite précédemment en se basant sur la longueur de la requête en langue naturelle, et s'explique par le fait que ces requêtes sont généralement constituées d'un

seul verbe auquel sont associés un ou deux compléments : un objet seul (“Clique le bouton”) ou un sujet et un objet (“Je veux cliquer le bouton”), et ne posent donc pas de problème particulier pour être mises sous forme formelle (*cf.* l'évaluation de la sortie de `DIG` sur la figure 6.9).

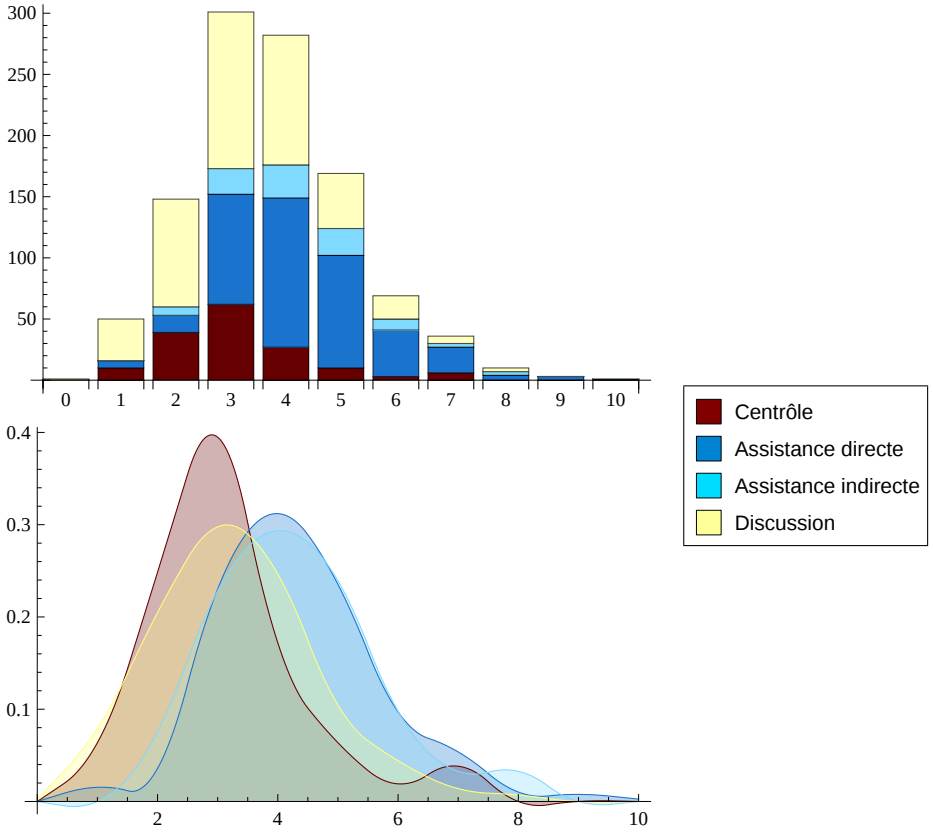


Figure 7.9 Nombre (haut) / Proportion (bas) de requêtes pour chaque activité conversationnelle en fonction de la profondeur de la requête formelle

Analyse de la profondeur Ce paramètre peut constituer un indicateur de la complexité sémantique de la requête (tout du moins si la représentation est correcte). Par exemple, si la phrase “Clique le bouton rouge” a une profondeur de 3 (*cf.* la figure 7.7), la question “Sais-tu si je peux cliquer le bouton rouge ?” implique d'inclure la même requête au sein de deux autres entités, ce qui peut s'écrire sous la forme :

```
KNOWLEDGE[
 person : of = person[id = 'system']
 concept : about = POSSIBILITY[
     todo = ((requête de la figure 7.7))
 ]
]
```

La profondeur est alors de 5 (3 pour la requête de la figure 7.7 et 2 pour l'utilisation des modalités `KNOWLEDGE` et `POSSIBILITY`. Ce changement entraînerait aussi un changement en

termes d'activité conversationnelle de la requête, puisque la phrase d'origine était une requête de contrôle tandis que la phrase ainsi modalisée est une requête d'assistance directe.

Sans surprise, et en accord avec l'exemple ci-dessus, la figure 7.9 révèle que les requêtes de contrôle tendent à être moins profondes (approximativement une loi normale avec un maximum à 3) que les requêtes d'assistance (maximum à 4) qui requièrent souvent des modalités additionnelles autour du contenu propositionnel des phrases. Les requêtes d'assistance directe et d'assistance indirecte ne peuvent toutefois pas être distinguées l'une de l'autre (test de χ^2). Finalement, en ce qui concerne les requêtes de discussion, leur profondeur varie beaucoup et ce paramètre n'est donc pas vraiment pertinent pour permettre de les discriminer des requêtes de contrôle ou d'assistance.

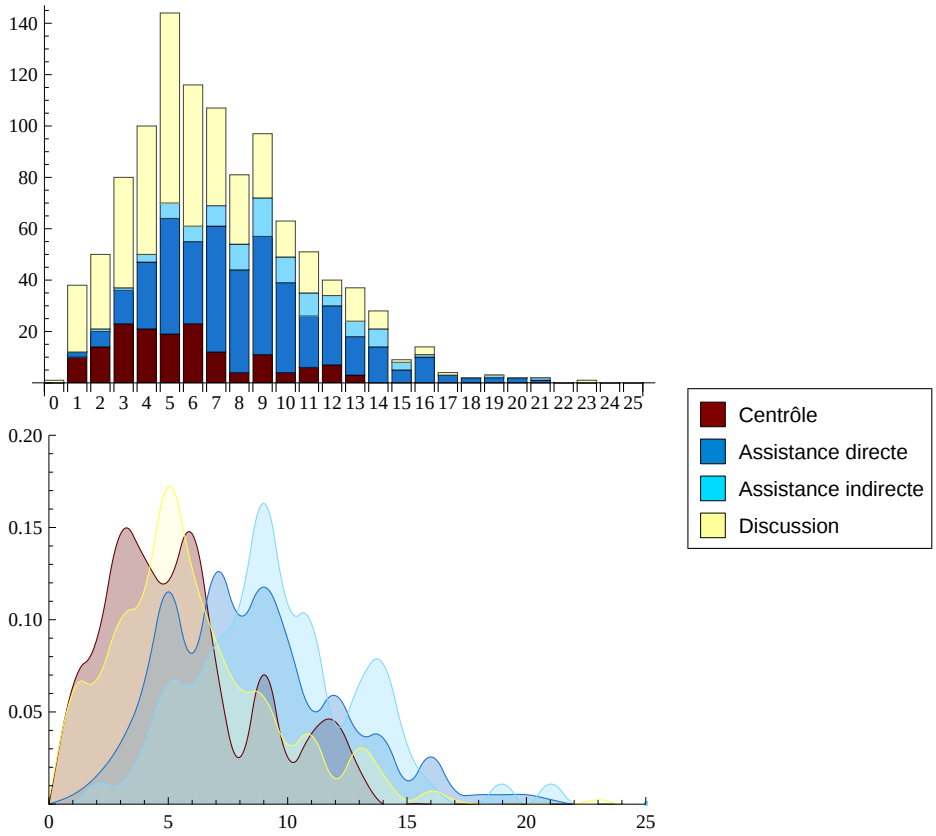


Figure 7.10 Nombre (haut) / Proportion (bas) de requêtes pour chaque activité conversationnelle en fonction du nombre d'entités dans la requête formelle

Analyse du nombre d'entités Lors du passage de la représentation en langue naturelle à la représentation sous forme formelle, trois cas sont possibles :

1. à un mot correspond une entité : c'est le cas le plus fréquent ;
2. à un mot correspond plusieurs entités : *e.g.* "calmer" entraîne l'instanciation de trois entités, étant modélisé par `Modify[manner = manner[val = '+'], property = patient[]]` ;
3. à plusieurs mots correspond une seule entité : c'est le cas de certaines expressions idiomatiques comme "s'il te plaît" qui se traduit simplement par `ASK[]`.

Les cas 2 et 3 s'équilibrant à peu près globalement, les résultats apportés par cette approche, présentés sur la figure 7.10, sont proches de ceux obtenus lorsqu'on s'intéressait à la longueur de la requête en langue naturelle. Encore une fois, on constate donc que les requêtes de contrôle et de discussion tendent simplement à être plutôt plus courtes que les requêtes d'assistance.

7.3.2.3 Combinaison de paramètres basiques de requêtes formelles

Pris séparément, les trois paramètres testés dans la section 7.3.2.2 ne sont pas suffisants pour pouvoir être utilisés pour une identification satisfaisante de l'activité conversationelle de la requête. Nous avons donc fait appel à différentes méthodes de classification classiques ayant recours à ces trois paramètres avec un processus de **validation croisée 10-fold**[4].

Les résultats de cette étude sont présentés dans la table 7.7. Il en ressort que le classifieur le plus efficace est la méthode des K Plus Proches Voisins (KPPV) (avec K=9), mais la performance demeure assez médiocre avec seulement 51,7% de requêtes correctement classifiées (avec un niveau de référence (baseline) situé à 40,2% pour un classifieur naïf identifiant toute requête comme appartenant à la catégorie "discussion").
Si l'on détaille les résultats obtenus avec l'approche des KPPV en fonction des activités conversationnelles (*cf.* table 7.8), il apparaît que ce sont clairement les requêtes d'assistance indirecte qui ne peuvent être clairement discriminées des autres. Au contraire, la performance pour les autres activités conversationnelles atteint un niveau correct de l'ordre de 73%.

Méthode de classification	Résultat
Niveau de référence	*40,2%*
KPPV (meilleure performance avec K=9)	**51,9%**
Perceptron multicouches	49,4%
Réseau bayésien	49,5%
Classifieur bayesien naïf	50,2%
Arbre de décision (C4.5)	51,2%

Tableau 7.7 Performance de différentes méthodes de classification combinant les paramètres basiques (10-folds)

Activité conversationnelle (classe)	Précision	Rappel	F-Mesure
Contrôle	0,701	0,737	0,719
Assistance directe	0,764	0,764	0,764
Assistance indirecte	0,470	0,330	0,388
Discussion	0,720	0,753	0,736

Tableau 7.8 Résultats détaillés de la classification KPPV (K=9) combinant les paramètres basiques (10-folds)

[4]Nous avons pour ce faire eu recours au logiciel Weka dans sa version 3.4.

La combinaison de paramètres simples fournit donc une méthode exploitable de classification de requêtes, mais celle-ci n'est que moyennement performante, et particulièrement mauvaise sur les requêtes d'assistance indirecte. Pour une application pratique vraiment utile, il serait donc souhaitable de trouver une autre méthode qui améliore les performances, au moins sur ces requêtes d'assistance indirecte.

7.3.2.4 Utilisation des profils interactionnels

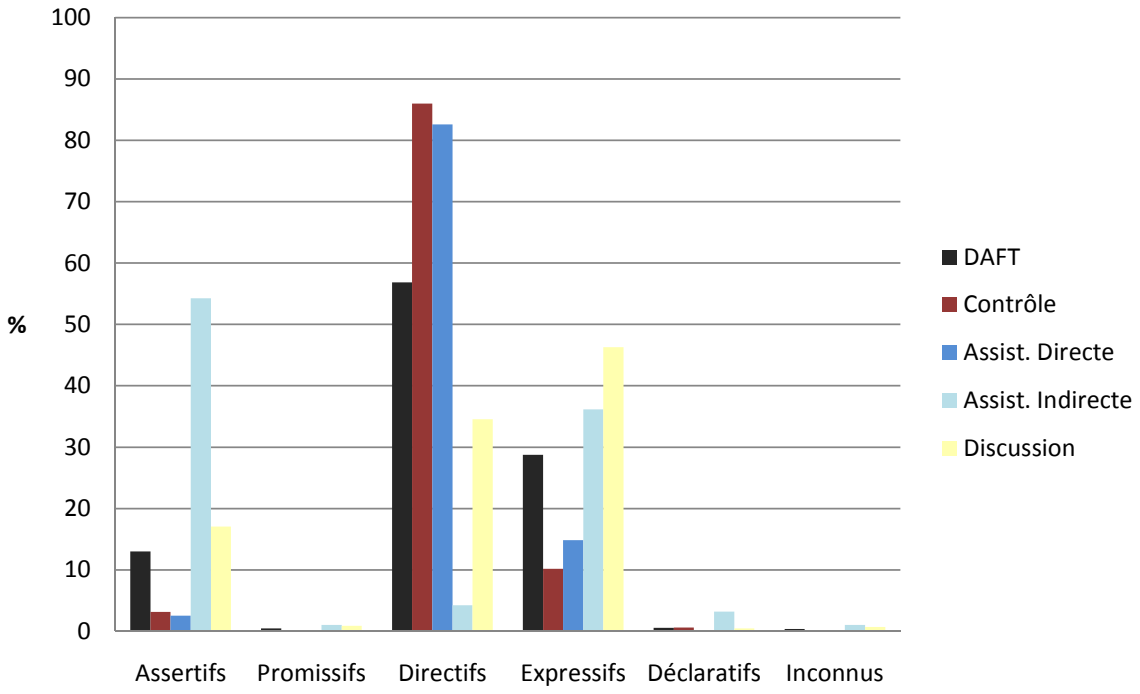


Figure 7.11 Comparaison des profils interactionnels des quatre activités conversationnelles du corpus *Daft*

Alors que les méthodes précédentes faisaient appel à des paramètres purement quantitatifs, il est également possible d'envisager une classification fondée sur un critère qualitatif lié à la pragmatique des requêtes, en utilisant l'annotation en termes d'actes de dialogue effectuée pour la caractérisation du corpus *Daft* (*cf.* section 7.1.2.1). Il est en effet possible de faire de nouveau appel aux profils interactionnels, tels qu'ils ont été définis dans cette même section. L'unité d'interaction choisie ici n'est en revanche plus le corpus dans son intégralité, mais les sous-corpus constitués par les ensembles de phrases correspondant à chaque activité conversationnelle.

Le résultat de cette comparaison, représenté sur la figure 7.11, confirme clairement l'hétérogénéité du corpus *Daft* (dont le profil interactionnel moyen, tel qu'il avait été établi dans la section 7.1, est rappelé en gris foncé). En particulier, il apparaît une différence très nette entre les requêtes d'assistance directe (composées essentiellement de directifs et de quelques expressifs) des requêtes d'assistance indirecte (formées d'assertifs et d'un peu d'expressifs). Au contraire, les profils interactionnels des activités de contrôle et d'assistance directe sont assez

similaires.
On note donc que cette méthode pourrait être assez complémentaire de celle vue dans la section 7.3.2.3 qui au contraire avait du mal à discriminer les requêtes d'assistance indirectes. Toutefois, bien qu'intéressant d'un point de vue théorique, ce résultat n'est pas réellement exploitable en pratique dans un système de classification automatique de nouvelles requêtes, dans le mesure où la détection d'actes de dialogues n'est elle-même pas triviale à gérer de manière automatique.

7.3.3 Classification en fonction de la structure : la requête comme un arbre

Bien que dans la plupart des captures d'écran illustrant des requêtes *DAFT* (comme la figure 7.7) nous avons opté pour une représentation compacte sous forme de boîtes imbriquées, il est tout à fait possible de considérer ces mêmes requêtes comme des arbres. À titre d'exemple, l'équivalent sous forme d'arbre de la requête de la figure 7.7 est représentée sur la figure 7.12.
L'intérêt de ce changement représentationnel est ici de pouvoir considérer l'application d'algorithmes d'apprentissage automatique (machine learning) s'appliquant à ce type de données structurées, notamment les approches basées sur l'utilisation de fonctions noyaux. Nous allons donc dans cette partie rappeler les principes d'utilisation de ces méthodes, étudier la possibilité de les appliquer dans le cas présent (moyennant certaines modifications) et présenter les résultats numériques ainsi obtenus.

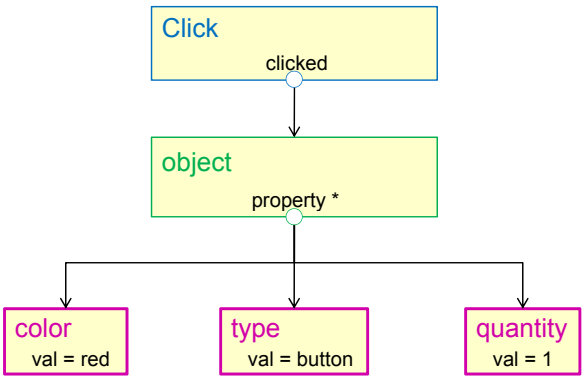


Figure 7.12 Représentation formelle *DAFT* de la requête R_{ex_1} "Clique le bouton rouge" mise sous forme d'arbre

7.3.3.1 Méthodologie : méthodes à base de fonction noyau pour les arbres

Principes généraux des méthodes à base de fonction noyau L'idée sous-jacente aux méthodes de classification fondées sur l'utilisation d'une fonction noyau consiste à effectuer un changement de dimensions permettant de rendre un problème de classification non-linéaire résolvable avec des méthodes de classification linéaire classiques. Mathématiquement, elle est fondée sur le théorème de Mercer qui démontre que toute fonction de noyau étant à la fois

continue, symétrique et semi-définie positive peut être exprimée sous la forme d'un produit scalaire dans un espace de dimension élevée [Mercer, 1909]. La conséquence de ce résultat est qu'à partir du moment où l'on dispose d'une définition de fonction de noyau, tout algorithme reposant uniquement sur le produit scalaire de deux vecteurs peut être transformé en une version non-linéaire en y remplaçant l'usage du produit scalaire par cette fonction noyau.

Application des fonctions noyaux aux arbres Dans le cas qui nous intéresse ici, nous avons besoin d'une fonction noyau basée sur la distance entre deux éléments R_1 et R_2 (*i.e.* deux listes imbriquées ou deux arbres correspondant à l'analyse sémantique formelle de deux requêtes d'utilisateurs). Pour cela, il nous faut introduire un certain nombre de notations que l'on va illustrer à partir de l'exemple de la figure 7.12, nommé ici R_{ex_1}. Ainsi, pour toute requête R_i, on définit :

$s(R_i)$	l'identifiant de R_i	(*e.g.* $s(R_{ex_1})$ = `Click`)
$f_{i,j}$	le j^{e} champ de R_i	(*e.g.* $f_{ex,2}$ = `clicked`)
$\|f_i\|$	le nombre de champs de R_i	(*e.g.* $\|f_{ex}\| = 3$)
$t(f_{i,j})$	le type de $f_{i,j}$	(*e.g.* $t(f_{ex,2})$ = `object`)
$v(f_{i,j})$	la valeur de $f_{i,j}$:	
	– une requête	(*e.g.* `clicked = object[...]`)
	– une liste de requêtes	(*e.g.* champ `property` d'`object`)
	– une valeur terminale ($\in$ liste ou un intervalle)	(*e.g.* `val = button`)
$\|v(f_{i,j})\|$	le nombre de valeurs de $f_{i,j}$	
	(1, sauf pour les champs à valeurs multiples comme ppt* pour les références)	
$\mathcal{R}_i$	l'ensemble de tous les sous-arbres extractible à partir de la requête R_i	
	(*e.g.* $\mathcal{R}_{ex}$ = `Click[...]`, `object[...]`, `type[...]`, `quantity[...]`, `color[...]`)	

On souhaite mesurer la distance entre les arbres représentant R_1 et R_2, ce qui peut se faire à l'aide d'une fonction noyau basée sur celle décrite par Collins & Duffy [2002] (et redétaillée dans Gaertner [2003]), permettant de mesurer la distance entre deux arbres orientés (les arcs partent de la racine et vont vers les feuilles) ordonnés (les noeuds fils sont dans un ordre constant) et étiquetés (une valeur unique est associée à chaque nœud de l'arbre). Cette fonction noyau repose sur l'utilisation de tous les sous-arbres de R_1 et R_2, c'est-à-dire avec la notation introduite ci-dessus, $\mathcal{R}_1$ et $\mathcal{R}_2$.
La fonction noyau k est donc mathématiquement définie par la formule :

$$k(R_1, R_2) = \sum_i h_i(R_1) h_i(R_2)$$

où $h_i(R)$ est le nombre de fois où le i^{e} sous-arbre apparaît parmi tous les sous-arbres possibles de la requête R. Ainsi, dans R_{ex_1}, le sous-arbre ayant pour racine le nœud `object` apparaît

trois fois : dans R_{ex_1} elle-même, dans le sous-arbre ayant pour racine `Click` et dans celui ayant pour racine `object` lui-même.

Le problème de cette définition est que le nombre de sous-arbres pour une requête un peu complexe peut être très important (puisqu'il augmente de manière exponentielle avec la taille de l'arbre), et que la complexité de l'algorithme en dépend. Pour revenir à une complexité polynomiale, si on considère deux sous-arbres r_1 et r_2 appartenant respectivement à $\mathcal{R}_1$ et $\mathcal{R}_2$, il est possible de réécrire la fonction noyau sous la forme :

$$k(R_1, R_2) = \sum_{r_1 \in \mathcal{R}_1, r_2 \in \mathcal{R}_2} S(r_1, r_2)$$

où $S(r_1, r_2)$ est le nombre de sous-arbres isomorphiques de r_1 et r_2.

Si l'on définit alors :

$\lvert\delta(r)\rvert$	le nombre de nœuds fils de l'entité r $\neq \lvert f_i\rvert$, les feuilles de l'arbre pouvant avoir des champs avec une valeur terminale (*e.g.* `type[val="button"]` a une valeur terminale mais pas de requête fille)
$\delta(r, j)$	le j$^{\text{e}}$ nœud fils de l'entité r

On peut calculer la valeur de $S(r_1, r_2)$ de la manière suivante :

$$\begin{aligned} S(r_1, r_2) &= 0 \text{ si } s(r_1) \neq s(r_2) \\ S(r_1, r_2) &= 1 \text{ si } s(r_1) = s(r_2) \wedge |\delta(r_1)| = |\delta(r_2)| = 0 \\ S(r_1, r_2) &= \prod_{j=1}^{|\delta(r_1)|} (1 + S(\delta(r_1, j), \delta(r_2, j))) \text{ sinon} \end{aligned}$$

Dans la mesure où $|\delta(r_1)| = |\delta(r_2)|$, car le nombre de nœuds fils d'une entité est fixé par son identifiant (*e.g.* toute requête `Click[...]` aura forcément trois champs pouvant être associés à trois nœuds fils, même si certains sont laissés vides, comme c'est le cas pour deux d'entre eux non représentés dans R_{ex_1}). Le calcul de k à partir de cette nouvelle définition récursive a une complexité temporelle nettement plus faible que précédemment, en $\mathcal{O}(|R_1||R_2|)$.

7.3.3.2 Méthodologie : application aux requêtes *DAFT*

Applicabilité de la fonction noyau pour les arbres aux requêtes *DAFT* Pour confirmer qu'il nous est possible d'appliquer cette méthode, nous vérifions que les contraintes précédemment énoncées sont satisfaites par l'analyse sémantique dont nous disposons :

– Ordre des nœuds fils : tout élément de requête est construit à partir du schéma *DAFT* associé, l'ordre des champs dans la requête est fixé uniquement par l'identifiant $s(R_i)$ et est donc toujours le même pour un identifiant donné. Les nœuds fils d'une entité sont par conséquent bien dans un ordre constant.

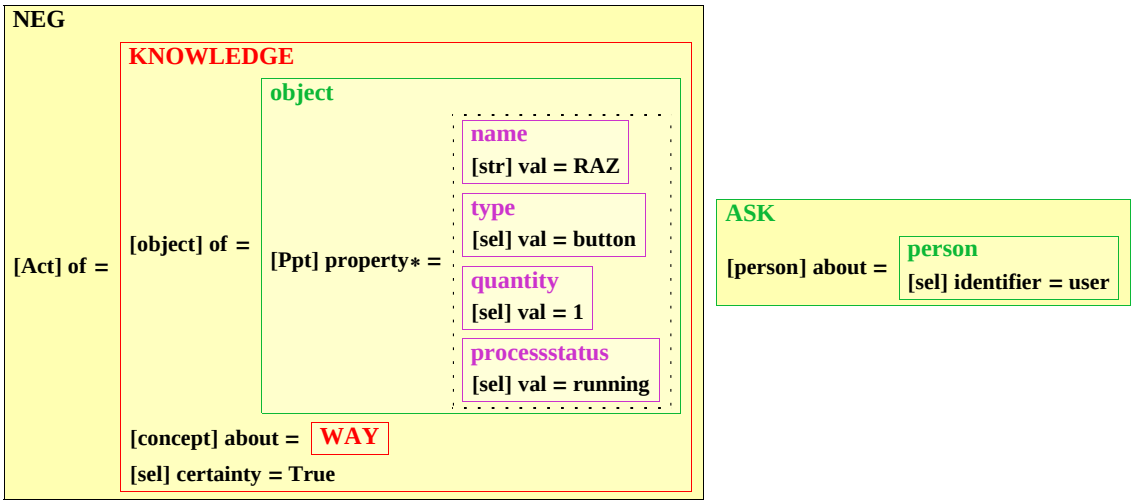


Figure 7.13 Représentation formelle *DAFT* de la requête R_{ex_2} "Je ne comprends pas comment fonctionne le bouton RAZ"

– Lien identifiant – nœuds fils : le nombre de champs est le même pour un identifiant donné (*e.g.* `Cliquer`), on a donc bien $f_i = f_j$ si $s(R_i) = s(R_j)$.

– Requêtes partielles à racines multiples : un problème semble se poser avec des requêtes dont l'analyse sémantique n'a été que partielle, comme c'est le cas avec la requête représentée (dans le mode de répresentation "boîtes imbriquées") sur la figure 7.13. En effet, ces requêtes n'ont pas une entité donnée constituant une racine unique. Toutefois, d'un point de vue formel, on peut considérer que les N arbres indépendants ainsi formés (N = 2 dans le cas de R_{ex_2}) peuvent être vus comme un arbre unique doté d'un nœud racine sans identifiant mais doté de N nœuds fils, de la même manière qu'on considère les 3 nœuds fils de `property*` dans R_{ex_1} comme étant au même niveau. Les éléments doivent être ordonnés : on considère qu'étant lié à l'ordre des mots dans la requête d'origine en langue naturelle, cet ordre n'a pas besoin d'être changé.

– Champs multiples : comme rappelé dans le point ci-dessus, un champ peut parfois contenir une liste de valeurs (et non une seule – *cf.* `property*`), ce qui signifie qu'un champ donné peut être associé à plus d'un nœud fils. D'un point de vue purement représentationnel, il n'y a pas de problème, puisqu'on peut associer trois nœuds fils à `property*`, tout comme `KNOWLEDGE` sur la figure 7.13 possède trois nœuds fils (un par champ).
Le problème du champ multiple toutefois est qu'il introduit une variabilité d'une part dans le nombre de fils associés à un champ, d'autre part dans l'ordre même des nœuds fils (lié à l'ordre d'apparition des éléments dans la phrase). Par exemple, les requêtes formelles *DAFT* correspondant aux phrases "Clique le petit bouton rouge" et "Clique le bouton rouge et petit" seront différentes (dans l'ordre des propriétés).

Les conditions nécessaires pour appliquer la fonction noyau aux requêtes dont nous disposons sont remplies, mais le dernier point concernant les champs multiples montre qu'une adaptation de la formule de calcul de S est nécessaire de manière à prendre en compte cette particularité.

Modifications nécessaires à la fonction noyau La fonction noyau, telle qu'introduite précédemment, n'est pas entièrement satisfaisante pour plusieurs raisons liées à la fois à la spécificité des requêtes *DAFT* (problèmes 1 et 2) et à des propriétés intrinsèques à la définition choisie (problèmes 3 et 4) :

1. La fonction noyau ne prend en compte que les identifiants des requêtes (par exemple `Click`) mais pas les valeurs des champs lorsque ceux-ci contiennent une valeur terminale (au sens donné dans la section 5.2.1.4, c'est-à-dire qui n'est pas une entité). Cela signifie par exemple que "Clique le bouton rouge" et "Clique le bouton bleu" seront deux requêtes considérées comme étant identiques car la valeur terminale `"red"` ou `"blue"` associée à l'entité `color[]` ne sera pas prise en compte (ce qui n'est pas grave dans cet exemple, mais demeure un problème d'un point de vue théorique).
2. La fonction noyau ne permet pas de gérer les champs à valeur multiples (*cf.* le dernier point du paragraphe précédent).
3. La valeur retournée par la fonction noyau est dépendante de la taille de l'arbre. Par exemple, la valeur retournée pour deux requêtes identiques avec peu d'éléments sera plus faible que celle retournée pour deux requêtes identiques dotées de plus d'éléments.
4. Le noyau subit un "effet de pic" (peaked), c'est-à-dire qu'une très légère modification au niveau d'un sous-arbre peut avoir un impact très important sur la valeur de la fonction de distance. Par exemple, on peut facilement avoir $k(R_1, R_1) \geq 10^2 \times k(R_1, R_2)$, ce qui signifie qu'un poids très important va être accordé à la requête la plus proche par rapport aux autres, ce qui peut poser problème lors de l'utilisation de certains algorithmes de classification.

La résolution des problèmes 1 et 2 liés aux valeurs spécifiques qui peuvent être associées aux champs (une valeur terminale dans le premier cas, une liste dans le second) passe par des modifications dans le calcul de S :

1. La prise en compte des valeurs terminales dans les sous-arbres entraîne les changements suivants :

$$
\begin{aligned}
S(r_1, r_2) = &\quad 0 \text{ si } s(r_1) \neq s(r_2) \\
S(r_1, r_2) = &\quad 1 \text{ si } s(r_1) = s(r_2) \wedge |f_1| = |f_2| = 0 \\
S(r_1, r_2) = &\quad \frac{1 + |v(f_1, j) = v(f_2, j), j \in [\![1, |f_1|]\!]|}{1 + |f_1|} \\
&\quad \text{si } s(r_1) = s(r_2) \wedge |\delta(r_1)| = |\delta(r_2)| = 0 \\
S(r_1, r_2) = &\quad \prod_{j=1}^{|\delta(r_1)|} (1 + S(\delta(r_1, j), \delta(r_2, j))) \text{ sinon}
\end{aligned}
$$

La deuxième ligne permet de gérer le cas où r_1 et r_2 ont des valeurs terminales identiques (*e.g.* `"red"` et `"red"`), tandis que la troisième ligne s'applique lorsque les identifiants de r_1 et de r_2 sont identiques mais pas nécessairement le contenu de leurs champs (*e.g.* `a[b,c]` et `a[b,d]` ont le même identifiant (`a`) mais seulement un champ en commun). Au lieu de considérer que deux

requêtes peuvent simplement être identiques (1) ou différentes (0), on pondère la valeur de leur proximité en fonction de la similarité des valeurs (terminales ou non) de leurs champs : si tous les champs contiennent la même valeur, ils sont identiques (1), mais si toutes les valeurs sont différentes, la proximité est faible mais non nulle ($1/(1+|f1|)$) dans la mesure où les identifiants au moins sont identiques.

2. Pour prendre en compte les champs à valeurs multiples, le problème est double car cela signifie qu'il faut gérer à la fois :

- La différence d'ordre potentielle entre des contenus de listes pourtant identiques mais générées dans un ordre différent d'après l'ordre des mots de la phrase. Dans le cas de l'exemple mentionné précédemment, on a :
  a) `object[ppt* = {color[val = "red"], size[val = "small"]}]`
  b) `object[ppt* = {size[val = "small"], color[val = "red"]}]`
  Pour traiter ce cas, on peut se ramener à un cas ordonné (comme suggéré par Vishwanathan & Smola [2004]), simplement en utilisant les identifiants de requêtes pour un classement par ordre alphabétique du contenu des champs multiples. Il est alors effectué lors d'un prétraitement des données, ce qui, dans l'exemple précédent, ramènerait toujours au cas a).
- La variabilité du nombre de champs, ce qui peut rendre impossible dans certains cas d'effectuer une simple comparaison deux à deux du contenu des champs. Cependant, l'algorithme peut être adapté de manière à tester plutôt la présence de valeurs issues d'un champ multiple de r_1 dans un champ multiple de r_2 (et inversement). On normalise enfin le résultat par le nombre total de valeurs contenues dans le champ multiple de r_1 et de r_2. D'un point de vue formel, cela donne :

$$S(r_1, r_2) = \frac{|\{v(f_{1,j}) \in f_{2,j}\}| + |\{v(f_{2,j}) \in f_{1,j}\}|}{|\{v(f_{1,j})\} \cup \{v(f_{2,j})\}|} \quad \text{si } s(r_1) = s(r_2) \wedge |v(f_{i,j})| \neq 1$$

En ce qui concerne les problèmes 3 et 4, une solution a été suggérée dans Collins & Duffy [2002] et est également applicable dans notre contexte :

3. Pour éviter l'augmentation de la valeur de k lorsque la complexité de la requête (nombre d'éléments) augmente, il est possible de procéder à une normalisation en définissant une nouvelle fonction noyau k' telle que :

$$k'(R_1, R_2) = \frac{k(R_1, R_2)}{\sqrt{k(R_1, R_1) \times k(R_2, R_2)}}$$

4. Finalement, pour éviter l'effet de pic, une solution consiste à ajouter un coefficient λ tel que $0 < \lambda \leq 1$ dans le calcul de S, ce qui permet de rendre le poids dépendant du nombre

de sous-arbres présents dans la représentation de la requête. Formellement, on modifie donc la formule de calcul de k de la manière suivante :

$$k(R_1, R_2) = \lambda^{\text{size}_i} \sum_i h_i(R_1) h_i(R_2)$$

La valeur de λ peut être changée pour optimiser la classification.

7.3.3.3 Résultats de la classification

En ayant ainsi défini une mesure de distance appropriée entre les requêtes, il devient possible de faire appel à diverses méthodes classiques de classification de données, en évaluant la distance entre d'une part la requête dont on souhaite identifier l'activité conversationnelle, et d'autre part l'ensemble des requêtes constituant l'ensemble d'apprentissage. Pour évaluer la performance du classifieur, nous avons eu recours à une validation croisée 10-fold, et les résultats donnés pour chaque classifieur correspondent à la moyenne des 10 résultats ainsi obtenus.

Nous avons fait appel à la méthode des KPPV, en faisant varier le nombre de voisins K de 1 à 15, et le paramètre λ (introduit pour éviter l'effet de pic) de 0,05 à 1, par pas de 0,05. Dans tous les cas, les performances optimales ont été obtenues pour $\lambda \in [0.7, 0.9]$ (*cf.* figure 7.14), et la performance du classifieur décroit lorsque K augmente (*cf.* figure 7.15 testant K $\in [2, 15]$), le meilleur résultat étant donc obtenu pour K= 1 (*i.e.* le cas du Plus Proche Voisin (PPV)) et $\lambda = 0.85$, qui permet d'obtenir correctement l'activité conversationnelle d'en moyenne 65,4% des requêtes.
Quand plus d'une seule classe est possible (*e.g.* dans le cas de 4 plus proches voisins, si deux cas sont de type contrôle et deux de type discussion), la classe est choisie aléatoirement parmi les possibilités disponibles. Changer cette stratégie de manière à prendre plutôt la catégorie du plus proche voisin n'apporte pas de différence significative dans les résultats (si ce n'est l'apparition logique d'un plateau au début de la courbe, puisque dans ce cas, K= 2 est identique à K= 1 – *cf.* figure 7.15)

7.3.4 Classification en fonction de la sémantique : la requête comme un vecteur

7.3.4.1 Méthodologie : définition des vecteurs de schèmes

Pour chacune des requêtes, il est possible de considérer une représentation simplifiée ne prenant pas en compte la structure des éléments mais simplement le nombre de schèmes de chaque catégorie parmi les 237 schèmes existants. Nous appellerons les vecteurs ainsi définis les vecteurs de schèmes. Les vecteurs de schèmes ainsi définis sont essentiellement creux, puisqu'une requête contient rarement plus d'une dizaine de schèmes uniques. Un extrait de

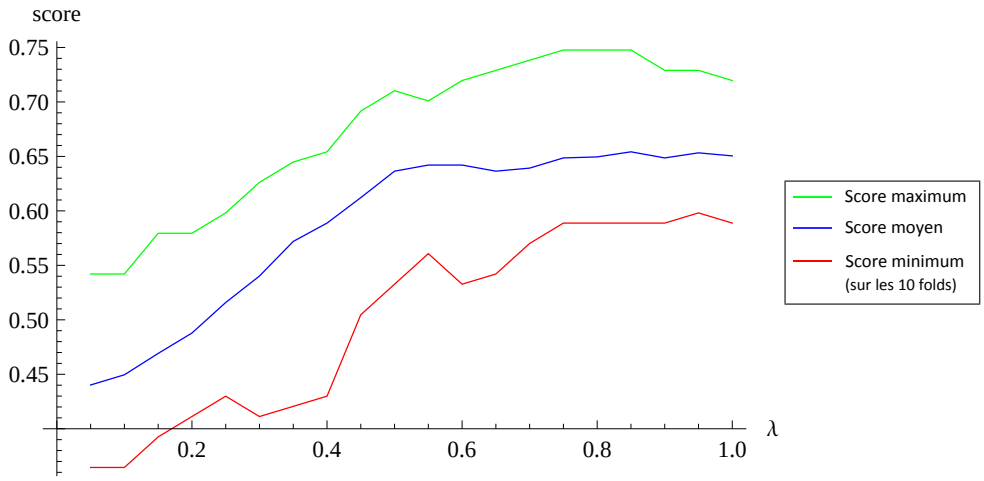
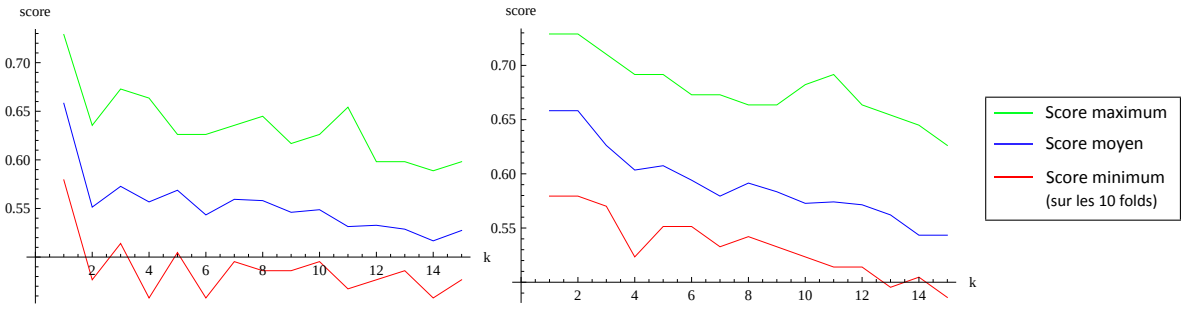


Figure 7.14 Résultat de la classification avec la fonction noyau et la méthode du PPV en fonction de λ – maximum observé pour K = 1 et $\lambda \in [0.7, 0.9]$



(a) Choix aléatoire (b) Choix du plus proche voisin

Figure 7.15 Résultat de la classification avec la fonction noyau et la méthode du KPPV en fonction de K pour $\lambda = 0.75$, avec deux choix en cas d'égalité

la représentation des vecteurs correspondant aux requêtes d'exemples déjà introduites figure dans le tableau 7.9.

Requête	ASK	NEG	WAY	...	Click	Modify	...	object	person	concept	...	type	name	quantity	color	...
R_{ex_1}	0	0	0	...	1	0	...	1	0	0	...	1	0	1	1	...
R_{ex_2}	1	1	1	...	0	0	...	1	1	0	...	1	1	1	0	...

Tableau 7.9 Représentation des requêtes d'exemples des figures 7.7 et 7.13 sous forme de vecteurs de schèmes

En transcrivant ainsi les 1 070 requêtes du corpus $Daft_{sub}$, on obtient une matrice de 1 070×237, qui peut être utilisée par n'importe quel classifieur classique pour identifier l'activité conversationnelle.

7.3.4.2 Résultats de la classification

Comme on l'a déjà fait dans les sous-sections précédentes, nous pouvons opter pour une approche des KPPV. Mais cette fois la fonction de distance d entre deux vecteurs de schèmes n'ayant pas été définie au préalable, plusieurs peuvent être essayées. Soit deux vecteurs de schèmes, $R = R_1, R_2, \ldots, R_i, \ldots, R_n$ et $S = S_1, S_2, \ldots, S_i, \ldots, S_n$ avec $n = 237$, on peut considérer les distances suivantes :

- la distance de Manhattan (ou dans L^1) : $d(R,S) = \sum_{i=1}^{n} |R_i - S_i|$
- la distance euclidienne (ou dans L^2) : $d(R,S) = \sqrt{\sum_{i=1}^{n} |R_i - S_i|}$
- la distance de Tchebychev (ou dans L^∞) : $d(R,S) = \lim_{p\to\infty} \sqrt[p]{\sum_{i=1}^{n} |R_i - S_i|} = \sup_i |R_i - S_i|$
- la distance de Bray-Curtis (ou de Sorensen) : $d(R,S) = \frac{\sum_{i=1}^{n} |R_i - S_i|}{\sum_{i=1}^{n} |R_i + S_i|}$
- la distance de Canberra : $d(R,S) = \sum_{i=1}^{n} \frac{|R_i - S_i|}{|R_i| + |S_i|}$
- la distance de cosinus : est basée sur l'angle Θ formé par R et S, tel que $\cos(\Theta) = \frac{R \cdot S}{\|R\| \|S\|}$, qui mesure la similarité entre R et S (comprise entre 0 et 1). La distance est donc donnée par $d(R,S) = 1 - \cos(\Theta)$
- la distance de Hamming : mesure le nombre de substitutions nécessaires pour passer de R à S. C'est un cas particulier de la distance de Levenshtein (ou distance d'édition) dans laquelle on n'autorise pas les opérations d'insertion et de suppression (*e.g.* la distance de Levenshtein entre $(0,1,0,1,1)$ et $(1,0,1,0,1)$ est de 2 (une suppression en fin et une insertion en début) alors que la distance de Hamming sera de 4). Cette dernière distance n'est pas considérée ici dans la mesure où non seulement les vecteurs sont de taille

égale (par hypothèse) et les opérations de substitutions suffisent donc, mais surtout les opérations d'insertion et de suppression n'ont pas grand sens ici.

Comme précédemment, pour obtenir des résultats comparables, nous avons eu recours à une méthode de validation croisée 10-fold et fait varier le nombre de voisins K de 1 à 20. La synthèse des résultats de cette évaluation avec les 7 mesures de distances introduites ci-dessus est présentée sur la figure 7.16. On constate que globalement, les distances dans L^p sont assez peu adaptées (Tchebychev étant particulièrement mauvaise). Hamming est également assez médiocre, tandis que les méthodes basées sur l'utilisation de valeurs absolues (Canberra et Bray-Curtis) et du cosinus pour mesurer la similarité sont un peu meilleures.

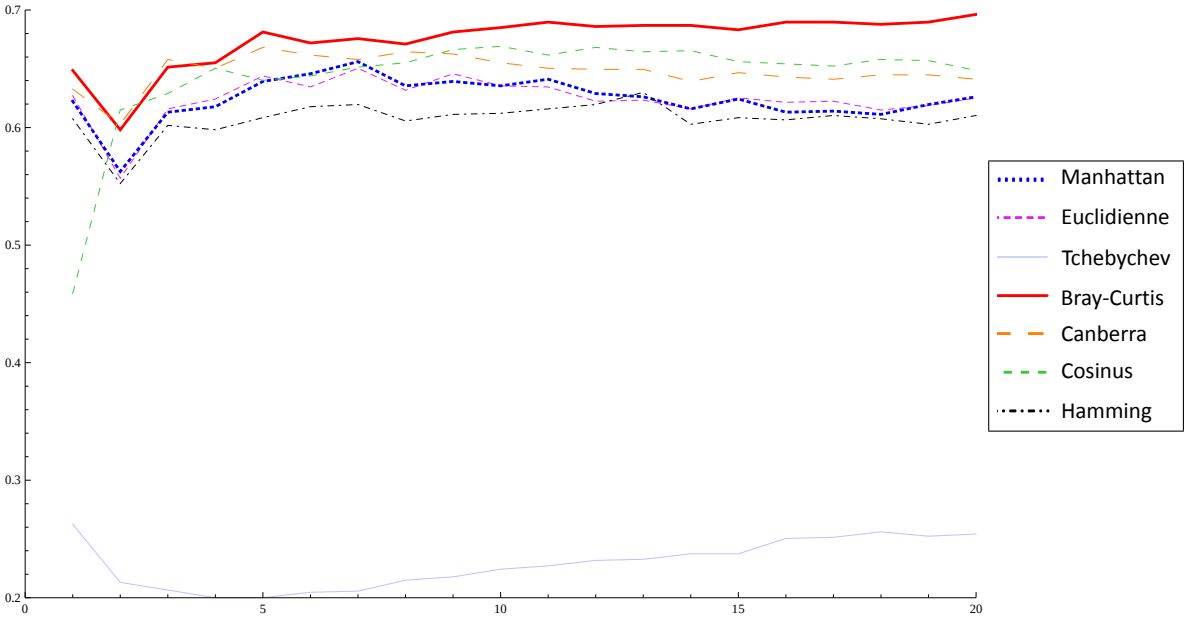
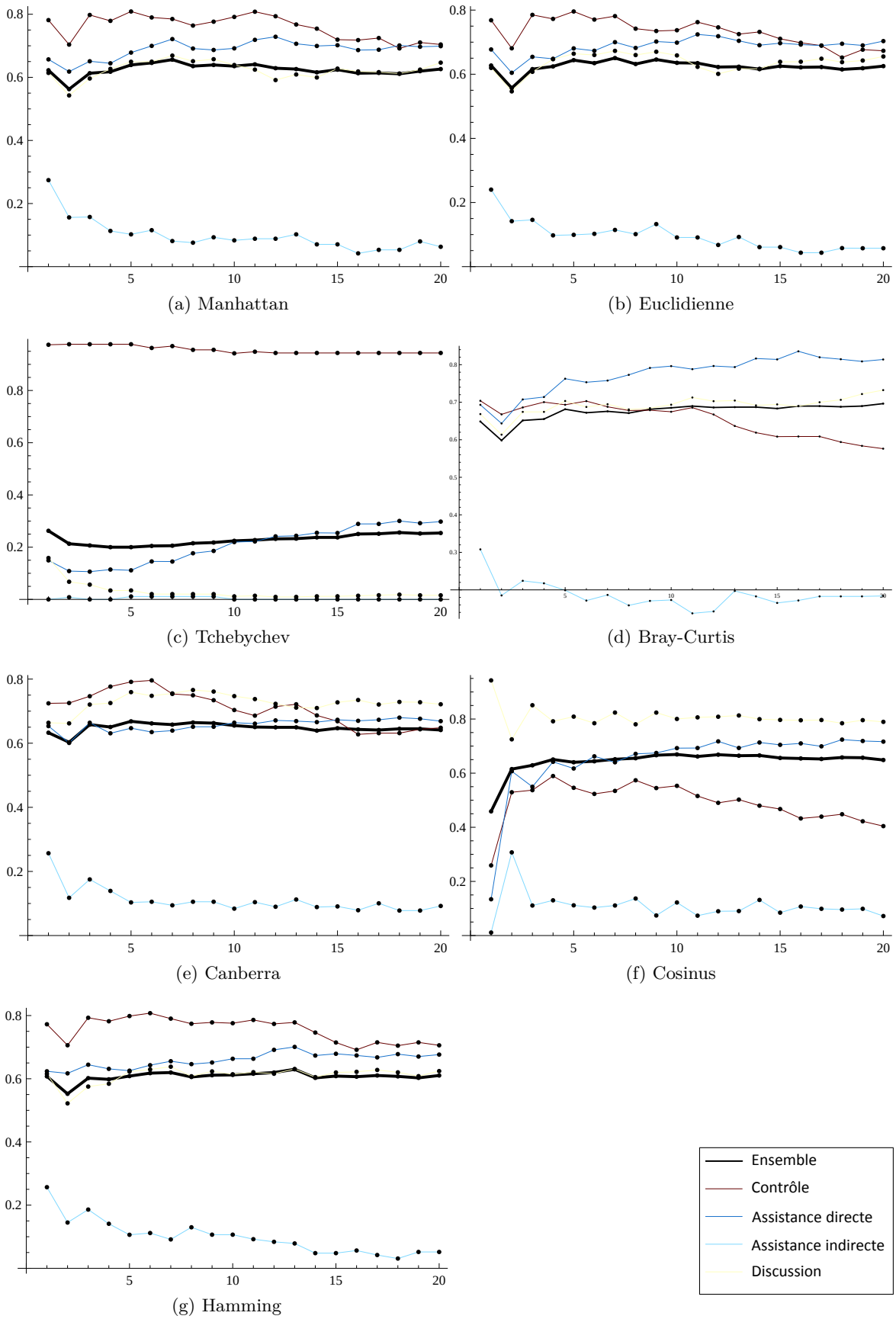


Figure 7.16 Comparaison des performances des différentes distances pour l'approche des KPPV ($K \in [\![1, 20]\!]$) pour les vecteurs de schèmes (10-folds)

Si on détaille par catégorie (*cf.* figure 7.17), on constate qu'une nouvelle fois ce sont les requêtes d'assistance indirecte qui posent problème et font baisser la performance globale du classifieur. Les faibles performances obtenues avec la distance de Tchebychev s'expliquent également (*cf.* figure 7.17c) par sa tendance à classer toute requête comme une requête de contrôle. La distance fournissant les meilleurs résultats ici est celle de Bray-Curtis, avec une performance de 69,2% pour K= 11, ce qui est meilleur que ce qui avait pu être obtenu avec l'approche par arbre dans la section 7.3.3.

D'autres méthodes de classification standards ont été employées à l'aide de l'outil *Weka* et les meilleurs résultats obtenus avec chacune des approches sont synthétisés dans la table 7.10. On constate que les meilleurs résultats ont pu être obtenus à l'aide d'un classifieur bayésien naïf (avec estimateur de noyau).



(a) Manhattan (b) Euclidienne

(c) Tchebychev (d) Bray-Curtis

(e) Canberra (f) Cosinus

(g) Hamming

Figure 7.17 Performance des différentes distances en fonction de l'activité conversationnelle (10-folds)

Méthode de classification	Résultat
Niveau de référence	*40,2%*
Perceptron multicouches	55,7%
KPPV (K=11, Bray-Curtis)	69,2%
Arbre de décision (C4.5)	70,4%
Adaboost avec C4.5	71,8%
Bayésien naïf (avec estimateur de noyau)	**74,1%**

Tableau 7.10 Performance de différentes méthodes de classification basées sur les vecteurs de schèmes (10-folds)

7.3.5 Combinaison de classifieurs

7.3.5.1 Principes

Dans la mesure où nous avons défini de nombreux classifieurs différents, il nous a paru intéressant de chercher à combiner les deux approches précédentes (modélisation des requêtes sous forme d'arbres et de vecteurs) dans un système à classifieurs multiples, à la manière de Kittler [1998] qui a montré que l'association de classifieurs faibles individuellement pouvait permettre de réaliser un classifieur plus fort en les associant. On distingue deux façons d'associer des classifieurs :

- une combinaison non paramétrique (sans apprentissage) : il s'agit en fait d'un système de vote, où la sortie de chaque classifieur est traitée de manière égale, sans prendre en compte les spécificités de celui-ci.
- une approche paramétrique (avec apprentissage) : dans laquelle on ajoute un nouveau classifieur qui prend en entrée les sorties des classifieurs de premier niveau (*i.e.* travaillant directement sur les données du problème).

7.3.5.2 Applicabilité

Pour chaque requête, les classifieurs précédents ont fournit une classe prédite C dans l'ensemble {Contrôle, Assistance directe, Assistance indirecte, Discussion}. Mais la plupart de ces mêmes classifieurs peuvent aussi fournir ce résultat sous la forme d'une probabilité pour une requête d'appartenir à une classe donnée, associant ainsi à chaque requête un quadruplet $\{p(Ctrl), p(AssD), p(AssI), p(Disc)\}$ (tel que $p(Ctrl) + p(AssD) + p(AssI) + p(Disc) = 1$). Nous allons donc traiter successivement dans les deux sous-sections suivantes ces deux cas : les classes, où les classifieurs seront associés de manière paramétrique, et les probabilités où nous les associerons de manière non paramétrique.

Les résultats obtenus avec les classifieurs précédents pris individuellement sont synthétisés dans la table 7.11. On leur associe également un identifiant, de manière à pouvoir exprimer facilement les combinaisons de classifieurs : "bas" correspond aux classifieurs appliqués aux paramètres basiques de la section 7.3.2.2, "arb" au classifieur utilisant une fonction noyau

développée pour mesurer la distance entre les arbres de requêtes dans la section 7.3.3 et "vec" aux classifieurs de la section 7.3.4 travaillant sur la représentation des requêtes sous forme de vecteurs.

À noter que les légères différences dans les valeurs données ici pour la méthode des KPPV avec la distance de Bray-Curtis proviennent du fait que les classifieurs sont ici évalués en **validation croisée leave-one-out** et non en validation croisée 10-fold comme précédemment (l'ensemble d'apprentissage est donc un peu plus grand, d'où des performances légèrement accrues).

Identifiant	Méthode de classification	Résultat
bas1	KPPV (K=9)	52,4%
bas2	Perceptron multicouches	50,3%
bas3	Réseau bayésien	51,1%
bas4	Classifieur bayesien naïf	49,3%
bas5	Arbre de décision (C4.5)	52,2%
arb	PPV pour $\lambda = 0.85$	66,2%
vec1	Bayésien naïf (avec estimateur de noyau)	74,4%
vec2	Arbre de decision (C4.5)	71,1%
vec3	KPPV (K=11, Bray-Curtis)	69,9%
vec4	Réseau bayésien	64,9%
vec5	Table de décision	71,0%
vec6	Adaboost C4.5	72,3%

Tableau 7.11 Performances des classifieurs individuels utilisés – synthèse des classifieurs des tables 7.7 et 7.10 dans le cas leave-one-out

7.3.5.3 Combinaison paramétrique en fonction des classes prédites

Méthodologie Dans le cadre de la combinaison paramétrique, deux facteurs peuvent varier :

- la combinaison de classifieurs de premier niveau (*i.e.* ceux du tableau 7.11) : toutes les combinaisons n'ont pas été testées de manière exhaustive, mais dans le pire des cas, lorsque l'on a N classifieurs de premier niveau, si l'ajout d'un $(N+1)^e$ peut ne pas améliorer la performance globale de l'ensemble, il ne peut la réduire puisqu'il existe toujours une combinaison (avec une table ou un arbre de décision) permettant d'annuler l'effet de ce classifieur le cas échéant. Nous nous sommes en particulier intéressés à quatre sous-cas :
  1. la combinaison des classifieurs "bas" : pour évaluer les ultimes limites des paramètres basiques ;
  2. la combinaison des classifieurs "vec" : les plus performants pris de manière individuelle ;
  3. la combinaison de classifieurs "vec" et "bas" : pour vérifier si en dépit de leur simplicité, ces derniers sont susceptibles d'apporter des informations supplémentaires ;
  4. la combinaison de classifieurs "vec" et du "arb" : ce dernier ayant des performances moindres mais une approche différente, on espère une synergie possible entre les deux approches.

– le classifieur utilisé pour la combinaison : nous avons ici eu recours à une palette variée de différents types de classifieurs : arbre de décision (C4.5), bayésien naïf, réseau bayésien, table de décision et PPV, évalués avec une validation croisée leave-one-out.

Résultats Les résultats sont synthétisés dans le tableau 7.12. Pour chaque combinaison de classifieurs de premier niveau (référencés en utilisant les identifiants introduits dans le tableau 7.11), seul figure dans celui-ci le meilleur résultat et le nom du classifieur ayant permis son obtention même si tous les classifieurs ont été essayés. Les quatre sous-sections du tableau correspondent aux quatre sous-cas évoqués ci-dessus, et l'on constate que :

1. il est possible d'améliorer sensiblement les classifieurs fondés sur les paramètres basiques (56,6% au lieu de 52,4%) à l'aide d'un classifieur PPV basé sur l'ensemble de ceux-ci ;
2. il est aussi possible d'améliorer légèrement les classifieurs fondés sur la représentation vectorielle des requêtes (76,1% au lieu de 74,4%) à l'aide d'une table de décision basée sur l'ensemble de ceux-ci ;
3. l'ajout de classifieurs "bas" aux "vec" ne permet pas d'amélioration des performances globales ;
4. l'ajout du classifieur "ker" aux "vec" ne permet pas non plus d'amélioration des performances.

Classifieurs du premier niveau	Classifieur de combinaison	Résultat
bas(1+2+3)	Table de décision	53,7%
bas(1+2+3+4+5)	PPV	56,6%
vec1+vec2	Arbre de decision (C4.5)	75,2%
vec1+vec3	Arbre de decision (C4.5)	74,4%
vec[1–6]	Table de décision	**76,1%**
bas1+vec1	Arbre de decision (C4.5)	74,4%
bas1+vec2	Arbre de decision (C4.5)	71,2%
bas1+vec3	Bayésien naïf	65,3%
bas[1–5]+vec[1–6]	Table de decision	**76,1%**
vec1+arb	Table de décision	74,4%
vec[1–6]+arb	Table de décision	76,1%

Tableau 7.12 Performances des combinaisons paramétriques de différents classifieurs (leave-one-out)

Il est donc inutile d'envisager une combinaison à la fois des classifieurs "bas", "vec" et "arb", dans mesure où ni "bas", ni "arb" n'apportent une amélioration de performance par rapport à la combinaison de tous les classifieurs "vec" entre eux. Le meilleur résultat, obtenu avec une combinaison des classifieurs "vec" avec une table de décision, est détaillé dans la table 7.13 qui permet de voir qu'il offre des performance d'un niveau similaire pour les requêtes de contrôle, d'assistance et de discussion (avec une F-mesure de l'ordre de 0,78, bien que la précision soit meilleur sur le contrôle et le rappel meilleur sur les requêtes de discussion). Mais lorsque l'on considère les requêtes d'assistance directe et indirecte séparément, une fois encore,

aucun classifieur n'améliore significativement la détection des requêtes d'assistance indirecte. La matrice de confusion révèle en outre qu'elles ne sont pas seulement confondues avec de l'assistance directe (34% des cas) mais aussi avec des requêtes de contrôle (27% des cas).

Activité conversationnelle (classe)	Précision	Rappel	F-Mesure
Contrôle	0,864	0,692	**0,769**
Assistance	0,806	0,785	**0,795**
— Assistance directe	0,758	0,818	0,787
— Assistance indirecte	0,560	0,298	0,389
Discussion	0,757	0,835	**0,794**

↓ classifié comme →	Contrôle	Assistance directe	Assistance indirecte	Discussion
Contrôle	108	19	6	23
Assistance directe	3	319	8	60
Assistance indirecte	9	25	28	32
Discussion	5	58	8	359

Tableau 7.13 Résultats détaillés de la combinaison des six classifieurs de l'approche vecteur avec une table de décision (précision, rappel et matrice de confusion)

7.3.5.4 Combinaison non paramétrique en fonction des probabilités d'appartenance

Méthodologie Dans cette section, nous avons considéré les mêmes classifieurs de premier niveau que précédemment (à l'exception de celui basé sur la représentation sous forme d'arbre, non prévu ainsi) mais en exploitant cette fois le vecteur de quatre probabilités associé à chaque classe au lieu de considérer une seule classe prédite C pour la requête R correspondant à $max_{i=1}^{4}(P(R \in C_i))$.

Pour calculer la probabilité a posteriori $P(R \in C_i)$ avec L classifieurs, il y a au moins cinq règles à considérer :

- Le maximum (max) : $P(R \in C_i) = max_{j=1}^{L}(P_{i,j})$
  où Pi, j est la probabilité pour R d'être dans la classe C_i d'après le j^{e} classifieur.
- Le minimum (min) : $P(R \in C_i) = min_{j=1}^{L}(P_{i,j})$
- La médiane (med) : $P(R \in C_i) = \begin{cases} \frac{m_{i,L/2}+m_{i,1+L/2}}{2} & \text{si } L \text{ est pair} \\ m_{i,\frac{(L+1)}{2}} & \text{si } L \text{ est impair} \end{cases}$
  si m_i est l'ensemble des probabilités associées à la classe C_i ordonnées de manière croissante, c'est-à-dire du classifieur donnant la plus faible probabilité à R d'être dans C_i à celui donnant la plus forte (*i.e.* $m_i = \{P_{i,j_1}, P_{i,j_2}, \ldots, P_{i,j_L}\}$ avec $P_{i,j_m} \leq P_{i,j_n}$ si et seulement si $m < n$).
- Le produit ($\prod$) : $P(R \in C_i) = \prod_{j=1}^{L} P_{i,j}$
- La somme ($\sum$) : $P(R \in C_i) = \sum_{j=1}^{L} P_{i,j}$

Nous avons testé ces 5 règles de manière exhaustive sur les 2 047 sous-ensembles possibles des classifieurs de l'ensemble {bas[1–5], vec[1–6]}.
Un exemple de comparaison entre ces méthodes dans le cas de la requête de contrôle R_{ex_1} figure dans le tableau 7.14, où il apparaît qu'elle n'est classifiée correctement que par la règle max, les autres prédisant qu'il s'agit d'une requête de discussion.

Activité conversationnelle (classe)	bas1	bas2	bas3	Max	Min	Med	$\prod$	$\sum$
Contrôle	0,37	0,39	0,47	**0,47**	0,37	0,39	0,068	1,23
Assistance directe	0,16	0,13	0,14	0,16	0,13	0,14	0,003	0,43
Assistance indirecte	0,05	0,02	0,01	0,05	0,01	0,02	0,000	0,08
Discussion	0,42	0,46	0,38	0,46	**0,38**	**0,42**	**0,073**	**1,26**

Tableau 7.14 Résultats de combinaisons non paramétriques de classifieurs basiques pour la requête de contrôle R_{ex_1}

Résultats Les combinaisons minimales (en termes de nombre de classifieurs) donnant les meilleurs résultats pour chaque règle sont données dans la figure 7.15. Il apparaît que seule la règle de somme permet d'obtenir une performance globale meilleure que "sem1" pris individuellement, avec une amélioration très marginale.

Règle	Classifieurs de premier niveau	Résultat
Max	sem2+sem3+sem5	72,4%
Min	bas2+sem2+sem5	72,9%
Med	sem1+sem4+sem5	73,9%
$\prod$	bas1+sem1+sem2+sem4+sem5	73,1%
$\sum$	sem1+sem2+sem3+sem4+sem5+sem6	**74,5%**

Tableau 7.15 Meilleurs résultats de combinaisons non paramétriques de classifieurs, pour chaque règle de calcul de la probabilité a posteriori d'appartenance à une classe

7.3.5.5 Conclusion

La combinaison de classifieurs permet effectivement une amélioration des performances globales du classifieur : toutefois, alors qu'on espérait une complémentarité entre l'utilisation des représentations d'une requête sous forme d'arbre et de vecteur, c'est finalement la combinaison de classifieurs uniquement basés sur la représentation sous forme de vecteur qui permet cette amélioration. Cela confirme le résultat précédent qui laisse penser que pour l'identification d'activités conversationnelles, la sémantique des éléments prime sur leur organisation structurelle.

7.4 Conclusion

Dans ce chapitre, nous avons vu qu'il était possible d'analyser les requêtes des utilisateurs à l'agent assistant en termes d'activités conversationnelles, parmi lesquelles on peut distinguer en particulier les requêtes de contrôle, d'assistance (directe et indirecte) et de discussion. Bien que de par sa constitution on s'attende à ce que le corpus *Daft* contienne une majorité de requêtes d'assistance, le contrôle (15%) et la discussion (40%) ne peuvent ignorés. Cette dernière catégorie de requêtes, dont la proportion est renforcée par la présence d'un agent personnifié, est particulièrement problématique car elle remet en cause notre objectif de couper dans la complexité de la langue naturelle. À défaut de traiter ces requêtes de discussion, il est donc essentiel de pouvoir identifier comme telles les requêtes ne relevant pas de l'assistance, dans le cadre d'un prétraitement. Nous avons donc eu recours à de nombreuses méthodes de classification exploitant diverses caractéristiques (longueur des phrases et des requêtes, profondeur et nombre d'entités dans celles-ci) et représentations (sous forme d'arbre ou de vecteur) des requêtes. La représentation sous forme de vecteur en fonction des entités constituant la requête (bien que moins riche que celle sous forme d'arbre) apparaît comme la plus efficace : en combinant plusieurs classifieurs exploitant cette forme, on obtient une performance de 76,1% de requêtes correctement classifiées. La distinction entre requêtes d'assistance directes et indirectes reste toutefois difficile à effectuer, sauf si l'on se base sur les actes de dialogues searliens des requêtes, mais ceux-ci ne peuvent cependant pas être annotés parfaitement de manière automatique.

Enfin notons que si les activités conversationnelles considérées ici restent d'assez haut niveau, il serait tout à fait envisageable de réemployer cette même méthodologie pour associer directement une requête mise sous forme formelle à une réaction de l'agent rationnel, par exemple pour venir se substituer aux règles régissant celui-ci lorsqu'aucune d'entre elles n'est applicable, alliant ainsi approche par règles et approche statistique.

Chapitre 8

Évaluations d'usage du système : agents sur le Web et agent rationnel

Sommaire

En parallèle du travail réalisé sur le développement d'un agent conversationnel assistant, centré sur la chaîne de traitement de requêtes en langue naturelle décrite dans la seconde partie de cette thèse, nous avons été amenés à exploiter certains de ces éléments dans d'autres contextes. Au-delà du système `DAFT` à proprement parler, ces travaux connexes nous permettent donc d'évaluer d'une part la réutilisabilité des différentes ressources constituées (clés sémantiques $\mathbb{K}_G$, langage formel *DAFT*, *etc.*) et, d'autre part, la méthodologie suivie. Nous envisageons enfin les besoins, en termes d'architecture, de l'agent rationnel $\mathtt{A}_R$ qui devra traiter les requêtes retranscrites en *DAFT* et constituera à terme le dernier module du système `DAFT`.

8.1 Expériences complémentaires sur le Web

8.1.1 Agents DIVA

8.1.1.1 Le besoin d'agents Web

Nous avons vu, dans la section 2.2.4, que l'utilisation des agents animés DIVA constitués d'images PNG transparentes, de feuilles de style CSS et de JavaScript pour le moteur d'animation permettent une intégration relativement aisée d'agents conversationnels sur une page web. Ils peuvent être connectés à la sortie de la chaîne de traitement `GRASP/DIG` de manière à exploiter la requête ainsi produite en *DAFT*. Toutefois, `GRASP` et `DIG` ayant été développés sous l'environnement *Mathematica*, ils nécessitent d'être installés sur un serveur distant, ce qui a un coût non négligeable (financier - *Mathematica* n'étant pas libre, mais aussi en terme de temps et de compétences pour mettre en place le système).
Afin de fournir un outil complet permettant à un concepteur d'applications Web de déployer facilement un agent conversationnel pour assister son application, nous avons donc été amenés à développer en parallèle de la chaîne de traitement décrite dans la seconde partie de cette thèse (complétée par l'agent rationnel $\mathtt{A}_R$ envisagé dans la section 8.2) des agents conversationnels reposant sur une approche de type "chatbot" et fonctionnant de manière indépendante sur une page Web. C'est à partir de cette chaîne de traitement, dont l'architecture est présentée sur la figure 8.1 et que nous allons détailler ci-dessous, que les agents employés dans les sections 8.1.2 et 8.1.3 ont été développés.

8.1.1.2 Analyse linguistique de requêtes par un chatbot

Travaux liés. Les agents conversationnels de type chatbot sont les lointains descendants d'Eliza [Weizenbaum, 1966] qui fut à l'origine conçu pour l'interaction textuelle entre des humains et un programme simulant un faux psychologue rogérien [Rogers, 1957]. Aujourd'hui, parmi les chatbots Web les plus connus on peut citer HAL[1], Jabberwacky[2], ALICE[3] ou Elbot[4]. Cette technologie est utilisée dans de nombreux sites pour l'accueil du grand public ou dans le cadre d'interactions purement ludiques, mais rarement pour fournir de l'Assistance au sens où nous l'entendons dans le projet DAFT. Ceci s'explique essentiellement par la trivialité des outils de TALN employés, essentiellement fondés sur de la reconnaissance de mots-clés[5], et donc incompatibles avec la nécessité d'une compréhension plus poussée des requêtes pour fournir une assistance utile.

[1] `http://zabaware.com`
[2] `http://www.jabberwacky.com`
[3] `http://alicebot.blogspot.com`
[4] `http://www.elbot.com`
[5] Même si ALICE [Wallace, 2008] offre une approche plus poussée à base de règles exprimées en Artificial Intelligence Markup Language (AIML) [Wallace, 2003].

Dans une étude menée sur quatre des principaux chatbots en activité (ALICE, EllaZ[6], Elbot et ULTRAHAL), Wollermann [2004] et Wollermann [2006] s'est intéressée à l'évaluation qualitative des réponses fournies par ces chatbots d'un point de vue sémantique (relations sémantiques, quantifieurs, anaphores...) et pragmatique (essentiellement du point de vue des maximes de Grice [1975]). Tous présentent de graves problèmes sémantiques et pragmatiques, et l'étude se conclue sur la nécessité d'une prise en compte plus fine de la langue naturelle dans le cas d'un dialogue finalisé. Nous reviendrons sur ces limites dans la section 8.1.2.

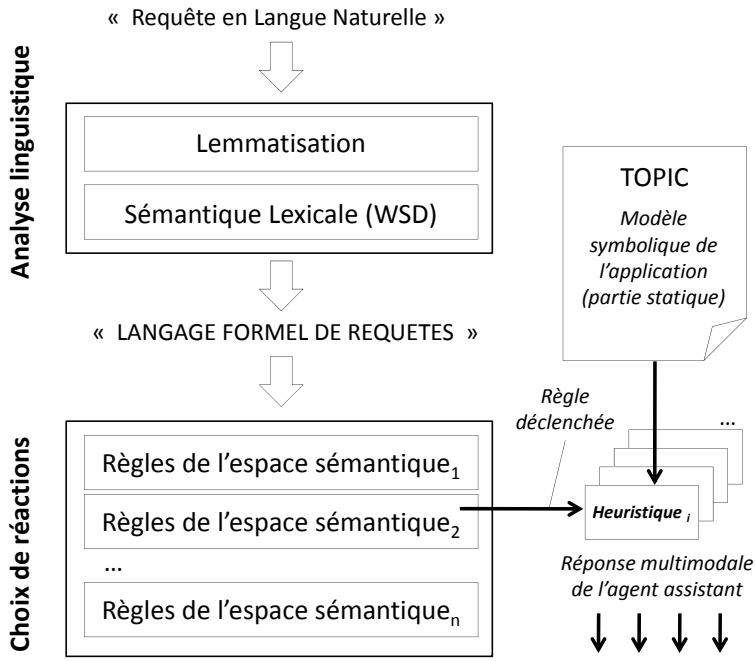


Figure 8.1 Architecture des agents Web DIVA

L'analyseur `GRIP`. Considérant que l'assistance peut être traitée comme une interface en langue naturelle (comme discuté en 3.1.3.4), on peut cependant espérer obtenir quelques résultats avec une approche "chatbot", à condition de lui apporter quelques améliorations substantielles. En parallèle de `GRASP` et `DIG`, nous avons donc proposé un moteur d'analyse simplifié (entièrement en Javascript), que nous nommerons ici `GRIP` pour le distinguer.
En termes de complexité, ce moteur d'analyse se rapproche davantage d'une architecture de type chatbot que de celle de `GRASP`, bien qu'il ait recours à des éléments généralement absents dans les chatbots classiques comme l'utilisation de classes sémantiques (*cf.* le tableau 8.1).

Le fonctionnement détaillé de `GRIP` au sein des agents DIVA a été décrit au sein de [Xuetao *et al.* , 2008] et [Xuetao *et al.* , 2009b]. Pour résumer, en termes de lemmatisation, `GRIP` se contente des opérations les plus simples ne nécessitant pas de lexique (suppression des accents, gestion des pluriels réguliers et suppression des élisions). Après cela, les lemmes sont transformés en clés sémantiques à l'aide d'un ensemble de règles de filtrage par motif (pattern matching) au format XML de la forme :

[6] `http://www.ellaz.com/AI/Talk.aspx`

Fonctionnalité	GRASP/ DIG	GRIP	Chatbot classique
Langue	Français	Français ou Anglais	N/A
Lemmatisation	Avancée	Simple	Simple
→ Lexique	$\mathbb{L}_G$: 4 266 entrées	Non	Possible
Analyse grammaticale	Oui	Non	Non
→ Règles	$\mathbb{R}_G$: 240 règles	-	-
Sémantique lexicale	Oui	Oui : analyseur simplifié	Non
→ Synsets	$\mathbb{K}_G$: 1 294 clés	Sous-ensemble de $\mathbb{K}_G$: 412 clés	-
Sémantique de la requête	Avancée	Simple (non structurée)	Non
→ Langage formel	*DAFT*	Non	-
→ Éléments	$\mathbb{S}$: 237 schèmes	-	-

Tableau 8.1 Table de comparaison des chaînes d'analyse de requêtes en langue naturelle GRASP, GRIP et d'un chatbot classique

```
<rule id= "ruleid" pat="RegularExpression" if="condition" go="continuation">
      <filter>[w1, w2, ..., wn]</filter>
</rule>
```

où :

- `id` est un attribut contenant le nom unique de la règle,
- `pat` est un attribut contenant une expression régulière utilisée pour le filtrage par motif,
- `if` est un attribut optionnel contenant une expression booléenne, le déclenchement de la règle ne se faisant que si la condition est vraie,
- `go` est un attribut optionnel indiquant l'ordre d'application des règles (si l'enchaînement par défaut est modifié),
- `<filter>` est une balise contenant la forme de sortie réécrite, composée d'une suite d'éléments `wi` pouvant être des clés ou des sous-ensembles de la forme d'entrée, désignés par leur position (*e.g.* "1" pour le premier élément capturé par la forme d'entrée).

Exemples de règles de filtrage de GRIP :

```
<rule id="7" pat="&lt;(.*) (button|brooch|clasp|badge)s? (.*)&gt;" go="NEXTRULE">
      <filter>[1,"THEBUTTON",3]</filter>
</rule>

<rule id="neg1" pat="&lt;(.*)( am | are | is | were )not (.*)&gt;" go="NEXTRULE">
      <filter>["NEG","BE",1,3]</filter>
</rule>
```

La première règle permet de transformer les références à un élément de type bouton de l'interface graphique en la clé sémantique THEBUTTON (telle que définie dans $\mathbb{L}_G$), la seconde fait partie des règles permettant de gérer la négation inversée en anglais.

Contrairement à GRASP, il n'y a pas de surcouche équivalente à DIG permettant de générer une requête structurée sémantiquement : celle-ci reste donc sous la forme d'une suite de clés partiellement ordonnée, et éventuellement de mots de la langue naturelle (puisque les mots non traités par une règle restent dans la requête sous forme lemmatisée).

8.1.1.3 Modélisation de l'application assistée

Le modèle de l'application et de l'agent, auquel a accès l'agent pour répondre aux requêtes des utilisateurs et à renseigner manuellement par le concepteur de l'agent ou de l'application, est fourni sous la forme d'un fichier dit de TOPIC (également sous forme XML), par exemple :

```
<xml>
  <type gloss="intrinsic nature of the entity">PERSON</type>
  <name gloss="a human-defined name to refer to this entity">Marco</name>
  <gender gloss="gender of the entity if any: male, female, unknown">male</gender>
  <purpose gloss="main objective of the topic">agent d'accueil du GT ACA</purpose>
  ...
</xml>
```

8.1.1.4 Définition de réactions associées aux requêtes

Les règles de réaction (réponse de l'agent à la requête, animation de l'agent DIVA et éventuellement action dans l'application) sont directement associées à une séquence de clés sémantiques, encore une fois avec un filtrage par motif. Par exemple, pour traiter le nom de l'utilisateur, on peut définir la règle suivante :

```
 <rule id="on-username" pat="&lt; USERNAME BE (\w+) &gt;" >
    <do>
       THETOPIC.x = TALK_capitalizefirst(TALK_getmatch(1));
       If (THETOPIC.x == THEUSER.name) {     // détection de la répétition
          TALK_say(['I knew it already','You said it!']);  // tirage au sort
          TALK_decrease(THEAGENT.coop, 0.1);               // diminue cooperativité
          TALK_exit();                       // évite que <say> ne soit traitée
       }
       else {
          THEUSER.name = THETOPIC.x;         // mémorisation
          TALK_increase(THEAGENT.coop, 0.5); // augmentation de la coopérativité
       }
    </do>
    <say>                                                 // tirage au sort
       <p>From now, I will call you _THETOPIC.name_.</p>
       <p>Ok, your name is _THETOPIC.name_...</p>
       <p>OK you are _THETOPIC.name_.</p>
    </say>
 </rule>
```

Elle réagit aux phrases de la forme "My name is John" en définissant un ensemble de réponses possibles (balise `<say>`) et en permettant de scripter (en Javascript) des actions complémentaire à réaliser (balise `<do>`). Ici, il s'agit de mémoriser l'information, changer la variable représentant la coopérativité de l'agent et de détecter les répétitions éventuelles de cette phrase par l'utilisateur.

8.1.2 Étude de l'impact de facteurs linguistiques sur l'acceptabilité d'un agent conversationnel

Comme nous le verrons dans la section 8.1.3.2 et comme en témoignent les exemples d'agents développés par des étudiants de Master sur la page Web du projet[7], les agents DIVA sont rapidement déployables sur des pages Web et fournissent donc un bon support d'expérimentation. Dans la mesure où il nous est apparu lors des différentes campagnes de recueil de corpus décrites dans le chapitre 3 que même lorsque l'agent se montre compétent et précis dans les réponses qu'il fournit à l'utilisateur, il ne parvient pas à être perçu comme humainement crédible et amical, nous avons souhaité mesurer l'impact des limites généralement présentes dans les chatbots classiques. Nous résumons donc dans cette sous-section l'expérience élaborée à cet effet et présentée plus en détail dans Xuetao *et al.* [2009a].

8.1.2.1 Problématique

Nous nous intéressons ici essentiellement aux requêtes dites "de discussion" (*cf.* figure 7.6), que nous avions laissées de côté jusqu'à maintenant puisque c'est pour répondre à ce type de requêtes que les architectures de type chatbot sont généralement utilisées. D'après les retours des utilisateurs, trois facteurs semblent porter particulièrement préjudice à la naturalité des échanges dialogiques avec un agent conversationnel, dont l'étude relève clairement de la pragmatique du dialogue [Gazdar, 1979; Horn & Ward, 2005]. Ces trois facteurs sont :

Une Responsivité (R) trop systématique. Les utilisateurs rapportent souvent leur étonnement devant l'obéissance excessive d'un agent purement rationnel, qui fournit toujours la réponse aux questions dont il connaît la réponse, indépendamment de son rôle, de ce qu'il s'est passé précédemment ou de la nature éventuellement "confidentielle" de l'information demandée, comme illustré par les colonnes 1 et 2 du tableau 8.2.

Un manque de Variabilité linguistique (V). Le manque de variabilité dans les réponses est un phénomène permettant immédiatement d'identifier que l'on est face à un agent artificiel (*cf.* colonne 1 du tableau 8.2). Ce problème a notamment été étudié dans le contexte des réponses évasives pour des questions hors-sujet par Patel *et al.* [2006] et Artstein *et al.* [2008]

[7] http://www.limsi.fr/~jps/online/diva/divahome/

qui ont montré que la répétition de “Je ne sais pas” pouvait briser l'immersion et mettre fin à l'interaction de manière prématurée.

Un manque de Dépendance (D) entre les réponses. Lorsqu'un utilisateur pose plusieurs fois une même question (d'un point de vue sémantique), par défaut l'agent traite celle-ci comme si c'était la première fois qu'elle lui était posée par cet utilisateur. Ainsi, dans la seconde colonne du tableau 8.2, l'insistance pour connaître l'âge de l'agent (chaque question serait en effet transformée par `GRASP` et `DIG` en une même requête donnée par la figure 8.2) serait perçue par un humain comme étrange voire indiscrète et malpolie, et devrait donc susciter une réaction différente (comme dans la troisième colonne de ce même tableau).

$R \wedge \overline{V} \wedge \overline{D}$	$R \wedge V \wedge \overline{D}$	$R \wedge V \wedge D$
U_1 : Comment t'appelles-tu ?	U_1 : Puis-je savoir ton âge ?	U_1 : Puis-je savoir ton âge ?
A_1 : Cher ami, je m'appelle Elsi.	A_1 : J'ai 25 ans	A_1 : 25 ans
U_2 : Peux-tu me rappeler ton nom ?	U_2 : Quel âge as-tu ?	U_2 : Quel âge as-tu ?
A_2 : Cher ami, je m'appelle Elsi.	A_2 : Je suis âgée de 25 ans	A_2 : Comme je l'ai dit, 25 ans
...	U_3 : Connais-tu ton âge ?	U_3 : Connais-tu ton âge ?
...	A_3 : 25 ans	A_3 : Je sais que tu le sais !
U_{30} : Ton nom est Elsa ?	U_4 : Quel est ton âge ?	U_4 : Quel est ton âge ?
A_{30} : Cher ami, je m'appelle Elsi.	A_4 : J'ai 25 ans, cher ami	A_4 : Ce n'est pas drôle...
...	...	...

Tableau 8.2 Trois exemples construits de dialogues humain-agent illustrant les problèmes de responsivité (R), variabilité (V) et dépendance (D)

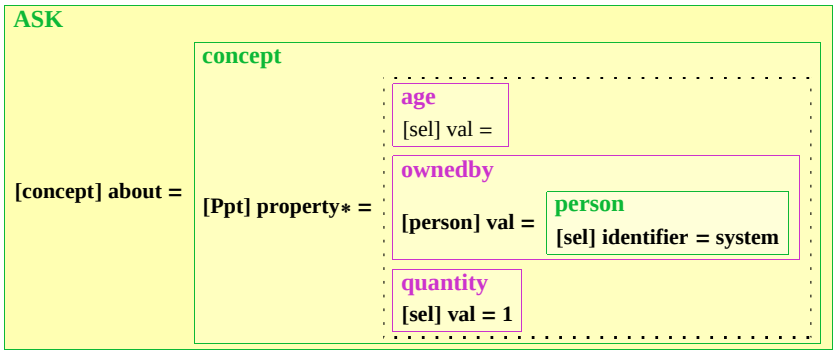


Figure 8.2 Analyse de la requête “Quel est ton âge ?” par `DIG`

Trois questions. Le tableau 8.2 illustre ces trois phénomènes sur des exemples construits. Notons que dans l'exemple de la troisième colonne, c'est la dépendance qui rend l'agent non coopératif en A_3 et A_4, alors qu'il l'était à l'origine (réponses A_1 et A_2). Nous désignerons par la suite les agents présentant ou non ces différents facteurs sous une forme de 3 lettres, *e.g.*$RV\overline{D}$ pour un agent responsif avec variabilité linguistique mais sans dépendance à la session dans les réponses qu'il fournit ($\overline{X}$ signifiant l'absence du facteur X). Trois questions se posent alors :

Q1 Faisabilité : est-il possible de définir des règles pour les agents DIVA de manière à tester l'absence ou la présence de ces trois facteurs ?

Q2 Perceptibilité : mis en présence d'un agent RVD, les utilisateurs l'identifient-ils consciemment comme différent d'un autre type d'agent ?

Q3 Acceptabilité : mis en présence d'un agent RVD, et indépendamment du fait qu'ils perçoivent ou non une différence, jugent-ils l'agent plus humain et amical ?

La section 8.1.1.4 permet de répondre par l'affirmative à Q1, puisqu'on a vu dans l'exemple de règle donné qu'il était possible de gérer la variation linguistique (par tirage au sort parmi une liste de réponses possibles) et de mémoriser si une question portant sur la valeur d'un paramètre donné a déjà été posée. De plus, il est également facile de définir une règle empêchant de fournir la valeur de certaines informations jugées "personnelles".
Pour répondre à Q2 et Q3, nous proposons l'expérience décrite dans la section suivante.

8.1.2.2 Expérience

Principe général. L'expérience consiste à laisser interagir les utilisateurs librement avec différentes versions d'un même agent (*i.e.* utilisant le même fichier TOPIC d'informations, sur une page Web identique et avec un agent virtuel animé identique) implémentant ou non les trois facteurs R, V et D. Après ces interactions (qu'on suggère aux sujets de conserver assez courtes), il leur est demandé d'évaluer ces agents selon sept paramètres P_1 à P_7 (listés plus bas).

Différences inter-agents. Les 3 facteurs R, V et D étudiés ne sont pas complètement indépendants les uns des autres : ainsi D étant une forme avancée de variabilité, ce facteur ne peut être implémenté que si V l'est ; par contre, V peut exister que l'agent réponde ou non à la question posée. Par conséquent, nous pouvons définir les comportements conversationnels effectifs suivants : RVD, $RV\overline{D}$, $R\overline{V}$, $\overline{R}VD$, $\overline{R}V\overline{D}$ et $\overline{RV}$.
Nous avons cependant choisi de n'implémenter ces différences de comportement que sur un unique attribut : l'âge de l'agent. En effet, l'âge peut être considéré comme ayant une certaine confidentialité, surtout dans le cas d'un agent de sexe féminin[8], et donc justifier un manque de responsivité. La différence inter-agents est donc globalement faible, mais d'une part l'âge est une des questions les plus posées dans un scénario de rencontre d'une nouvelle personne sur Internet (*cf.* le phénomène ASL (Age/Sex/Location) [Merchant, 2001]), et d'autre part implémenter ce comportement sur tous les paramètres de l'agent nous a semblé susceptible de différencier beaucoup trop les agents les uns des autres. On peut observer les différences de réponse aux questions sur l'âge de l'agent, pour chacun des six comportements conversationnels, dans les exemples du tableau 8.3. Il est suggéré aux sujets de poser plusieurs fois les

[8] En effet, les personnes interagissant avec un agent conversationnel lui associent des stéréotypes liés à son genre encore plus qu'ils ne le font avec de vrais humains [Angeli & Brahnam, 2006], ce qui en fait un paramètre de choix pour cette expérience.

mêmes questions (même s'ils demeurent libres de ne pas le faire).
De plus, un sujet interagit successivement avec seulement deux agents : en effet, utiliser plus de deux agents aurait été prendre le risque d'une perte de motivation lors des dernières interactions, tandis qu'un seul agent n'aurait pas permis de mesurer les différences interpersonnelles lors de la phase d'évaluation. Considérant l'agent RVD comme notre cas "idéal" ou cas de référence, chaque sujet interagit avec lui et avec un des cinq autres agents (soit 10 combinaisons possibles, puisque pour prendre en compte l'ordre d'exposition, l'agent RVD peut être le premier ou le second avec lequel le sujet interagit).

Agent	Première réponse	Deuxième réponse	Troisième réponse
RVD	25 ans	Je t'ai dit 25 ans	Je ne vais pas répondre encore une fois
$RV\overline{D}$	25 ans	J'ai 25 ans	Je suis âgée de 25 ans
$R\overline{V}$	25 ans	25 ans	25 ans
$\overline{R}VD$	Je ne le dirai pas	J'ai dit que je ne le dirai pas	Arrête d'insister à ce sujet !
$\overline{R}V\overline{D}$	Je ne le dirai pas	Je ne répondrai pas	Je préfère ne pas répondre
$\overline{RV}$	Je ne le dirai pas	Je ne le dirai pas	Je ne le dirai pas

Tableau 8.3 Exemples construits de réponses possibles à la question "quel âge as-tu ?" posée trois fois lors d'une même session dialogique en fonction des divers comportements conversationnels de l'agent

Évaluation des interactions. Nous demandons aux sujets après chacune des deux interactions d'évaluer l'agent selon sept paramètres en remplissant un questionnaire, que nous nommerons Q_1 pour le questionnaire donné après la première interaction, et Q_2 pour celui donné après la seconde (les questions de Q_1 et Q_2 sont donc identiques). Après les deux interactions, il leur est également demandé de remplir un troisième questionnaire (Q_3), où ils doivent comparer les deux agents selon ces sept mêmes paramètres. Ces deux questionnaires (Q_1/Q_2 et Q_3) sont donnés en annexe G. Ces sept paramètres, corrélés aux trois facteurs (R, V et D) utilisés, sont :
P_1 Précision : "25 ans" *vs* "la vingtaine" ;
P_2 Pertinence : "25 ans" *vs* "Il fait beau aujourd'hui" ;
P_3 Crédibilité : "25 ans" *vs* "142 ans" (peu crédible au regard l'agent animé utilisé représentant une jeune femme) ;
P_4 Humanité perçue, *i.e.* "était-il évident qu'il s'agissait d'un agent et non d'un humain ?" ;
P_5 Variabilité : "25 ans" puis "25 ans" *vs* "25 ans" puis "Je suis âgée de 25 ans" ;
P_6 Coopérativité : "25 ans" *vs* "Je ne le dirai pas" ;
P_7 Satisfaction générale quant à l'interaction avec l'agent.

L'évaluation individuelle se fait selon une échelle de Likert à 5 niveaux, tandis que la comparaison est réalisée en choisissant le meilleur agent parmi les deux, si un a été perçu comme meilleur au regard du paramètre considéré (avec possibilité de choisir "aucun" si nécessaire).

8.1.2.3 Résultats

21 sujets (14 hommes et 7 femmes) âgés de 20 à 60 ans ont pris part à l'expérience. Majoritairement d'un niveau universitaire, ils avaient en revanche des connaissances diverses en informatique. Ainsi, pour la moitié d'entre eux cette expérience constituait leur premier contact avec un agent conversationnel.
Les résultats de la comparaison des questionnaires Q_1 et Q_2 en prenant comme hypothèse pour effectuer un test de Student ($\alpha = 0.05$) H_0 : "$\mu_{\text{paramètre, agent } RVD} = \mu_{\text{paramètre, autre agent}}$" sont donnés dans la table 8.4[9]. L'analyse des questionnaires Q_3 (comparant explicitement les deux agents) est quant à elle synthétisée dans la table 8.5.

Agent		RVD	$RV\overline{D}$	$R\overline{V}$	$\overline{R}VD$	$\overline{R}V\overline{D}$	$\overline{RV}$
Précision	$\overline{x}$	2,78	2,5	**3,5**	2,2	1,8	2,25
	σ	1,06	2,12	1	0,45	0,84	0,5
Pertinence	$\overline{x}$	2,72	2	**3,25**	2,4	3	2,25
	σ	1,22	0	0,96	0,55	1,22	0,96
Crédibilité	$\overline{x}$	3,39	**4,5**	3,75	3,2	3,4	3,5
	σ	1,04	0,71	0,5	1,48	1,34	0,58
Humanité	$\overline{x}$	2,72	3	**3,75**	2,8	3	2,25
	σ	1,13	2,83	0,5	1,30	1,22	1,26
Variabilité	$\overline{x}$	**3,06**	3	3	2,2	3	2,25
	σ	1,39	2,82	1,15	1,64	1,58	1,26
Coopérativité	$\overline{x}$	**2,44**	1,5	1,75	2,4	1,4	1,25
	σ	1,25	0,71	0,96	1,95	0,55	0,5
Satisfaction	$\overline{x}$	2,83	2,5	**3,5**	2,8	2,4	1,75
	σ	1,25	0,71	1	1,10	1,34	0,96
Nombre de sujets	N	18	2	4	5	5	4

Tableau 8.4 Moyenne ($\overline{x}$) et écart-type (σ) des notes données (de 1 à 5) dans les questionnaires post-interactions Q_1 et Q_2 pour chacun des six agents.
• Souligné : valeurs statistiquement différentes de RVD (rejet de H_0).
• Gras : meilleure évaluation pour le paramètre considéré.

8.1.2.4 Analyse et discussion

L'agent RVD est le plus acceptable.

Les sujets perçoivent une différence : Pour répondre à la question Q2, les sujets perçoivent généralement une différence entre les deux agents avec lesquels ils ont interagit puisqu'il y a rarement égalité entre les deux agents (sauf peut-être RVD et $RV\overline{D}$ qui sont proches).

[9] Les sujets ayant été en partie contactés par Internet (dans l'expérience, l'agent est placé dans une page Web, à la manière de la figure 3.3), certains sujets ne sont pas allés jusqu'au bout de l'expérience, d'où le déséquilibre entre les différentes paires d'agents testés.

Agents comparés	Précision	Pertinence	Crédibilité	Humanité	Satisfaction
$RVD > RV\overline{D}$	0	0	**50**	0	**50**
$RVD < RV\overline{D}$	0	0	0	0	0
$RVD = RV\overline{D}$	100	100	50	100	50
$RVD > R\overline{V}$	25	25	25	**25**	**25**
$RVD < R\overline{V}$	25	25	25	0	0
$RVD = R\overline{V}$	50	50	50	75	75
$RVD > \overline{R}VD$	**100**	**100**	**66,7**	33,3	**66,7**
$RVD < \overline{R}VD$	0	0	0	**66,7**	33,3
$RVD = \overline{R}VD$	0	0	33,3	0	0
$RVD > \overline{R}V\overline{D}$	**60**	**40**	**40**	**40**	**60**
$RVD < \overline{R}V\overline{D}$	0	20	20	20	20
$RVD = \overline{R}V\overline{D}$	40	40	40	40	20
$RVD > \overline{RV}$	**25**	25	**50**	**50**	**50**
$RVD < \overline{RV}$	0	**50**	25	0	25
$RVD = \overline{RV}$	75	25	25	50	25
$RVD >$ tous	**44,4**	**38,9**	**44,4**	**33,3**	**50,0**
$RVD <$ tous	5,6	22,2	16,7	16,7	16,7
$RVD =$ tous	50,0	38,9	38,9	50,0	33,3

Tableau 8.5 Analyse comparative des agents RVD par rapport aux autres :
• $RVD > N$ (resp. $<$) : % d'utilisateurs pensant l'agent RVD meilleur (resp. moins bon) que l'agent N pour le paramètre P.
• $RVD = N$: % d'utilisateurs pensant les agents RVD et N identiques pour P.
• Gras : l'agent RVD a été globalement préféré à l'agent N pour P.

RVD est le plus satisfaisant : Pour tous les paramètres testés, l'agent au comportement RVD est globalement mieux évalué que les autres, particulièrement en termes de satisfaction générale (*cf.* la dernière ligne du tableau 8.5). En prenant les paramètres individuellement, il est aussi meilleur que tout autre agent sur l'ensemble des paramètres (*cf.* partie supérieure du tableau 8.5), ce qui confirme notre hypothèse de départ.
On peut voir plus particulièrement que la variabilité linguistique améliore le caractère "humain" de l'agent (RVD *vs* $R\overline{V}$), tandis que la dépendance à la session améliore la crédibilité (RVD *vs* $RV\overline{D}$).

Deux phénomènes plus surprenants.

Le manque de responsivité peut rendre l'agent plus "humain" : L'agent $\overline{R}VD$ est perçu comme le plus humain, ce qui s'explique sans doute par le choix de l'attribut "âge" comme unique paramètre changeant, et confirme que les utilisateurs inteprètent dans ce cas l'absence de réponse comme un comportement stéréotypé typiquement humain.
Il est intéressant de voir aussi que les agents $\overline{R}V\overline{D}$ et $\overline{RV}$ ont eux un score sensiblement plus bas : on peut supposer qu'en l'absence de capacité à varier ses réponses ou détecter les questions déjà posées, les utilisateurs interprètent différemment la non-réponse concernant l'âge. Si l'agent s'est montré capable de comportements "complexes" auparavant, l'absence de réponse sur l'âge est considérée comme une illustration supplémentaire de son "intelligence", mais s'il n'est même pas capable de varier ses réponses, ce même comportement sera interprété comme une preuve supplémentaire de son incompétence.

Les utilisateurs sont habitués aux agents serviles : Le fait qu'après une interaction les utilisateurs attribuent la note la plus élevée à l'agent coopératif sans variabilité linguistique $R\overline{V}$ pour plusieurs paramètres est plus étonnant (*cf.* tableau 8.4) : il est en effet paradoxal qu'ils préfèrent l'agent RVD aux autres mais le notent moins bien individuellement. Nous pouvons supposer que, par défaut, les utilisateurs s'attendent à ce qu'un système informatique soit coopératif, voire servile. Mais une fois passé l'effet de surprise lié au comportement de l'agent RVD (ou $\overline{R}VD$), ils préfèreraient sans doute interagir avec celui ayant un comportement plus "humain" sur le long terme.

8.1.3 Intégration d'un agent assistant au sein d'un environnement Web collaboratif de conception musicale

Dans le cadre du projet franco-brésilien COFECUB PRAIA[10] portant sur la création d'agents rationnels et affectifs, nous avons eu l'occasion de tester l'intégration effective d'un agent DIVA en tant qu'assistant pour une application Web réelle pré-existante. En effet, dans

[10] http://gia.inf.ufrgs.br/praia/

la section 8.1.2, nous avons vu que l'architecture des agents DIVA les rendaient capables d'apporter certaines améliorations par rapport aux chatbots classiques sur des requêtes de discussion, mais il reste encore à vérifier si le moteur d'analyse `GRIP` est aussi adapté au traitement de requêtes d'assistance directe simples.
Cette intégration a permis en premier lieu de valider la faisabilité d'une telle tâche sur une application de taille conséquente, et surtout de systématiser la procédure en 3 étapes employée :

1. collecte d'un corpus de requêtes,
2. analyse du corpus pour l'identification du vocabulaire propre à l'application,
3. définition de règles de réaction associées.

Cette section résume les travaux publiés dans [Miletto *et al.* , 2009] et [Sansonnet *et al.* , 2009].

8.1.3.1 CODES

COoperative Music Prototype DESign (CODES) est un environnement Web (développé en FLEX™) permettant la conception de morceaux musicaux de manière collaborative entre plusieurs musiciens novices. Au sein de l'écran principal de composition (représenté sur la figure 8.3), un morceau est créé par l'ajout (par glisser-déposer) de séquences musicales (représentées par des icônes d'instruments choisis en bas de la fenêtre) dans l'environnement de travail (la grille). Le morceau est lu de gauche à droite à un tempo donné (la barre rouge verticale indique la position actuelle de la lecture), de telle manière que deux séquences situées sur la même colonne sont jouées en même temps. Une fois un morceau créé, son créateur peut inviter d'autres utilisateurs Web de CODES (liste en bas à gauche) à venir l'éditer via le partage sur Internet. Pour plus de détails sur l'interface de CODES, voir Miletto *et al.* [2007].

8.1.3.2 Intégration des agents DIVA dans CODES : CODIVA

D'un point de vue purement technique (interfaçage entre la technologie propriétaire utilisée (FLEX) et Javascript), l'ajout des agents DIVA au sein de l'environnement CODES pour former la plate-forme CODIVA (*cf.* figure 8.4) n'a pas posé de problème particulier. Nous allons donc détailler ici essentiellement la méthodologie suivie pour définir les ressources linguistiques utilisées dans les règles de l'agent.

Protocole expérimental de collecte de corpus. L'expérience réalisée avant l'ajout de l'agent DIVA dans CODES vise à collecter un ensemble représentatif de requêtes d'usagers novices accomplissant des tâches simples au sein de CODES, dont l'analyse manuelle doit permettre la définition des règles de réaction de l'agent.
Nous avons fait appel à 12 sujets (8 femmes/4 hommes) entre 23 et 35 ans, étudiants dans un domaine autre que l'informatique ou la musique et sans compétences particulières dans ces deux domaines. Chaque sujet passe l'expérience de manière individuelle.

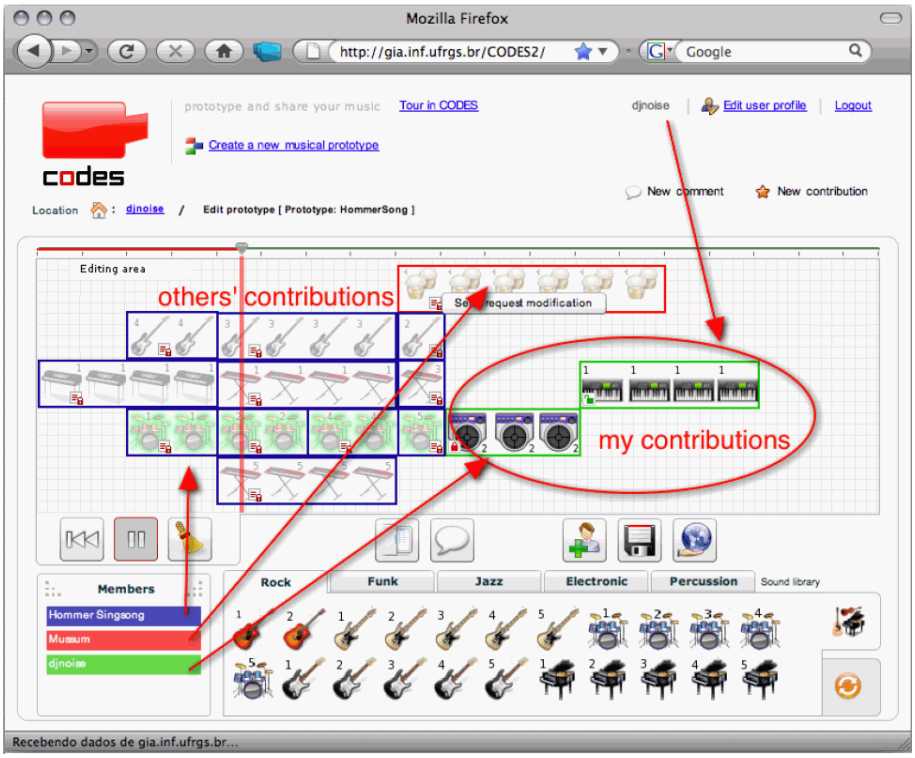


Figure 8.3 CODES : interface collaborative de création d'un morceau de musique.
Au centre : portée musicale avec la contribution de chaque membre, en bas à gauche : participants à la composition, en bas : séquences musicales disponibles.

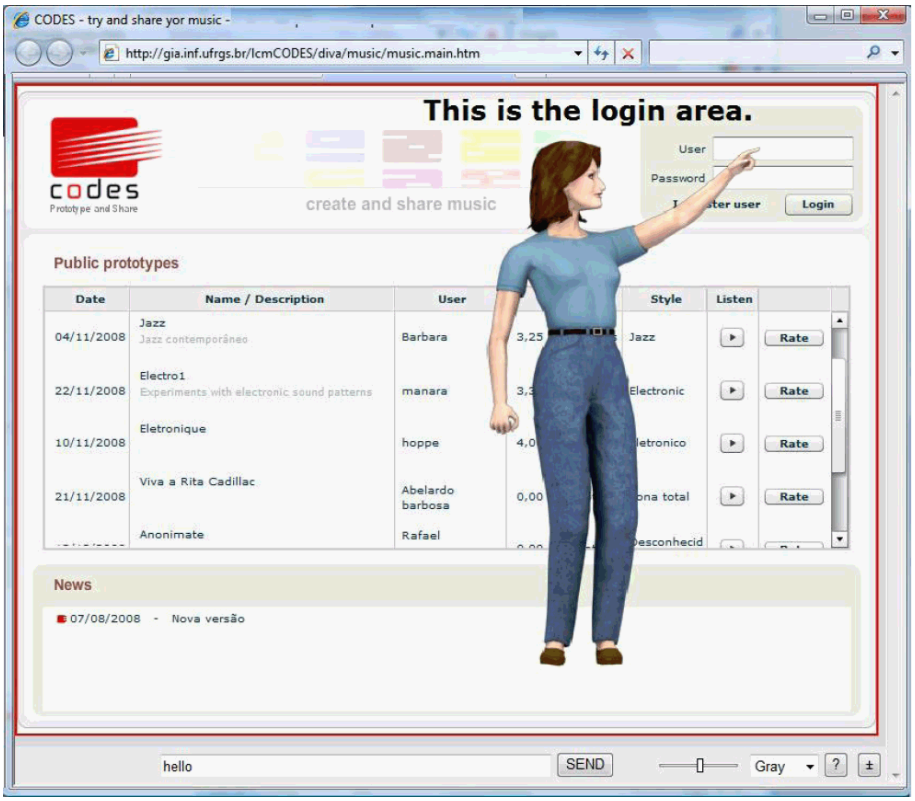


Figure 8.4 CODIVA : intégration d'un agent DIVA à CODES, ici en surimpression de la page d'accueil de l'application

Au début de l'expérience, on explique au sujet qu'il va devoir réaliser cinq tâches séquentielles (T_i doit être accomplie pour passer à T_{i+1}) de difficulté croissante au sein de l'environnement CODES (*cf.* Miletto *et al.* [2009] pour le détail de ces tâches). Durant la réalisation de ces tâches, il est demandé au sujet de penser à haute voix (principe du "think aloud" [Ummelen & Neutelings, 2000]) tandis que l'expérimentateur présent à ses côtés joue le rôle de "l'ami derrière l'épaule" de Capobianco & Carbonell [2001] pour répondre à ses éventuelles questions.

Analyse du corpus collecté. Chaque session (d'environ 10-15 minutes) a été entièrement enregistrée puis les requêtes du sujet ont été retranscrites sous forme textuelle, en remplaçant simplement les éventuels pronoms et indexicaux accompagnant un geste déictique (*e.g.* "qu'est-ce que c'est *ça* ?") par le nom de l'élément pointé par le sujet à l'écran (*e.g.* "qu'est-ce que c'est *le balai* ?"). 115 requêtes ont été ainsi collectées en anglais, portugais ou espagnol et retraduites en anglais pour la phase d'annotation (un extrait est donné dans le tableau 8.6).

N°	Requêtes (avant traitement des anaphores)	Anaphore	Transcription manuelle
1	Can I put *it* back to the library ?	sound pattern	CANI(Move(SP, SL))
2	Can I try all the styles ?	-	CANI(Try(MUSIC))
3	How can I stop *it* ?	sound pattern	HOWTO(Stop(SP))
4	How one can put the sound pattern *there* ?	editing area	HOWTO(Put(SP, EA))
5	How can I take off the broom ?	-	HOWTO(Quit(Broom))
6	How can I deactivate the broom ?	-	HOWTO(Quit(Broom))
7	How can I play the whole sequence ?	-	HOWTO(Play(MP))
8	How can I come *back* ?	editing area	HOWTO(Comeback(EA))
9	What do you mean with 3 sound patterns ?	-	ASK(Meaning(SP))
10	Should I listen to *all of them* ?	library	SHOULDI(Listen(SL))
11	Where is the editing area ?	-	ASK(Location(EA))
12	Are the musical styles : rock, funk, jazz ?	-	CHECK(Listof(MUSIC))

Tableau 8.6 Exemples de requêtes recueillies au sujet de CODES et retranscrites manuellement en pseudo-*DAFT*

D'un point de vue quantitatif, sans entrer dans les détails, nous pouvons faire trois remarques générales quant aux questions des sujets :

- L'assistance ne nécessite pas de traitement du dialogue : comme nous en avons déjà discuté dans le cas du corpus *Daft* (*cf.* section 3.1.3.4), lorsqu'une réponse satisfaisante lui est apportée (ce qui était le cas ici, l'expérimentateur présent derrière l'épaule connaissant bien l'application), un tour de dialogue suffit et les questions de l'utilisateur novice ne sont pas liées les unes aux autres.
- Il y a une grande variabilité dans la loquacité des sujets (de 4 à 17 questions - *cf.* figure 8.5 de gauche), sans que l'on puisse expliquer ici s'il s'agit d'une différence liée à leur personnalité propre ou à une exposition variable aux applications Web (bien qu'ils se soient tous auto-évalués comme novices en informatique, ils ont déjà eu accès à Internet).
- Bien que les tâches soient de complexité croissante, le nombre de questions pour chaque tâche va décroissant (*cf.* figure 8.5 de droite). La pertinence des réponses fournies n'étant

ici pas en doute (elles permettaient toujours l'accomplissement de la tâche), on peut penser que l'utilisateur se familiarise avec l'environnement et que les tâches prenant place dans le même contexte nécessitent donc moins de questions. Ainsi, il semble que c'est lorsque l'utilisateur entre en contact avec l'application que son besoin d'assistance est le plus fort, même s'il cherche par la suite à accomplir des tâches plus difficiles.

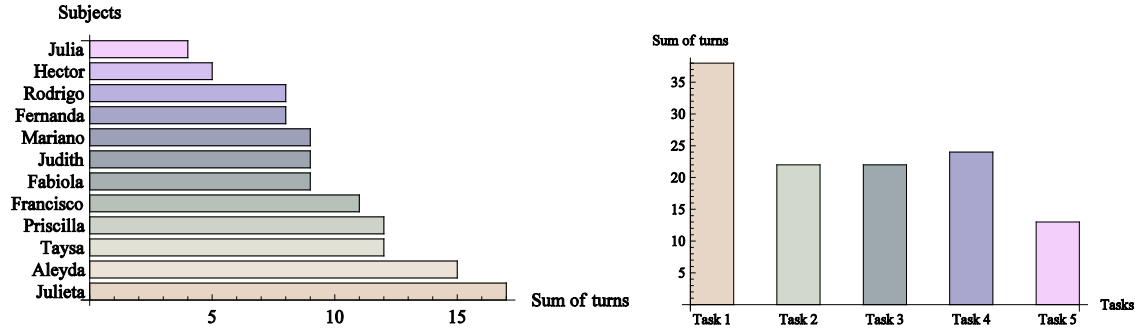


Figure 8.5 Nombre de questions posées par sujet (figure de gauche) et par tâche (figure de droite)

Identification du vocabulaire et des structures linguistiques. Le corpus étant de taille modeste, nous avons pu l'annoter entièrement selon une version simplifiée de la syntaxe des requêtes DAFT (à la manière de ce qui a été fait en 5.1), c'est-à-dire avec des requêtes de la forme : *Modalité[Prédicat[Références]]*.

Nous avons ainsi identifié, par ordre de fréquence décroissante :

- 6 modalités : HOWTO (≡ `ASK[WAY[]]`), CHECK (≡ `CHECK[]`), SHOULDI (≡ `ASK[OBLIGATION[]]` et `HELP[]`), ASK (≡ `ASK[]`), CANI (≡ `ASK[POSSIBILITY[]]` ou `ASK[AUTHORIZATION[]]`) et WHY (≡ `ASK[REASON[]]`).
- 35 prédicats (verbes mentaux ou verbes d'action) : Meaning, Listen, Choose, Click, Record, Find, Not, Change, Go, Stop, Delete, Quit, Move, Location, Available, Possibility, Put, Play, Significance, Obligation, Effect, Undo, Listof, Equals, Difference, Do, Try, Angry, Repeat, Use, Like, Correct, Found, Say, Necessity.
- 11 classes de références : SoundPattern (47), EditingArea (18), SoundLibrary (18), MusicInstrument (11), GUIElement (10), Broom (9), Number (6), MusicalPrototype (3), ManualElement (1), MusicStyle (1), Alignement (1).

On constate donc que la dépendance au domaine est proportionnelle à la profondeur de l'élément dans la requête, puisque si les 6 modalités ont un équivalent évident en *DAFT* (donné plus haut entre parenthèses) tout comme une partie des prédicats, les références sont quant à elles en grande majorité spécifiques à l'application (dans cette liste, seuls des éléments de la classe GUIElement et ManualElement étaient présents dans les clés sémantiques $\mathbb{K}_G$). En outre, on retrouve une distribution classique des éléments selon une loi de Zipf.

Par ailleurs, si l'on se concentre sur les 6 modalités employées, il est intéressant de noter là aussi une variabilité importante selon les sujets. Ainsi, on peut distinguer d'un côté des sujets, que l'on peut qualifier d'"hésitants", qui utilisent l'agent comme un moyen de se rassurer dans

les choix qu'ils font (CHECK, SHOULDI, CANI), et de l'autre, ceux qu'on peut qualifier de "dominants", qui considèrent l'agent comme un simple service d'information (HOWTO, ASK, WHY), comme le montre la répartition des sujets sur la figure 8.6. Cette information pourrait être utile dans un objectif d'adaptation de l'assistance fournie à différents profils d'utilisateurs.

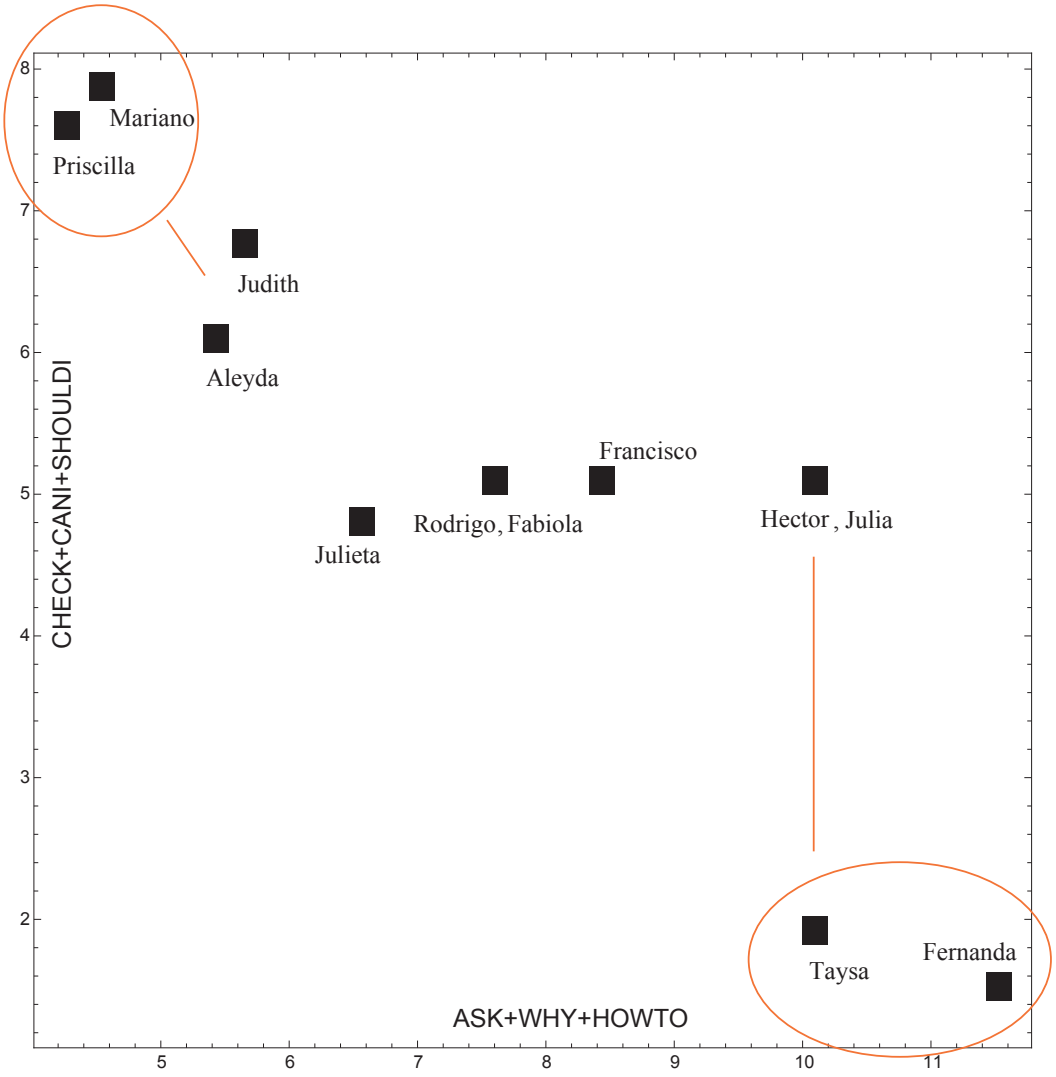


Figure 8.6 Distribution des sujets en fonction des modalités employées dans leurs requêtes.
En haut à gauche : les sujets "dominants", en bas à droite : les sujets "hésitants".

8.2 Vers des agents rationnels et comportementaux

Dans le cadre du développement du module de traitement placé à la suite de la chaîne d'analyse en langue naturelle (*cf.* figure 1), à savoir l'agent rationnel ($\mathtt{A}_R$), nous nous sommes intéressés aux contraintes pesant sur celui-ci et à la manière de définir une architecture d'agent permettant de les prendre en compte. Nous proposons donc dans cette section une synthèse des pistes explorées dans les publications [Bouchet & Sansonnet, 2009a], [Bouchet & Sansonnet, 2009c] et [Bouchet & Sansonnet, 2009b].

8.2.1 Besoin d'un agent rationnel et cognitif

8.2.1.1 De l'intérêt d'un modèle psychologique

L'expérience décrite dans la section 8.1.2 suggère que les utilisateurs attendent de l'agent un comportement qui n'est pas seulement rationnel mais qui exprime aussi des traits de comportement "typiquement humains". De plus, il apparaît qu'une fois que l'on a obtenu en sortie de `DIG` la représentation sémantiquement correcte d'une requête exprimée en *DAFT*, un raisonnement purement rationnel ne permet pas forcément de produire une réaction unique et non ambigüe.

Considérons par exemple une requête d'assistance directe assez courante : "Comment faire pour quitter ?". D'un point de vue purement rationnel, l'agent pourrait répondre en détaillant la procédure et en pointant éventuellement vers le bouton permettant cette action. Pourtant, ce choix n'est pas "neutre" du point de vue de la relation psychologique entre l'utilisateur et l'agent. Et dans la mesure où Reeves & Nass [1996] ont montré la tendance naturelle des humains à l'anthropomorphisme, c'est-à-dire l'attribution de traits psychologiques à des objets technologiques en général, il est probable que les mêmes phénomènes en œuvre derrière le "Persona Effect" [Lester *et al.* , 1997] ne font que renforcer cette tendance. Une réponse de l'agent étant forcément interprétée par l'utilisateur qui la reçoit, on voit que même si l'on vise à une certaine neutralité de ton dans le cadre d'un ACA, il est indispensable de connaître les attentes de l'utilisateur pour adapter la réponse formulée et ainsi maîtriser la manière dont celle-ci sera perçue.

Ainsi, si l'agent a été amical jusque là, l'utilisateur peut s'attendre à ce que l'agent cherche à le retenir dans cette action qui va également mettre fin à leur interaction, ou au moins à comprendre ses raisons (est-ce par dépit de n'avoir pu accomplir ce qu'il voulait ?). La réponse "neutre" sera alors perçue comme "froide et distante" par l'utilisateur. Il faut donc que les réactions de l'agent soient constantes : si rien ne change, l'agent ne doit pas changer sa manière de répondre. Dans l'étude des émotions, la constance ("consistency") a d'ailleurs été soulignée comme étant un facteur crucial de crédibilité par Ortony [2003].

Inversement, si l'utilisateur a traité l'agent de manière dégradante (et plus ou moins impolie) à de nombreuses reprises, l'utilisateur peut s'attendre à une certaine défiance à son égard, et dans ce contexte, une réponse "neutre" (bien que correcte et utile) sera perçue au mieux comme "servile", au pire comme "stupide" (si l'utilisateur estime que c'est parce que l'agent n'a pas compris les affronts passés). Il faut donc également une adaptation des réactions : lorsque la situation change, l'agent doit prendre en compte dans sa manière de répondre cette évolution qui fait sens psychologiquement.
On voit donc que si la prise en compte de plusieurs tours de dialogue n'était pas indispensable pour répondre aux questions d'assistance d'un point de vue rationnel (*cf.* section 3.1.3.4), il n'en demeure pas moins que interactions passées affectent les attentes ultérieures de l'utilisateur au niveau du comportement de l'agent.

Pour répondre à cette double exigence, il est indispensable de doter l'agent rationnel d'un modèle psychologique permettant d'adapter ses réactions en fonction d'un état psychologique (humeur ou "mood") qui évolue au fil du temps en fonction des interactions avec l'utilisateur, tout en conservant un comportement global consistant grâce à des traits de personnalité statiques[11].

8.2.1.2 De l'intérêt de contraintes cognitives

On a vu que l'inclusion d'un modèle psychologique au sein de l'agent rationnel est en mesure d'apporter un degré de réalisme supplémentaire à l'ACA. Son intégration à la prise de décision dans un système classique à base d'heuristiques se fait typiquement, soit par l'ajout dans les conditions de règles de variables liées au modèle psychologique, soit par l'ajout d'un paquet de règles supplémentaires indépendantes à appliquer avant ou après les règles rationnelles. Dans les deux cas, cette implémentation manque de réalisme, ce qui peut être problématique pour plusieurs raisons :

1. même lorsqu'une décision prend en compte des variables de nature psychologique, elle est toujours calculée intentionnellement, c'est-à-dire que l'agent est théoriquement capable d'exhiber, si on lui demande, les règles qu'il a appliquées. Par conséquent, il est également en mesure de déterminer son comportement rationnel sans ces variables psychologiques ;
2. si nous considérons un agent développé de manière collaborative (chaque concepteur définit ses propres heuristiques et celles-ci agissent ensemble au sein de l'agent rationnel), chaque règle peut potentiellement être cachée ou altérée par d'autres règles de priorité supérieure[12]. Ceci est problématique lorsque des comportements liés à des émotions fortes sont produits. Par exemple, une heuristique générant une réaction de type "énervement" pourrait ne pas être exprimée à cause d'une autre heuristique stipulant que l'agent ne doit pas exprimer de comportement extrême – un contrôle de soi qui n'est pas toujours possible pour un humain soumis à de fortes émotions ;
3. si l'on considère un agent optimisant son comportement en fonction de ses interactions avec des utilisateurs humains (ce que Sloman [2000] nomme self-monitoring), il pourrait être amené à se débarrasser d'heuristiques qui augmentent son réalisme mais diminuent son efficacité (par exemple ne pas répondre à une question si l'utilisateur l'a précédemment insulté). En effet, alors que l'efficacité peut se mesurer objectivement en comparant les résultats prédits par une heuristique à l'état du monde obtenu en l'appliquant, il semble plus difficile d'imaginer une manière pour l'agent d'évaluer son réalisme qui est une notion beaucoup plus subjective.

[11] Bien entendu, les traits peuvent évoluer sur le long terme. Ici nous entendons donc par statique le fait qu'un trait demeure constant durant une session dialogique.

[12] Le problème se pose aussi lorsque le concepteur d'heuristiques est unique mais que leur nombre est élevé, ou qu'elles sont définies sur une période étendue permettant de ne pas avoir toutes les règles déjà existantes en tête.

Pour toutes ces raisons, il nous semble important de pouvoir implémenter certaines contraintes liées au modèle psychologique de manière suffisamment profonde pour rendre l'agent incapable non seulement d'y avoir accès, mais aussi de détecter leur application. Cette notion, contre-intuitive par rapport aux systèmes à base de règles traditionnels, correspond à ce que nous nommerons par la suite des "biais cognitifs" (ou juste "biais"). Un biais cognitif est donc une règle de transformation classique dans sa définition, mais appliquée :

1. en dehors du moteur de règles principal de l'agent rationnel ;
2. aussi bien sur les messages provenant de `DIG` et allant vers l'agent animé chargé d'exprimer la réponse, que sur les interactions de l'agent avec le monde extérieur et avec sa propre mémoire interne.

Les biais cognitifs peuvent donc être comparés à des filtres sur les canaux de communication internes de l'agent rationnel, ce qui garantit que :

1. leur application se fasse toujours en premier ou en dernier,
2. il n'y ait plus de traces du message originel (avant modification),
3. les règles qu'ils appliquent ne soient pas altérables par d'autres heuristiques de l'agent rationnel.

8.2.1.3 Travaux liés

Dans le domaine des systèmes multi-agents modélisant des communautés humaines, l'idée d'agents cognitifs (utilisant des théories cognitives pour modéliser les capacités de raisonnement de l'agent) a déjà été explorée. Par exemple CoJACK [Norling & Ritter, 2004; Evertsz *et al.* , 2008] introduit une couche supplémentaire à l'environnement de création d'agents BDI JACK, permettant de prendre en compte des paramètres simulant les contraintes physiologiques humaines comme le temps nécessaire pour la cognition, les limites de la mémoire de travail (*e.g.* "perdre une connaissance" si son niveau d'activation est faible ou "oublier l'étape suivante" dans le cadre d'une procédure), un degré d'attention limité ou l'utilisation d'éléments susceptibles d'altérer les processus cognitifs.
Plusieurs études se sont aussi intéressées à l'ajout d'émotions en modifiant directement l'architecture BDI, par exemple pour prendre en compte la peur, l'anxiété ou la confiance en soi en utilisant des paramètres supplémentaires comme les désirs fondamentaux, les capacités ou les ressources [Pereira *et al.* , 2008]. D'autres comme Pauchet *et al.* [2007] cherchent à remplacer la librairie de plans statiques utilisée en BDI par un module de planification dynamique prenant en compte la personnalité définie pour l'agent. Casali *et al.* [2004] considère de son côté le recours à des logiques alternatives comme celle de Łukasiewicz pour la modélisation des croyances, désirs et intentions. Enfin, Dastani & van der Torre [2002] a montré que l'ordre même dans lequel des heuristiques étaient appliquées pouvait avoir un impact significatif sur la façon dont la personnalité de l'agent est perçue, permettant de caractériser différentes personnalités d'agents avec des traits de caractères comme "stable", "social" ou "égoïste".

8.2.2 Architecture de l'agent rationnel et psychologique

Afin d'exhiber ce que pourraient être les paramètres de personnalité pertinents pour un agent assistant, il est indispensable de procéder à des expérimentations avec diverses classes d'agents. Ceci débouche sur la nécessité de développer une architecture capable de supporter cette variabilité.

8.2.2.1 Élements du modèle

On distingue l'agent rationnel et cognitif $\mathtt{A}_R$ du reste du monde, $\mathbb{W}$, constitué par l'utilisateur et l'application assistée et avec lequel l'agent peut interagir (on suppose que le contenu du monde extérieur peut évoluer de manière indépendante de l'agent).
L'agent rationnel et cognitif $\mathtt{A}_R$ est quant à lui formellement défini comme étant un triplet $\mathtt{A}_R = \langle \mathscr{E}, \mathbb{M}, \Psi \rangle$, dans lequel :

- $\mathscr{E}$ est le moteur ("engine") en charge de l'exécution des réactions liées aux heuristiques ;
- $\mathbb{M}$ représente la mémoire de l'agent stockant toutes les informations que possède l'agent ;
- Ψ représente le modèle psychologique des états mentaux de l'agent.

L'organisation générale de ces éléments entre eux est synthétisée sur la figure 8.7. Le fonctionnement dynamique est décrit dans la section 8.2.2.5.

Mémoire ($\mathbb{M}$). La mémoire de l'agent peut se subdiviser en trois sous-ensembles, selon la nature des informations contenues :

- $\mathbb{M}_\mathrm{S}$, la mémoire sémantique de l'agent, qui contient le modèle partiel de $\mathbb{W}$ que l'agent a pu acquérir de manière directe à l'aide d'observateurs, et éventuellement complété par des informations produites par inférence.
- $\mathbb{M}_\mathrm{E}$, la mémoire épisodique de l'agent, dont la différence avec MS provient du fait qu'elle se focalise sur les informations concernant l'agent lui-même, représentant ainsi sa mémoire autobiographique au sens de Tulving [1983] (*i.e.* "ce qui lui est arrivé"). Il s'agit donc d'un historique des messages échangés avec le monde extérieur (utilisateur et application). Dans la mesure où l'on a vu que l'historique des interactions passées n'était pas utile pour la réaction de l'agent, en pratique cette mémoire n'est pas utilisée activement par les heuristiques. En effet, une interaction peut avoir un impact durable sur les interactions suivantes (via les variables psychologiques), mais il n'est pas nécessaire de reconsidérer les interactions passées.
- $\mathbb{M}_\mathrm{P}$, la mémoire procédurale de l'agent, composée de l'ensemble des heuristiques de l'agent, c'est-à-dire les réactions possibles lorsque certaines conditions sont remplies.

Modèle psychologique (Ψ). Nous avons décrit dans la section 8.2.1.1 l'intérêt du modèle psychologique de l'agent et montré qu'il devait contenir des attributs à valeur statique ainsi que d'autres à valeur dynamique. De plus, on peut distinguer le fait que l'agent peut avoir :

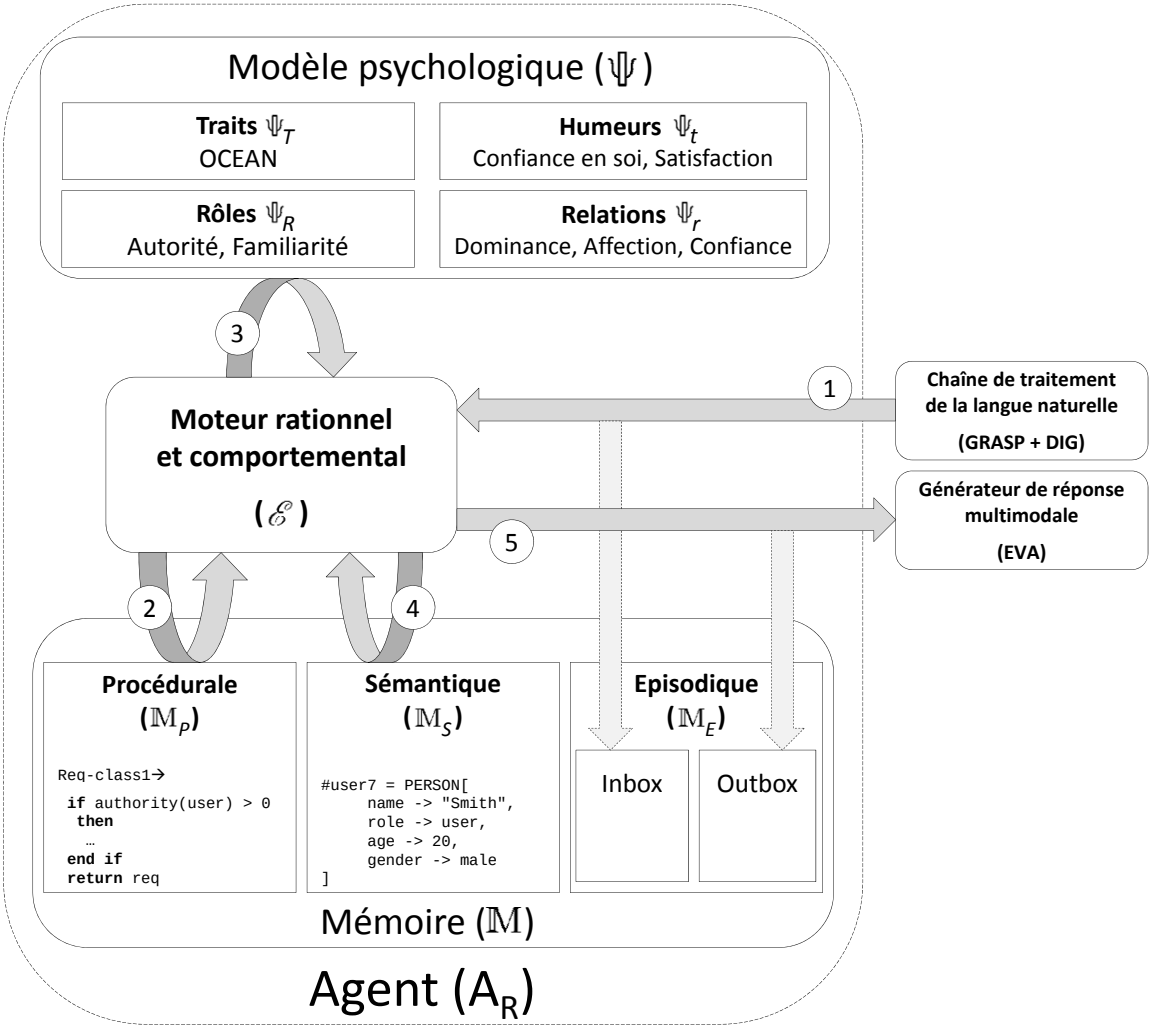


Figure 8.7 Architecture générale proposée pour l'agent rationnel et psychologique – les flèches grises indiquent l'ordre de traitement typique d'une requête formelle (en *DAFT*).

– un état mental intrinsèque, lié à sa personnalité profonde (*e.g.* il est de nature amicale, ce qui affecte toutes ses interactions) ;
– un état mental extrinsèque, lié à l'utilisateur avec lequel il est en train d'interagir (*e.g.* l'utilisateur vient de l'insulter, ce qui tend à rendre l'agent inamical envers lui dans les interactions suivantes).

Nous pouvons alors définir quatre classes d'états mentaux en fonction de leur dynamique et de leur arité fonctionnelle, tels que résumés dans le tableau 8.7.

État mental	Unaire	Binaire
Statique	Traits Ψ_{T} (O, C, E, A, N)	Rôles Ψ_{R} (A, F)
Dynamique	Humeurs Ψ_{t} (E, B, S, C)	Relations Ψ_{r} (D, A, T)

Tableau 8.7 Les quatre types d'états mentaux d'un agent, selon leur dynamique et leur arité

Chaque classe d'état mental se décompose en plusieurs paramètres (identifiés dans le tableau 8.7 par leur initiale) et que nous allons maintenant détailler.

Les traits ($\Psi_{\mathbf{T}}$) correspondent aux attributs classiques de personnalité et peuvent être considérés comme stables au cours d'une session. Nous reprenons ici le modèle des "Big Five" [Goldberg, 1981] autour duquel s'est établi un certain consensus en psychologie, et considérons donc les cinq traits suivants :

– **O**uverture à l'expérience : représente l'imagination, la curiosité et le goût de l'aventure ;
– Caractère **C**onsciencieux : représente l'auto-discipline et la tendance au respect des obligations ;
– **E**xtraversion : représente l'énergie, la force des émotions positives et le goût des autres ;
– **A**gréabilité : représente la tendance à se montrer compatissant, affectueux et coopératif ;
– **N**euroticisme : représente la tendance à ressentir facilement des émotions négatives (colère, anxiété, vulnérabilité, *etc.*).

Les humeurs ($\Psi_{\mathbf{t}}$) ("moods" en anglais) représentent les facteurs de personnalité qui varient avec le temps, modifiés par des heuristiques et des biais cognitifs. L'état humoral d'un agent est divisé en deux parties :

1. Paramètres physiques : qui modélisent l'agent en tant qu'entité physique du monde (à la manière de la métaphore des attributs physiques employées dans les jeux vidéo).
2. Paramètres épistémiques : qui modélisent l'agent en tant qu'entité capable d'évaluation de ses actions rationnelles.

Nous pouvons alors considérer les quatre humeurs suivantes :

– **É**nergie physique : représente la force physique de l'agent, au sens large ;
– **B**onheur physiologique : représente le bien-être physique de l'agent, selon sa situation physique ;

- **C**onfiance en soi ("confidence") : représente l'assurance de l'agent au sujet de ses propres capacités ;
- **S**atisfaction intellectuelle : représente le bien-être mental de l'agent, selon l'analyse qu'il fait de sa situation intentionnelle (*i.e.* vis-à-vis de ses buts, désirs et intentions).

Dans le contexte d'un ACA, celui-ci n'ayant pas d'incarnation physique dans le monde (comme par exemple un robot), on ne considérera ici que les paramètres épistémiques (confiance en soi et satisfaction).

Les rôles ($\Psi_{\mathbf{R}}$) représentent des relations statiques à caractère institutionnel, entre l'agent et les autres entités du monde (ici, les différents utilisateurs possibles). On peut définir deux grandes catégories :

- **A**utorité : elle caractérise le droit que l'agent croit posséder d'être directif vis-à-vis de son interlocuteur, ou réciproquement de ne pas accepter (facilement) des directives en provenance de celui-ci. Cette relation est souvent antisymétrique :
  `Authority(X,Y) = -Authority(Y,X)`
- **F**amiliarité : elle caractérise le droit que l'agent croit posséder de se comporter de manière informelle vis-à-vis de son interlocuteur. Cette relation est souvent symétrique :
  `Familiarity(X,Y) = Familiarity(Y,X)`

Dans le cadre d'un ACA, l'autorité sera a priori clairement en faveur de l'utilisateur.

Les relations ($\Psi_{\mathbf{r}}$) modélisent les relations dynamiques entre l'agent et les autres entités (typiquement les utilisateurs). Nous en distinguons au moins trois sortes :

- **D**ominance : exprime que le sentiment de puissance que ressent l'agent par rapport à son interlocuteur. Cette relation est souvent antisymétrique :
  `Dominance(X,Y) = -Dominance(Y,X)` ;
- **A**ffection : exprime une attirance et une tendance à agir amicalement de l'agent envers l'interlocuteur. Cette relation n'est pas nécessairement symétrique ;
- Confiance ("**T**rust") : exprime que l'agent fait confiance à l'interlocuteur. Cette relation n'est pas nécessairement symétrique.

Valeurs des paramètres On utilisera la notation abrégée $\Psi_x.Y$ pour désigner le paramètre d'initiale Y dans la classe x, par exemple Ψ_{T}.E pour l'extraversion et Ψ_{r}.A(a,u) pour l'affection de l'agent envers l'utilisateur.
À chaque paramètre est associée une valeur numérique dans l'intervalle réel $[-1, 1]$, tel que 0 définit "l'état mental neutre", 1 correspond à l'expression maximale du paramètre, et -1 l'expression de l'opposé du paramètre décrit. Ainsi, -1 en extraversion signifie une forte introversion et -1 en affection pour un utilisateur représente une haine profonde à l'égard de cet utilisateur.

8.2.2.2 Représentation des informations (*MDL*)

Les informations contenues dans $\mathbb{W}$, $\mathbb{M}_S$ et Ψ sont représentées dans un langage de description *Model Description Language (MDL)* sous la forme de triplets (à la manière des triplets Resource Description Framework (RDF) [Lassila & Swick, 1999]) associés à un identifiant tels que :

$$\#id = H\left[\bigcup_i a_i \rightarrow v_i\right]$$

où $\#id$ correspond à l'identifiant unique de la référence, H est l'en-tête du triplet, a_i un attribut parmi la liste des attributs associés à H et v_i une valeur dans le domaine de validité associé à a_i. v_i peut donc être :

- une valeur terminale (chaîne de caractères, nombre, valeur d'un ensemble...),
- un triplet au même format,
- un identifiant correspondant à un triplet existant.

Exemples de triplets MDL de $\mathbb{W}$ ou $\mathbb{M}_S$:

```
#user7 = PERSON[
      name          -> "Jean-Claude",
      role          -> user,
      age           -> 40,
      gender        -> male,
      ...
]
```

Exemples de triplets MDL de Ψ :

```
#traits = traits[
      openness            -> 0,
      conscientiousness   -> 0.5,
      ...
]

#role1 = role[
      towards             -> #user7,
      authority           -> -0.8,
      familiarity         -> 0.2
]
```

8.2.2.3 Communication entre les éléments du modèle (*MQL*)

Les interactions entre $\mathbb{W}$ et $\mathtt{A}_R$, mais aussi entre la mémoire $\mathbb{M}$ et le modèle psychologique Ψ (au sein de $\mathtt{A}_R$) sont gérées par le moteur $\mathscr{E}$ par l'envoi de messages exprimés dans le langage d'interface avec le modèle ou *Model Query Language (MQL)*.

Ce langage *MQL* est minimal[13] et constitué de cinq types de messages seulement :

- `INFORM[`acteur, triplet`]` :transmet un triplet à un acteur (élément de $\mathbb{W}$, $\mathbb{M}$ ou Ψ) pour que celui-ci soit inséré ou modifié (si l'attribut existe déjà). Aucun message n'est attendu en retour.
- `GET[`acteur, triplet`]` : requiert la valeur d'un triplet d'un acteur, et attend en retour un message de type `INFORM` ou `FAIL`.
- `CHECK[`acteur, triplet`]` : demande à un acteur si la valeur d'un attribut d'un triplet de sa base de connaissances est bien celle passée en paramètre. On attend en retour un message de type `OK` si c'est le cas, `INFORM` avec la bonne valeur pour le triplet si l'attribut existe mais que sa valeur est différente ou `FAIL` si l'attribut n'existe pas.
- `FAIL[`triplet`]` : en réponse à un `GET` ou à un `CHECK`, et contenant le triplet transmis dans le message d'origine.
- `OK[`triplet`]` : en réponse à un `CHECK`, et contenant le triplet transmis dans le message d'origine.

8.2.2.4 Heuristiques (*HDL*)

Les heuristiques de l'agent sont stockées dans la mémoire procédurale $\mathbb{M}_\mathrm{P}$ et définies dans un langage de description spécifique appelé *Heuristic Description Language (HDL)*. Une heuristique se compose de deux parties :

- une en-tête, définissant la classe de requêtes *DAFT* pour laquelle l'heuristique est adaptée. Elle est formée d'un ensemble de schèmes structurés, pour permettre un filtrage par motif. Par exemple `ASK[about=POSSIBILITY[todo=A_[__]]]` pour les questions sur la possibilité d'exécution d'une action[14]
- un corps, constitué d'un ensemble d'instructions conditionnelles dont l'exécution dépend de la valeur de triplets *MDL* de $\mathbb{M}_\mathrm{S}$, $\mathbb{W}$ ou Ψ.

Formellement, une heuristique est donc de la forme :

$$\mathrm{H} : \mathrm{id}[\text{pattern } \mathit{DAFT}] \mapsto \{\text{ScriptGardé}_1, \ldots, \text{ScriptGardé}_n\}$$

où :

$$\begin{aligned}
\text{ScriptGardé} &\equiv \{\text{Garde}_1 \rightarrow \text{Script}_1, \ldots, \text{Garde}_n \rightarrow \text{Script}_n\} \\
\text{Garde}_i &\equiv \text{Expression logique} \mid \text{Message } \mathit{MQL} \mid \emptyset \qquad (\emptyset = \text{toujours vrai}) \\
\text{Script}_i &\equiv \text{Instruction} \mid \{\text{Instruction}_1, \ldots, \text{Instruction}_n\} \\
\text{Instruction}_i &\equiv \text{ScriptGardé} \mid \text{Message } \mathit{MQL} \text{ interne} \mid \text{Requête } \mathit{DAFT} \text{ à l'utilisateur}
\end{aligned}$$

Notons qu'on considère que le générateur de requêtes en langue naturelle (qui n'est pas traité dans cette thèse) prend également en entrée des requêtes données dans les heuristiques, elles aussi exprimées dans le langage formel *DAFT*.

[13] Bien que les différents éléments cohabitant au sein de l'agent rationnel se rapprochent par certains égards d'un petit système multi-agents, il n'est pas utile ici de définir un véritable langage de communication de type ACL-FIPA [O'Brien & Nicol, 1998].

[14] Suivant le principe de notation des motifs en Mathematica, $x\_$ permet de faire référence à une entité d'après sa classe $c \in M, A, R, P$, tandis que _ correspond à n'importe quelle entité, et que ___ dénote une séquence (équivalent au symbole + dans le vocabulaire classique des expressions régulières).

Exemple d'heuristique rationnelle de gestion de demande de valeur d'un attribut :

H_{R1} : ask-agent-attribute[
ASK[about=R_1_[ppt*={__, P_1_[val=∅], __, ownedby[val=person[id=system]], __}]]
] ↦ {
∅ → GET[$\mathbb{M}_S$, {self, P_, ∅}]
INFORM[{self, P_, V_}]
↪ → LINK[ref=R_1, isa=concept[ppt*={val = V}]]
FAIL[{self, P_, V_}]
↪ → NEG[about=KNOWLEDGE[R_1]]
}

L'heuristique ci-dessus permet par exemple de déterminer la réaction à la question "Quel est ton âge ?", dont la représentation en *DAFT* produite par DIG est donnée par la figure 8.2. En supposant que la requête *DAFT* soit transformée en langue naturelle, on pourrait alors avoir comme réponse "Mon âge est 25" ou "Je ne connais pas mon âge".

8.2.2.5 Fonctionnement dynamique

Exécution typique. On considère que l'agent est seulement réactif : les heuristiques contenues dans $\mathbb{M}_P$ ne sont donc consultées par le moteur $\mathscr{E}$ que lorsqu'une requête est envoyée par DIG. Le fonctionnement typique, résumé sur la figure 8.7, est donc le suivant :

1. La chaîne de traitement de la langue naturelle composée de GRASP et DIG transmet une requête formelle exprimée en *DAFT* à l'agent rationnel $\mathtt{A}_R$. Elle est mémorisée dans la mémoire épisodique.
2. Le moteur $\mathscr{E}$ cherche dans la mémoire procédurale une ou plusieurs heuristiques associées à la requête entrante et les exécute.
3. Ces heuristiques permettent de définir des réactions différentes en fonction de l'état mental actuel, et il faut donc consulter Ψ. Elles nécessitent aussi généralement d'aller chercher des informations sur l'agent ou l'application dans $\mathbb{M}_S$ ou $\mathbb{W}$ pour répondre à la requête.
4. Éventuellement, la requête peut aussi entraîner une évolution du contenu de Ψ et $\mathbb{M}_S$ (modification de la valeur d'un triplet, ajout ou suppression de triplet(s)).
5. Finalement, au moins une réponse exprimée en *DAFT* est produite et transmise au générateur de réponse multimodale, qui est en charge de la génération de texte, de gestes et d'émotions appropriés.

Impact de la psychologie sur le fonctionnement dynamique. Notons que la politique même définissant le fonctionnement dynamique pourrait être dépendante de l'état mental de l'agent. Par exemple, lorsque l'on souhaite vérifier la valeur d'une propriété de l'application assistée avec une requête de type CHECK[...], ceci peut se faire soit en consultant la mémoire $\mathbb{M}_S$, soit en accédant directement à l'application dans $\mathbb{W}$. On peut donc envisager deux possibilités :

- Priorité au test dans $\mathbb{M}_S$: si l'agent a une valeur élevée associée à la confiance qu'il a en lui-même, il essaye de retrouver directement l'information désirée depuis $\mathbb{M}_S$. Il ne consulte alors $\mathbb{W}$ que s'il n'arrive pas à trouver une réponse dans sa propre mémoire ;
- Priorité au test dans $\mathbb{W}$: à l'inverse, si l'agent n'a pas confiance en lui, il ne se donnera pas la peine de vérifier dans sa propre mémoire et essayera directement de retrouver l'information depuis $\mathbb{W}$.

Le choix même des informations stockées dans $\mathbb{M}_S$ peut lui même dépendre encore une fois de la psychologie de l'agent. Ainsi, $\mathbb{M}_S$ peut être :

- Copie stricte de $\mathbb{W}$: pour un agent sérieux (*i.e.* ayant une valeur élevée associée au trait "conscientiousness"), la mémoire de l'agent se comporte exactement comme un cache et un historique de $\mathbb{W}$;
- Copie surchargée de $\mathbb{W}$: pour un agent prêt à prendre plus de risques, la mémoire de l'agent contient aussi le résultat des calculs qu'il est amené à effectuer préalablement en interne.

À partir de là, on constate qu'un agent ayant confiance en lui mais pas très sérieux peut manquer d'efficacité car il est amené à répondre directement à l'utilisateur, sur la base d'informations qui sont peut-être obsolètes. Cependant, en termes de réalisme, ce comportement émule assez bien celui d'un être humain possédant ce trait de personnalité (Conscientiousness $= -1$) et cette humeur (Confident $= 1$) ; et s'il peut sembler inacceptable à un utilisateur lui-même sérieux, il n'est pas sûr qu'un utilisateur ayant le même profil blâmerait l'agent pour cela.

8.2.3 Intégration des biais cognitifs à l'architecture

L'intérêt des biais cognitifs au sein d'un agent rationnel et psychologique ayant été discuté en section 8.2.1.2, nous nous proposons donc maintenant de considérer les moyens de les intégrer à l'architecture présentée sur la figure 8.7, tel qu'illustré par la figure 8.8.

8.2.3.1 Distinction biais-heuristique

Nous avons présenté en 8.2.2.3 la façon dont le moteur $\mathscr{E}$ communique avec $\mathbb{M}_S$, Ψ et $\mathbb{W}$ via des messages *MQL*. Ces messages peuvent donc être modifiés lorsqu'ils transitent à partir ou vers $\mathscr{E}$ par des filtres implémentant les biais cognitifs. Formellement, on peut donc définir un biais comme étant une transformation de message *MQL* réalisée à l'insu de $\mathscr{E}$. Un biais b s'appliquant sur le canal de communication allant d'un élément X à un élément Y sera représenté par $X \xrightarrow{b} Y$.
Un biais peut être représenté dans une syntaxe similaire à celle employée en *HDL* pour la description des heuristiques. Soulignons cependant que la différence fondamentale avec les heuristiques stockées dans $\mathbb{M}_P$ est l'impossibilité effective pour l'agent d'expliquer les biais

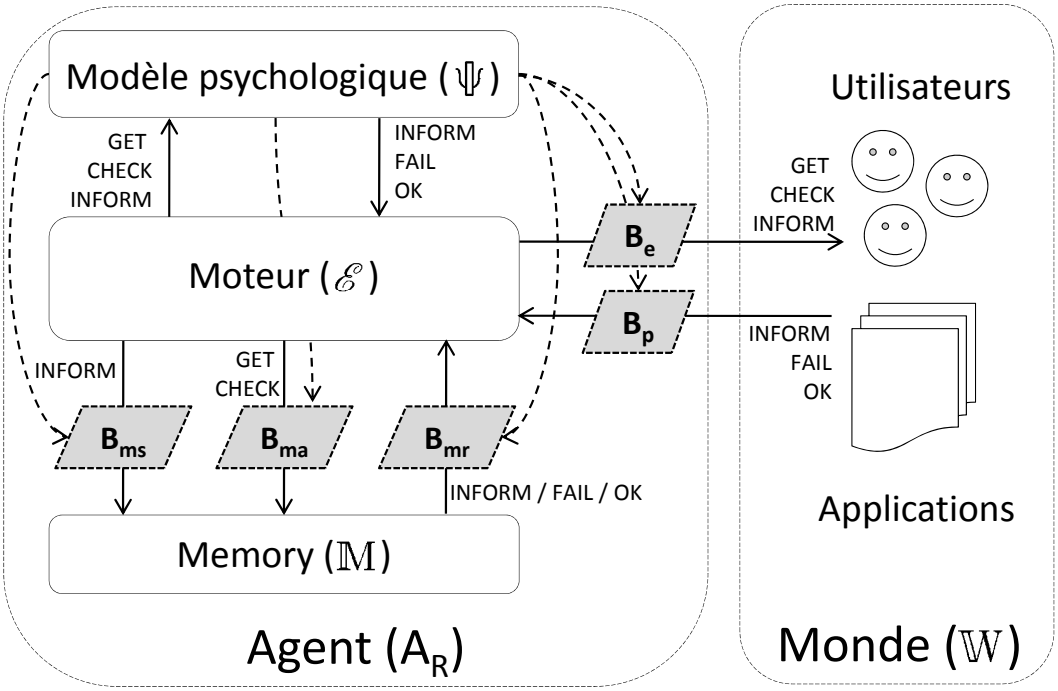


Figure 8.8 Architecture générale de l'agent rationnel et psychologique avec biais cognitifs – les flèches simples indiquent les messages *MQL* filtrés par chaque biais, les flèches pointillées symbolisent l'influence de Ψ sur les biais.

puisqu'il n'a pas connaissance de leur action. En outre, tandis qu'une heuristique donnée ne s'applique que sur les requêtes formelles *DAFT* correspondant à son en-tête, un biais s'applique à tout message (*MQL* ou *DAFT*) transmis sur le canal où il est présent. Enfin, alors que les heuristiques peuvent dépendre de paramètres psychologiques (de Ψ) et rationnels (de $\mathbb{M}_S$), un biais est purement psychologique, ne comportant dans sa garde que des conditions issues de Ψ. Le tableau 8.8 résume ces différences.

Critère	Heuristique	Biais cognitif
Objectif principal	Génération d'une requête *DAFT*	Modification d'un message *MQL* ou d'une requête *DAFT*
Portée	Une classe de requêtes *DAFT*	Toute requête entre deux éléments donnés de $\mathtt{A}_R$
Conditions d'application	Contenu de la requête *DAFT*, valeurs de $\mathbb{W}$, de $\mathbb{M}_S$ et/ou de Ψ	Ψ seulement
Introspection possible	Oui	Non

Tableau 8.8 Résumé des différences entres les règles appliquées par les biais cognitifs et les heuristiques

8.2.3.2 Catégories de biais

Les biais sont orientés, ce qui signifie que pour un couple (X, Y) d'éléments de l'architecture de $\mathtt{A}_R$, il est possible de définir deux biais : $X \xrightarrow{b} Y \neq Y \xrightarrow{b} X$. De plus, les biais sont

dépendants du type de message *MQL* transmis entre deux éléments. Par conséquent, si chacun des quatre éléments ($\mathscr{E}$, $\mathbb{M}$, Ψ et $\mathbb{W}$) pouvait communiquer avec chacun des trois autres, il y aurait six canaux bidirectionnels portant cinq types de messages (`INFORM`, `GET`, `CHECK`, `OK` et `FAIL`), ce qui engendrerait une combinatoire de $6 \times 2 \times 5 = 60$ biais différents. Cependant, la plupart d'entre eux ne sont pas pertinents : eux ne sont pas pertinents :

- Seul $\mathscr{E}$ est un élément actif, et donc :
  - $\mathscr{E}$ est le seul capable d'initier l'envoi de messages `INFORM`, `GET` et `CHECK` ;
  - $\mathscr{E}$ n'envoie jamais de `OK` ou de `FAIL` dans la mesure où ce sont exclusivement des messages de réponse ;
  - $\mathbb{W}$, $\mathbb{M}$ et Ψ ne communiquent pas entre eux, et ne peuvent envoyer que des messages de réponse `INFORM`, `OK` et `FAIL` à $\mathscr{E}$.
- Il est difficile d'imaginer des situations où l'agent ne serait pas capable de connaître exactement son propre état mental, c'est pourquoi nous ne considérerons pas l'existence de biais entre $\mathscr{E}$ et Ψ. En fait, nous supposerons que l'accès à Ψ est transparent à partir des heuristiques et des biais. Cela signifie que l'on peut utiliser les propriétés de Ψ directement dans la partie gauche d'une règle sans envoyer de message `GET` préalable. En effet, d'une part l'ensemble des propriétés de Ψ est connu et fini et il n'y a donc pas de `FAIL` possible en retour, et d'autre part le message de `INFORM` renvoyé en retour ne peut jamais être biaisé.

Suite à ces remarques, on voit qu'il ne reste que six canaux unidirectionnels ($\mathscr{E} \rightarrow \mathbb{M}$, $\mathscr{E} \rightarrow \Psi$, $\mathscr{E} \rightarrow \mathbb{W}$, $\mathbb{M} \rightarrow \mathscr{E}$, $\Psi \rightarrow \mathscr{E}$, $\mathbb{W} \rightarrow \mathscr{E}$) parmi lesquels seuls quatre peuvent être munis de biais, comme indiqué sur la figure 8.8. Sur chacun de ces canaux ne peuvent transiter que trois types de messages :

- `GET`, `CHECK` et `INFORM` pour $\mathscr{E} \rightarrow \mathbb{M}$ et $\mathscr{E} \rightarrow \mathbb{W}$;
- `OK`, `FAIL` et `INFORM` pour $\mathbb{M} \rightarrow \mathscr{E}$ et $\mathbb{W} \rightarrow \mathscr{E}$.

Dans la mesure où l'on souhaite simplement distinguer les requêtes "en lecture" (`GET` et `CHECK`) des requêtes "en écriture" (`INFORM`) dans le cas de l'accès à la mémoire, on a finalement 12 biais possibles, que l'on peut répartir selon cinq catégories :

- Biais Perceptifs ($\mathbb{W} \xrightarrow{\mathrm{B_p}} \mathscr{E}$) : ce sont des biais sur une requête provenant du monde $\mathbb{W}$, c'est-à-dire soit de l'utilisateur via une requête formelle *DAFT* transmise par `DIG`, soit de l'application assistée s'il s'agit de la réponse à un message de type `GET` ou `CHECK` envoyé précédemment par $\mathscr{E}$.
- Biais Expressifs ($\mathscr{E} \xrightarrow{\mathrm{B_e}} \mathbb{W}$) : ce sont des biais appliqués sur un message `INFORM`, `GET` ou `CHECK` envoyé vers le monde $\mathbb{W}$.
- Biais de Recherche de faits en mémoire ("memory retrieve" : $\mathbb{M} \xrightarrow{\mathrm{B_{mr}}} \mathscr{E}$) : ils s'appliquent aux messages `OK`, `FAIL` ou `INFORM` émanant de la mémoire (en réponse à un message `GET` ou `CHECK` envoyé précédemment).
- Biais d'accès en lecture mémoire ("memory access" : $\mathscr{E} \xrightarrow{\mathrm{B_{ma}}} \mathbb{M}$) : ils portent sur un message de type `GET` ou `CHECK` envoyé à la mémoire.

– Biais d'écriture mémoire ("memory store" : $\mathscr{E} \xrightarrow{\mathrm{B_{ms}}} \mathbb{M}$) : ils portent sur un message INFORM envoyé à la mémoire.

Des exemples pour chacune de ces catégories sont donnés dans le tableau 8.9.

8.2.3.3 Représentation des biais

Tout comme les heuristiques, les biais possèdent deux parties :

– une en-tête, définissant la catégorie du biais parmi les cinq décrites en 8.2.3.2, et éventuellement le type de messages *MQL* auquel il s'applique.
– un corps, formellement identique à celui d'une heuristique (en *HDL*), constitué d'un ensemble d'instructions conditionnelles mais dont l'exécution dépend uniquement de la valeur de triplets *MDL* de Ψ (*cf.* tableau 8.8).

Nous donnons ci-dessous une implémentation possible d'un biais.

Exemple de biais : le biais perceptif de victimisation :

H_{B1} : victimization[perceptive, {*DAFT*-request r}] $\mapsto$ {

$(\Psi_{\mathrm{T}}.\mathrm{N} > 0.3 \wedge \Psi_{\mathrm{T}}.\mathrm{N} < 0.7 \wedge \Psi_{\mathrm{t}}.\mathrm{S} \leq -0.5) \quad \rightarrow \quad$ r $\leftarrow$ DISAPPROVE[r]

$(\Psi_{\mathrm{T}}.\mathrm{N} \geq 0.7 \wedge \Psi_{\mathrm{t}}.\mathrm{S} \leq -0.5) \quad \rightarrow \quad$ r $\leftarrow$ BLAME[who=agent, about=r]

}

Le biais ci-dessus s'applique sur les requêtes *DAFT* envoyées par DIG à l'agent $\mathtt{A}_R$ dans les cas où l'agent est de type neurotique et insatisfait. On distingue deux cas selon le degré de neuroticisme de l'agent[a]. Dans le premier cas, il tend à interpréter toute requête comme une expression de mécontentement de la part de l'utilisateur. Dans le second cas, il considére en plus que l'utilisateur le juge responsable.

Notons que BLAME et DISAPPROVE ne sont pas des modalités listées dans le langage *DAFT*, et que leur interprétation concrète selon le contenu propositionnel de la requête serait à traiter par des heuristiques rationnelles.

[a] Celui-ci étant par définition statique, cela signifie que pour un agent donné, au plus une seule des deux règles pourra être appliquée. L'intérêt de définir plusieurs cas est donc d'avoir des biais génériques, pouvant être utilisés dans différents agents dont on ne connaît pas a priori la valeur de leurs traits de personnalité statiques.

8.2.4 Perspectives

L'utilisation d'une architecture où les décisions dépendent à la fois de paramètres subjectifs et objectifs fait que l'efficacité de l'aide apportée par un ACA devient fortement dépendante de ses traits de personnalité. Dans la mesure où les heuristiques prennent en compte l'état mental (dynamique) mais aussi la personnalité (statique) de l'agent, elles offrent une certaine généricité, et on peut imaginer disposer de plusieurs types d'agents, permettant de choisir le plus adapté à un utilisateur donné (Reeves & Nass [1996] considérant qu'un utilisateur préfère interagir avec un agent ayant les mêmes traits de personnalité que lui).

Cat.	Nom du biais	Messages concernés	Conditions d'application	Description
B_p	Victimisation	Requêtes *DAFT*	Ψ_T.N élevé $\wedge$ Ψ_t.S faible	Perception d'une charge négative, voire de reproche à son égard, dans les requêtes de l'utilisateur
B_p	Minimisation	Requêtes *DAFT*	Ψ_T.N faible $\wedge$ Ψ_T.A élevé $\wedge$ Ψ_t.S élevé	Sous-évaluation de la charge négative éventuellement présente dans les requêtes de l'utilisateur
B_e	Stress	Requêtes *DAFT*	Ψ_R.A(a,u) très élevé	Expression d'une nervosité accrue dans les réponses générées, que l'agent ne peut contrôler (indépendamment par exemple du fait qu'il ait ou non la réponse à la question posée par l'utilisateur, qui peut générer un stress additionnel)
B_e	Enjouement	Requêtes *DAFT*	Ψ_T.E élevé $\wedge$ Ψ_R.F(a,u) élevé $\wedge$ Ψ_t.S élevé	Expression visible de sa bonne humeur (de manière langagière ou non, ce choix relevant du générateur)
B_e	Morosité	Requêtes *DAFT*	Ψ_T.E élevé $\wedge$ Ψ_R.F(a,u) élevé $\wedge$ Ψ_t.S faible	Expression visible de sa mauvaise humeur
B_{mr}	Doute	Messages `INFORM`, `OK` et `FAIL`	Ψ_t.S faible $\wedge$ Ψ_t.C faible	L'agent manque de confiance en lui et va donc remettre plus facilement en question ou minimiser les informations issues de sa propre mémoire $\mathbb{M}_S$
B_{ma}	Mauvaise foi	Messages `GET` et `CHECK`	Ψ_t.S très faible $\wedge$ Ψ_R.A(a,u) très élevé	L'agent ne pouvant se rebeller ouvertement peut, à la manière d'un "acte manqué", omettre certaines informations lors de sa recherche en mémoire (*e.g.* pour lister "les petits boutons rouges", il ne recherchera que "les petits boutons") : pour lui, il aura donc accompli ce qui lui était demandé
B_{ms}	Oubli	Messages `INFORM`	Ψ_T.N faible $\wedge$ Ψ_t.S élevé	Une information négative (*e.g.* une critique de l'utilisateur à son égard) peut ne pas être mémorisée dans $\mathbb{M}_E$
B_{ms}	Désordonné	Messages `INFORM`	Ψ_t.C faible	L'agent peut oublier de stocker des éléments d'information dans $\mathbb{M}_S$ ou $\mathbb{M}_E$ de manière aléatoire

Tableau 8.9 Exemples de biais cognitifs dans chacune des cinq catégories

Les biais cognitifs permettent quant à eux d'émuler certaines contraintes des comportements humains en donnant une primauté aux états mentaux des agents par rapport au raisonnement rationnel. L'intérêt de cette approche, en particulier pour un ACA, reste cependant à évaluer dans une étude où des utilisateurs novices seraient placés face à trois classes d'agents :

S1 Un agent purement rationnel ;

S2 Un agent rationnel et psychologique ;

S3 Un agent rationnel et psychologique, incluant également des biais cognitifs.

Nous nous attendons à une amélioration, en termes de réalisme, en passant de S1 à S2 ainsi que de S2 à S3. Il pourrait aussi y avoir un léger accroissement de la compétence entre S1 et S2. Cependant il est probable que l'introduction des biais conduise à une baisse du degré de compétence perçu, ce qui pose la question du choix difficile entre compétence et réalisme.

Conclusion

Nous avons montré dans cette thèse que l'utilisation d'agents conversationnels pour répondre au problème ouvert que constitue l'assistance aux utilisateurs novices est une approche prometteuse pour passer outre le "paradoxe de la motivation". Elle permet en effet de combiner les bénéfices de l'usage d'une part de la langue naturelle, moyen d'expression spontanément utilisé par les utilisateurs en quête d'assistance, d'autre part d'un agent conversationnel animé, implémentant des comportements humains, et augmentant ainsi la crédibilité du système.

Dans un premier temps, nous avons justifié l'intérêt d'une approche fondée sur un corpus dédié, et décrit la manière dont celui que nous avons construit à cette fin, le corpus *Daft*, a lui-même été constitué en combinant une procédure expérimentale de recueil de données et une utilisation de plusieurs sources de thésaurus. Nous nous sommes assurés que ce corpus de 11 626 phrases était représentatif des requêtes posées par des utilisateurs en situation d'assistance, tant en termes de vocabulaire que de structures linguistiques employées, et couvrait suffisamment ce domaine pour pouvoir servir de fondement à la conception de la chaîne de traitement de la langue que nous avons construite.
En le contrastant par rapport à d'autres corpus similaires de dialogues menés dans le cadre de l'accomplissement d'une tâche, nous avons pu par la suite exhiber certaines de ses spécificités linguistiques. Nous avons aussi pu constater qu'en plus des requêtes d'assistance attendues, la méthodologie employée nous avait permis de recueillir des requêtes relevant d'une part de l'activité de contrôle de l'application assistée, et d'autre part de d'activités de type discussion, sans rapport avec la tâche elle-même et que l'on estime liées à la présence du personnage de l'agent animé.

Dans un second temps, nous avons décrit la phase de conception d'une chaîne de traitement des requêtes en langue naturelle en deux temps : un premier module, `GRASP`, se charge de l'analyse syntaxico-sémantique de la requête et produit une représentation structurée grammaticalement, puis un second module, `DIG`, transforme celle-ci de manière à fournir en sortie une requête structurée sémantiquement. La conception de ces deux modules est profondément liée à une analyse et annotation manuelle préalable des phrases du corpus *Daft*, qui a permis de produire différentes ressources linguistiques.
`GRASP` repose en effet sur l'utilisation d'un lexique de 4 266 lemmes et de leurs flexions associées $\mathbb{L}_G$, d'un ensemble de 1 294 clés sémantiques $\mathbb{K}_G$ représentant tous les sens occurrant dans le

corpus *Daft* et d'un ensemble de 240 règles d'analyse grammaticales $\mathbb{R}_G$.
De la même manière, `DIG` se fonde en premier lieu sur un langage formel de représentation de la sémantique des requêtes, *DAFT 2.0*, dont nous avons donné la syntaxe et la sémantique, et qui repose sur un ensemble $\mathbb{S}$ de 237 schèmes. Après avoir montré l'efficacité de ce langage sur un sous-ensemble de phrases du corpus *Daft* annoté manuellement, nous avons fait de même sur les représentations formelles produites automatiquement cette fois-ci par `DIG`, à l'aide d'un ensemble $\mathbb{R}_{\mathcal{M}}$ de 24 règles de combinaison des différents éléments de la requête entre eux.

Dans un dernier temps, nous avons cherché à valoriser les éléments produits lors de cette conception. Ainsi, nous avons exploré des méthodes d'identification automatique de la classe conversationnelle des requêtes en fonction de l'analyse fournie par `DIG`. Nous avons également réemployé la méthodologie de conception de `GRASP` et `DIG` dans le cadre du développement d'agents 100% Web, les agents DIVA, notamment pour leur intégration au sein d'une application de taille conséquente : l'environnement collaboratif de conception musicale CODES.
Nous avons enfin réfléchi à la nature de l'architecture nécessaire pour le dernier module de l'agent assistant `DAFT`, l'agent rationnel $\mathtt{A}_R$ en charge de déterminer la réaction de l'agent. Nous avons montré qu'il ne pouvait être entièrement rationnel et proposé en conséquence un modèle psychologique pour celui-ci.

Les travaux présentés dans cette thèse ont donné lieu à plusieurs publications au sein de différentes communautés scientifiques (agents, traitement de la langue, data mining, applications Web ou e-learning), dont 1 revue nationale, 10 conférences (dont 7 internationales) et 5 ateliers (dont 3 internationaux).

Perspectives

Le dernier chapitre a déjà permis d'entrevoir les deux principales perspectives ouvertes par le travail mené dans le cadre de cette thèse :

- La réutilisation de la méthodologie et des ressources développées dans le cadre de collaborations pour déployer des agents assistants au sein d'application réelles. Ceci devrait permettre en retour d'enrichir les ressources linguistiques employées (le corpus *Daft*, le lexique $\mathbb{L}_G$ et les clés sémantiques $\mathbb{K}_G$), et donc à terme d'améliorer les performances du système (moyennant des ajustements aux deux ensembles de règles $\mathbb{R}_G$ et $\mathbb{R}_{\mathcal{M}}$ et éventuellement en étendant l'ensemble des schèmes $\mathbb{S}$).
- Le développement du dernier module du système `DAFT` : l'agent rationnel $\mathtt{A}_R$. Cet agent devra s'appuyer sur des heuristiques exploitant trois données : les requêtes structurées sémantiquement selon le langage *DAFT* fournies par `DIG`, le modèle de l'application assistée et le modèle (notamment psychologique) de l'agent lui-même. En retour, cet agent fournira en sortie des requêtes également exprimées selon le langage formel *DAFT*, et seront finalement traitées par un module de génération de langue naturelle qui reste à définir.

Bibliographie

Abney, Steven P., Flickenger, Dan, Gdaniec, Claudia, Grishman, Ralph, Harrison, Philip, Hindle, Donald, Ingria, Robert, Jelinek, Frederick, Klavans, Judith L., Liberman, Mark, Marcus, Mitchell P., Roukos, Salim, Santorini, B., & Strzalkowski, T. 1991. Procedure for quantitatively comparing the syntactic coverage of English grammars. *Pages 306–311 of :* Black, Ezra (ed), *Proceedings of the workshop on Speech and Natural Language*. Pacific Grove, California : Association for Computational Linguistics.

Adda, Gilles, Mariani, Joseph, Paroubek, Patrick, Rajman, Martin, & Lecomte, Josette. 1999 (July). Métrique et premiers résultats de l'évaluation GRACE des étiqueteurs morphosyntaxiques pour le français. *Pages 15–24 of : Actes de la 16e conférence sur le Traitement Automatique des Langues Naturelles (TALN'99).*

Allen, James F. 1995. *Natural Language Understanding*. 2nd edition edn. The Benjamin/Cummings Publishing Company.

Allen, James F., Schubert, Lenhart K., Ferguson, George, Heeman, Peter, Hwang, Chung Hee, Kato, Tsuneaki, Light, Marc, Martin, Nathaniel, Miller, Bradford, Poesio, Massimo, & Traum, David R. 1995. The TRAINS project : a case study in building a conversational planning agent. *Journal of Experimental & Theoretical Artificial Intelligence*, **7**(1), 7–48.

Allen, James F, Byron, Donna, Dzikovska, Myroslava, Ferguson, George, Galescu, Lucian, & Stent, Amanda. 2000. An Architecture for a Generic Dialogue Shell. *Natural Language Engineering*, **6**(3), 213–228.

Amalberti, René. 1996. *La conduite des systèmes à risques*. PUF.

Amalberti, René, Carbonell, Noëlle, & Falzon, Pierre. 1993. User representations of computer systems in human-computer speech interaction. *International Journal of Man-Machine Studies*, **38**(4), 547–566.

Anderson, Anne, Bader, M., Bard, Ellen G., Boyle, E., Doherty, G.M., Garrod, S., Isard, Stephen, Kowtko, Jacqueline, McAllister, J., Miller, J., Sotillo, Cathy, Thompson, Henry S., & Weinert, R. 1991. The HCRC Map Task Corpus. *Language and Speech*, 351–366.

Androutsopoulos, Ion, & Aretoulaki, Maria. 2003. Natural Language Interaction. *Pages 629–649 of :* Mitkov, Ruslan (ed), *The Oxford Handbook of Computational Linguistics*. Oxford University Press.

Angeli, Antonella De, & Brahnam, Sheryl. 2006 (May). Sex stereotypes and conversational agents. *Pages 22–25 of : Proc. the Workshop on Gender & Interaction.*

Artstein, Ron, Cannon, Jacob, Gandhe, Sudeep, Gerten, Jillian, Henderer, Joe, Leuski, Anton, & Traum, David. 2008 (Dec.). *Coherence of Off-Topic Responses for a Virtual Character*. Tech. rept.

Asher, Nicholas, & Lascarides, Alex. 2003. *Logics of conversation*. Cambridge University Press.

Atkins, B. T, & Lewis, H. M. A. 1996. Language in Use. *In : The Collins-Robert French-English Dictionary*, 1st edn. Harper Collins Publishers.

Atkinson, Robert K. 2002. Optimizing Learning from Examples Using Animated Pedagogical Agents. *Journal of Educational Psychology*, **94**(2), 416–427.

Aust, Harald, Oerder, Martin, Seide, Frank, & Steinbiss, Volker. 1995. The Philips automatic train timetable information system. *Speech Communication*, **17**(3-4), 249–262.

Austin, John. 1962. *How to Do Things with Words*. Oxford University Press.

Ayache, Christelle, Grau, Brigitte, & Vilnat, Anne. 2006 (May). EQueR : the French Evaluation Campaign of Questions Answering Systems. *In : Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC'06)*.

Baader, Franz, Calvanese, Diego, McGuinness, Deborah L., Nardi, Daniele, & Patel-Schneider, Peter (eds). 2007. *The Description Logic Handbook : Theory, Implementation, and Applications*. 2nd edn. Cambridge University Press.

Baker, Collin F., Fillmore, Charles J., & Lowe, John B. 1998. The Berkeley FrameNet Project. *Pages 86–90 of : Proceedings of the 17th international conference on Computational linguistics - Volume 1*. Montreal, Quebec, Canada : Association for Computational Linguistics.

Baker, Sheridan W. 1976. *The Complete Stylist and Handbook*. New York, NY, USA : Thomas Y. Crowell Co.

Bateman, John, & Zock, Michael. 2003. Natural Language Generation. *Pages 284–304 of :* Mitkov, Ruslan (ed), *The Oxford Handbook of Computational Linguistics*. Oxford University Press.

Baylor, Amy L, & Kim, Yanghee. 2004. Pedagogical agent design : The impact of agent realism, gender, ethnicity, and instructional role. *Page 592–603 of :* Lester, James C., Vicari, Rosa Maria, & Paraguaçu, Fabio (eds), *Intelligent Tutoring Systems*. Maceio, Alagoas, Brazil : Springer.

Beaudoin-Lafon, Michel. 1997. Interaction instrumentale : de la manipulation directe à la réalité augmentée. *Pages 97–104 of : Actes des Neuvièmes Journées Francophones sur l'Interaction Homme-Machine (IHM'97)*. Futuroscope : Cépaduès Editions.

Bederson, Benjamin B., Hollan, James D., Perlin, Ken, Meyer, Jonathan, Bacon, David, & Furnas, George. 1996. Pad++ : A Zoomable Graphical Sketchpad For Exploring Alternate Interface Physics. *Journal of Visual Languages & Computing*, **7**(1), 3–32.

Beer, Stan. 2007 (Feb.). *Google manager : Google Apps replaced Microsoft Office at 100,000 businesses*.

Berger, Alexandra. 2006 (Dec.). *La communication entre agents de communautés mixtes : un Langage de Conversation Expressif pour agents artificiels*. Ph.D. thesis, INP Grenoble.

Bernsen, Niels Ole. 1994. Foundations of multimodal representations : a taxonomy of representational modalities. *Interacting with computers*, **6**(4), 347–371.

Beun, R.J., de Vos, Eveliene, & Witteman, C.L.M. 2003. Embodied conversational agents : effects on memory performance and anthropomorphisation. *Pages 315–319 of : Proc. of IVA'2003*, vol. 2792. Berlin : Springer.

Black, Ezra, Garside, Roger, & Leech, Geoffrey. 1993. *Statistically-driven computer grammars of English : The IBM/Lancaster approach.* Language & Computers. Amsterdam : Rodopi.

Blignaut, Pieter, McDonald, Theo, & Tolmie, Janse. 2000. Attitudes and Anxiety Towards Computer Use. *Pages 442–445 of :* Khosrowpour, Mehdi (ed), *Challenges of information technology management in the 21st century.* Anchorage, Alaska, USA : Idea Group Publishing.

Boehm, Barry W. 1988. A Spiral Model of Software Development and Enhancement. *Computer*, **21**(5), 61–72.

Borges, Jorge Luis. 1951. Funès ou la mémoire. *In : Fictions.* Paris : Gallimard.

Bossard, Aurélien. 2007. Vers une ressource prédicative pour l'extraction d'information. *In :* Hathout, Nabil, & Muller, Philippe (eds), *Actes de la 11e Rencontre des Etudiants Chercheurs en Informatique pour le Traitement Automatique des Langues (RECITAL 2007)*, vol. 2. Toulouse, France : IRIT Press.

Bouchet, François, & Sansonnet, Jean-Paul. 2009a. Apports Complémentaires de la Subjectivité et des Biais Cognitifs à la Rationalité dans le Contexte de la Fonction d'Assistance. *Pages 69–80 of :* Maudet, Nicolas, Schobbens, Pierre-Yves, & Guyomard, Marc (eds), *Actes des Cinquièmes Journées Francophones Modèles Formels de l'Interaction (MFI'09).* Lannion, France : Université de Rennes 1.

Bouchet, François, & Sansonnet, Jean-Paul. 2009b (Dec.). A framework for modeling the relationships between the rational and behavioral reactions of assisting conversational agents. *In : Proc. of the 7th European Workshop on Multi-Agent Systems (EUMAS'09).*

Bouchet, François, & Sansonnet, Jean-Paul. 2009c. Subjectivity and Cognitive Biases Modeling for a Realistic and Efficient Assisting Conversational Agent. *Pages 209–216 of : Proc. of 2009 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, vol. 2. Milan, Italy : IEEE Computer Society.

Bouchet, Jullien, Nigay, Laurence, & Ganille, Thierry. 2004. ICARE software components for rapidly developing multimodal interfaces. *Pages 251–258 of : Proceedings of the 6th international conference on Multimodal interfaces.* State College, PA, USA : ACM.

Brajnik, Giorgio, & Tasso, Carlo. 1992. A flexible tool for developing user modeling applications with nonmonotonic reasoning capabilities. *Page 42–66 of : Proc. of the Third International Workshop on User Modeling.*

Brill, Eric. 1995. Transformation-based error-driven learning and natural language processing : A case study in part-of-speech tagging. *Computational linguistics*, **21**(4), 543–565.

Buisine, Stéphanie, & Martin, Jean-Claude. 2005. Children's and Adults' Multimodal Interaction with 2D Conversational Agents. *Pages 1240–1243 of : Proceedings of the SIGCHI conference on Human factors in computing systems.* Portland, Oregon, USA : ACM Press.

Caelen, Jean. 2003. Stratégies de dialogue. *Pages 29–39 of :* Herzig, Andreas, Chaib-Draa, Brahim, & Mathieu, Philippe (eds), *Actes des Secondes Journées Francophones des Modèles Formels de l'Interaction (MFI'03).* Lille, France : Cépaduès.

Calzolari, Nicoletta, Choukri, Khalid, Maegaard, Bente, Mariani, Joseph, Odjik, Jan, Piperidis, Stelios, & Tapias, Daniel (eds). 2008. *Proceedings of the Sixth International Language Resources and Evaluation Conference (LREC'08).* Marrakech, Morocco : European Language Resources Association (ELRA).

Cambre, Marjorie A., & Cook, Desmond L. 1987. Measurement and Remediation of Computer Anxiety. *Educational Technology*, **27**(12), 15–20.

Capobianco, Antonio. 2002. *Stratégies d'aide en ligne contextuelles : Acquisition d'expertises, modélisation et évaluation expérimentale.* Ph.D. thesis, Université Henri Poincaré - Nancy 1.

Capobianco, Antonio, & Carbonell, Noëlle. 2001 (Aug.). Contextual online help : elicitation of human experts' strategies. *Pages 824–828 of : Proceedings of HCI'01.*

Capobianco, Antonio, & Carbonell, Noëlle. 2002 (Oct.). Conception d'aides en ligne pour le grand public : défis et propositions. *Pages 309–335 of :* Drouin, G. Eude J. -M. Robert A. (ed), *Actes du 8ème Colloque Francophone Ergonomie et Informatique Avancée (ERGO-IA'2002).*

Capobianco, Antonio, & Carbonell, Noëlle. 2006. Aides en ligne à l'utilisation de logiciels grand public : problèmes spécifiques de conception et solutions potentielles. *Intellectica*, **2**(44), 87–120.

Carbonell, Noëlle, & Kieffer, Suzanne. 2005. Do oral messages help visual search ? *Page 131–158 of :* van Kuppevelt, Jan, Dybkjaer, Laila, & Bernsen, Niels Ole (eds), *Advances in Natural Multimodal Dialogue.* Dordrecht, Netherlands : Springer.

Carletta, Jean, Isard, Amy, Isard, Stephen, Kowtko, Jacqueline, Doherty-Sneddon, Gwyneth, & Anderson, Anne. 1996 (June). *HCRC dialogue structure coding manual.* Tech. rept. HCRC, University of Edinburgh.

Carroll, John, & Aaronson, Amy. 1988. Learning by doing with simulated intelligent help. *Communications of the ACM*, **31**(9), 1064–1079.

Carroll, John M., & Carrithers, Caroline. 1984. Training wheels in a user interface. *Communications of the ACM*, **27**(8), 800–806.

Carroll, John M., & Mack, Robert L. 1984. Learning to Use a Word Processor : By Doing, by Thinking, and by Knowing. *Pages 13–51 of :* Thomas, John C., & Schneider, Michael L. (eds), *Human Factors in Computer Systems.* Ablex Publishing Corporation.

Carroll, John M., & Rosson, Mary Beth. 1987. Paradox of the active user. *Pages 80–111 of : Interfacing thought : cognitive aspects of human-computer interaction.* MIT Press.

Casali, Ana, Godo, Lluís, & Sierra, Carles. 2004. Graded BDI models for agent architectures. *Pages 126–143 of : Proceedings of CLIMA-V.* Lecture Notes in Computer Science, vol. 3487.

Chanod, Jean-Pierre, & Tapanainen, Pasi. 1995. Tagging French : comparing a statistical and a constraint-based method. *Page 149–156 of : Proceedings of the seventh conference on European chapter of the Association for Computational Linguistics.*

Chapelier, Laurent, Fay-Varnier, Christine, & Roussanaly, Azim. 1995 (June). Modelling an Intelligent Help System from a Wizard of Oz Experiment. *Pages 101–104 of : Proc. of SDS-1995.*

Chicoisne, Guillaume, & Pesty, Sylvie. 1999. Modèle de conversation et agents rationnels socialement corrects. *Pages 91–104 of : Atelier thématique TALN 1999 "La Langue dans l'Interaction Homme-Machine".*

Chicoisne, Guillaume, & Pesty, Sylvie. 2003. Un modèle de conversation mixte pour l'interaction humain/agent. *Technique et Science Informatiques*, **22**, 375–380.

Chomsky, Noam. 1956. Three models for the description of language. *IRE Transactions on Information Theory*, **2**(3), 113–124.

Chu-Carroll, Jennifer. 2000. MIMIC : an adaptive mixed initiative spoken dialogue system for information queries. *Pages 97–104 of : Proceedings of the sixth conference on Applied natural language processing.* Seattle, Washington : Association for Computational Linguistics.

Clement, Lionel, Sagot, Benoît, & Lang, Bernard. 2004. Morphology based automatic acquisition of large-coverage lexica. *Pages 1841–1844 of : Proceedings of LREC'04.*

Cohill, Andrew M., & Williges, Robert C. 1985. Retrieval of Help Information for Novice Users of Interactive Computer Systems. *Human Factors*, **27**(3), 335–343.

Collins, M., & Duffy, N. 2002. Convolution Kernels for Natural Language. *In :* Dietterich, T. G., Becker, S., & Ghahramani, Z. (eds), *Advances in Neural Information Processing Systems 14.* Cambridge, MA : MIT Press.

Corblin, Francis. 1987. *Indéfini, défini et démonstratif.* Langues et cultures. Genève : DROZ.

Core, Mark G., & Allen, James F. 1997 (Nov.). Coding dialogs with the DAMSL annotation scheme. *Pages 28–35 of : Working Notes of the AAAI Fall Symposium on Communicative Action in Humans and Machines.*

Dale, Robert, & Reiter, Ehud. 1995. Computational Interpretations of the Gricean Maxims in the Generation of Referring Expressions. *Cognitive Science*, **19**(2), 233–63.

Damerau, Fred J. 1964. A technique for computer detection and correction of spelling errors. *Commun. ACM*, **7**(3), 171–176.

Danlos, Laurence, Lapalme, Guy, & Lux, Veronika. 2000. Generating a Controlled Language. *Pages 141–147 of : Proceedings of the first international conference on Natural language generation - Volume 14.* Mitzpe Ramon, Israel : Association for Computational Linguistics.

Dastani, Mehdi, & van der Torre, Leendert. 2002. A classification of cognitive agents. *Pages 256—261 of : Proceedings of Cogsci02.*

Devillers, Laurence, Maynard, Hélène, Rosset, Sophie, Paroubek, Patrick, Mostefa, Djamel, Choukri, Khalid, Charnay, Laurent, Bousquet, Caroline, Vigouroux, Nadine, Béchet, Frédéric, Romary, Laurent, Antoine, Jean-Yves, Villaneau, Jeanne, Vergnes, Myriame, & Goulian, Jérôme. 2004 (May). The French MEDIA/EVALDA project : the evaluation of the understanding capability of Spoken Language Dialogue Systems. *Pages 2131–2134 of : Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC'04)*, vol. 6.

Donini, Francesco M., Lenzerini, Maurizio, Nardi, Daniele, & Schaerf, Andrea. 1996. Reasoning in description logics. *Pages 191–236 of : Principles of knowledge representation.* Center for the Study of Language and Information.

Doyle, Patrick. 1999 (May). When is a communicative agent a good idea. *In : Proc. of the 3rd International Conference on Autonomous Agents.*

Duffy, Thomas M., Mehlenbacher, Brad, & Palmer, James E. 1989. The evaluation of online help systems : a conceptual model. *Pages 362–387 of : The society of text : hypertext, hypermedia, and the social construction of information.* MIT Press.

Dutoit, Thierry, & Stylianou, Yannis. 2003. Text-To-Speech Synthesis. *Pages 323–338 of :* Mitkov, Ruslan (ed), *The Oxford Handbook of Computational Linguistics*. Oxford University Press.

Dzida, Wolfgang, Herda, S., & Itzfeldt, W. D. 1978. User-perceived quality of interactive systems. *Pages 188–195 of : Proceedings of the 3rd international conference on Software engineering*. Atlanta, Georgia, United States : IEEE Press.

Dzida, Wolfgang, Hoffmann, C., & Valder, W. 1987 (Sept.). Mastering the complexity of dialogue systems by the aid of work contexts. *Pages 29–33 of :* Bullinger, H.G., Kornwachs, K., & Shackel, B. (eds), *Proceedings of the Second IFIP Conference on Human-Computer Interaction*.

Engelbart, Douglas C. 1962 (Oct.). *Augmenting Human Intellect : a Conceptual Framework*. Tech. rept. AFOSR-3223. Stanford Research Institute.

Evertsz, Rick, Ritter, Franck E., Busetta, Paolo, & Pedrotti, Matteo. 2008. Realistic Behaviour Variation in a BDI-based Cognitive Architecture. *In : Proc. of SimTecT'08*.

Falzon, Pierre, Amalberti, René, & Carbonell, Noëlle. 1986. Dialogue Control Strategies in Oral Communication. *Pages 73–98 of :* Hopper, K.T., & Newman, I.A. (eds), *Foundation for Human-Computer Communication*. Résumé disponible dans les fichiers attachés.

Fellbaum, Christiane. 1998. *WordNet : An Electronic Lexical Database*. MIT Press.

Ferguson, George, & Allen, James F. 1998. TRIPS : an integrated intelligent problem-solving assistant. *Pages 567–572 of : AAAI'98/IAAI'98 : Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*. Menlo Park, CA, USA : American Association for Artificial Intelligence.

Fernandez, Ana, Saint-Dizier, Patrick, Vazquez, Gloria, Benamara, Farah, & Kamel, Mouna. 2002. The VOLEM project : a framework for the construction of advanced multilingual lexicons. *Pages 89–98 of : Proceedings of the Language Engineering Conference*. IEEE Computer Society.

Fillmore, Charles J. 1982. Frame semantics. *Pages 373–400 of : Cognitive linguistics : basic readings*.

Finin, Tim, Fritzson, Richard, McKay, Don, & McEntire, Robin. 1994. KQML as an agent communication language. *Pages 456–463 of : Proceedings of the third international conference on Information and knowledge management*. Gaithersburg, Maryland, United States : ACM.

Fink, Josef, Kobsa, Alfred, & Nill, Andreas. 1998. Adaptable and adaptive information provision for all users, including disabled and elderly people. *New Review of Hypermedia and Multimedia*, **4**(1), 163–188.

Fisher, James. 1991. Defining the novice user. *Behaviour & Information Technology*, **10**(5), 437–441.

Francis, W. Nelson. 1964. *A standard sample of present-day English for use with digital computers*. Report to the U.S Office of Education on Cooperative Research Project E-007. Brown University, Providence, RI.

Furnas, George W. 1986. Generalized fisheye views. *SIGCHI Bull.*, **17**(4), 16–23.

Gaertner, Thomas. 2003. A survey of kernels for structured data. *SIGKDD Explor. Newsl.*, **5**(1), 49–58.

Gaiffe, Bertrand, Landragin, Frédéric, & Quignard, Matthieu. 2004. Le dialogue naturel comme un service dans un contexte multi-applicatif. *Pages 57–66 of :* Paroubek, Patrick, & Sansonnet, Jean-Paul (eds), *Actes de la Journée d'Etude ATALA Agental "Agent et Langue"*. Paris, France : ATALA.

Gale, William A., Church, Kenneth W., & Yarowsky, David. 1992. One sense per discourse. *Pages 233–237 of : Proceedings of the workshop on Speech and Natural Language*. Harriman, New York : Association for Computational Linguistics.

Galluccio, Roberto G. Pérez. 2006 (Oct.). Humanizing CALL : The use of pedagogical agents as language tutors. *In : NERALLT 2006*.

Gardent, Claire, Guillaume, Bruno, Perrier, Guy, & Falk, Ingrid. 2006 (Apr.). Extraction d'information de sous-catégorisation à partir des tables du LADL. *Pages 139–148 of : Verbum ex machina : actes de la 13e conférence sur le traitement automatique des langues naturelles (TALN 2006)*.

Garrett, Jesse James. 2005 (Feb.). *Ajax : A new approach to web applications*.

Gazdar, Gerald. 1979. *Pragmatics : Implicature, presupposition and logical form*. Academic Press.

Geach, Peter T. 1962. *Reference and Generality : An Examination of Some Medieval and Modern Theories*. Ithaca, NY : Cornell University Press.

Gentner, Don, & Nielsen, Jakob. 1996. The Anti-Mac interface. *Commun. ACM*, **39**(8), 70–82.

Gibson, James J. 1977. The Theory of Affordances. *Pages 67–82 of :* Shaw, Robert, & Bransford, John (eds), *Perceiving, Acting and Knowing*.

Goddeau, D., Brill, E., Glass, J. R, Pao, C., Phillips, M., Polifroni, J., Seneff, S., & Zue, V. W. 1994. Galaxy : A human-language interface to on-line travel information. *In : Third International Conference on Spoken Language Processing*.

Godfrey, J.J., & Holliman, E. 1997. Switchboard-1 Release 2. *Linguistics Data Consortium*.

Goldberg, Lewis R. 1981. Language and individual differences : The search for universal in personality lexicons. *Review of personality and social psychology*, **2**, 141–165.

Goodwin, Nancy C. 1987. Functionality and usability. *Commun. ACM*, **30**(3), 229–233.

Graesser, Art, Jackson, G., Ventura, Matthew, Mueller, James, Hu, Xiangen, & Person, Natalie. 2003. The Impact of Conversational Navigational Guides on the Learning, Use, and Perceptions of Users of a Web Site. *Page CH423 of : Agent-Mediated Knowledge Management*.

Greene, Barbara, & Rubin, Gerald. 1971. *Automatic Grammatical Tagging of English*. Providence, RI : Brown University Press.

Grice, Herbert Paul. 1975. Logic and Conversation. *Syntax and semantics*, **3**(Speech Acts), 41–58.

Grishman, Ralph, & Kittredge, Richard (eds). 1986. *Analyzing language in restricted domains : sublanguage description and processing*. Lawrence Erlbaum Associates, Inc.

Gross, Maurice. 1975. *Méthodes en syntaxe*. Hermann Paris.

Gérard, Sébastien. 2003 (June). *Un modèle cognitif pour l'interprétation des expressions référentielles dans le cadre d'un système générique de dialogue Homme-Machine*. Ph.D. thesis.

Harris, Grant, Begg, Ian, & Upfold, Douglas. 1980. On the role of the speaker's expectations in interpersonal communication. *Journal of Verbal Learning & Verbal Behavior*, **19**(5), 597–607.

Harrison, Susan M. 1995. A comparison of still, animated, or nonillustrated on-line help with written or spoken instructions in a graphical user interface. *Pages 82–89 of : Proceedings of the SIGCHI conference on Human factors in computing systems*. Denver, Colorado, USA : ACM Press/Addison-Wesley Publishing Co.

Hasson, Cyril. 2007. *Étude de modèles d'assistance pour les contenus web actifs*. Masters thesis, Université Paris-Sud 11.

Hatier, Fredérique (ed). 2006. *Bescherelle : La Conjugaison Pour Tous*. Hatier.

Heeman, Peter, & Allen, James F. 1995. The TRAINS Spoken Dialog Corpus. CD-ROM. *Linguistics Data Consortium*.

Horn, Laurence R., & Ward, Gregory (eds). 2005. *The handbook of pragmatics*. Blackwell Handbooks in Linguistics. Blackwell Publishing.

Horton, William. 1994. *Designing and Writing Online Documentation : Hypermedia for Self-Supporting Products*. 2 edn. New York, NY, USA : John Wiley & Sons, Inc.

Horvitz, Eric, Breese, Jack, Heckerman, David, Hovel, David, & Rommelse, Koos. 1998. The Lumiere project : Bayesian user modeling for inferring the goals and needs of software users. *Page 256–265 of :* Cooper, G.E., & Moral, S. (eds), *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*. San Francisco, CA, USA : Morgan Kaufmann.

Horvitz, Eric, Breese, John S., Heckerman, David, Hobson, Samuel D., Hovel, David O., Klein, Adrien C., Rommelse, Jacobus A., & Shaw, Gregory L. 2001 (July). *Intelligent User Assistance Facility*.

Hovy, Eduard. 2000. *Language Generation*.

Howden, Nick, Ronnquist, Ralph, Hodgson, Andrew, & Lucas, Andrew. 2001 (May). JACK intelligent agents-summary of an agent infrastructure. *In : 5th International Conference on Autonomous Agents*.

Ide, Nancy, & Véronis, Jean. 1998. Introduction to the special issue on word sense disambiguation : the state of the art. *Computational Linguistics*, **24**(1), 2–40.

Isbister, Katherine, Nakanishi, Hideyuki, Ishida, Toru, & Nass, Cliff. 2000. Helper agent : designing an assistant for human-human interaction in a virtual meeting space. *Pages 57–64 of : Proceedings of the SIGCHI conference on Human factors in computing systems*. The Hague, The Netherlands : ACM.

Jackson, So-Ryang, Cherry, Joan, & Fryer, Barbara. 1992. Online scenario-based task help. *IEEE Transactions on Professional Communication*, **35**(2), 91–97.

Jameson, Anthony, Großmann-Hutter, Barbara, March, Leonie, & Rummer, Ralf. 2000. Creating an empirical basis for adaptation decisions. *Pages 149–156 of : Proceedings of the 5th international conference on Intelligent user interfaces*. New Orleans, Louisiana, United States : ACM.

Johnson, Jeff, Roberts, Teresa L., Verplank, William, Smith, David .C., Irby, Charles H., Beard, Marian, & Mackey, Kevin. 1989. The Xerox Star : a retrospective. *Computer*, **22**(9), 11–26, 28–29.

Jonassen, David H. 1988. Designing structured hypertext and structuring access to hypertext. *Educational Technology*, **28**(11), 13–16.

Jonassen, David H. 1993. Effects of Semantically Structured Hypertext Knowledge Bases on Users' Knowledge Structures. *In :* McKnight, C., Dillon, A., & Richardson, J. (eds), *Hypertext : A Pyshcological Perspective*. New York : Ellis Horwood.

Joseph, Biju, Steinberg, Esther R., & Jones, A. Russell. 1989. User perceptions and expectations of an information retrieval system. *Behaviour & Information Technology*, **8**(2), 77–88.

Jurafsky, Daniel, Bates, Rebecca, Coccaro, Noah, Martin, Rachel, Meteer, Marie, Ries, Klaus, Shriberg, Elizabeth, Stolcke, Andreas, Taylor, Paul, & Ess-Dykema, Carol Van. 1998. *Switchboard Discourse Language Modeling Project Final Report*. Tech. rept. Center for Speech and Language Processing, Johns Hopkins University, Baltimore, MD, USA.

Jurafsky, Daniel, Martin, James H., Russell, Stuart, & Norvig, Peter. 2008. Dialogue and Conversational Agents. *Pages 847–894 of : Speech and Language Processing*, 2nd edn. Prentice Hall Series in Artificial Intelligence. Upper Saddle River, NJ, USA : Prentice Hall.

Kamp, Hans. 1981. A Theory of Truth and Semantic Representation. *Pages 277–322 of :* Groenendijk, J., Janssen, Th., & Stokhof, M. (eds), *Formal Methods in the Study of Language*. Amsterdam : Mathematisch Centrum.

Kamp, Hans, & Reyle, Uwe. 1993. *From Discourse to Logic : Introduction to Model-theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Studies in Linguistics and Philosophy, vol. 42. Dordrecht : Kluwer Academic Publishers.

Kay, Alan C. 1977. Microelectronics and the personal computer. *Scientific American*, **237**(3), 230–244.

Kearsley, Greg. 1988. *Online Help Systems : Design and Implementation*. Human/Computer Interaction, no. 12. Ablex Publishing Corporation.

Kelly, Michael J, & Chapanis, Alphonse. 1977. Limited Vocabulary Natural Language Dialogue. *International Journal of Man-Machine Studies*, **9**(4), 479–501.

Kerr, Stephen T. 1986. Instructional Text : The Transition from Page to Screen. *Visible Language*, **20**(4), 368–392.

Kilgarriff, Adam. 2001. Comparing corpora. *International journal of corpus linguistics*, **6**(1), 97–133.

Kingsbury, Paul, & Palmer, Matha. 2002. From treebank to propbank. *Pages 1989–1993 of : Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC-2002)*.

Kittler, J. 1998. Combining classifiers : A theoretical framework. *Pattern Analysis & Applications*, **1**(1), 18–27.

Kittredge, Richard I. 2003. Sublanguages and controlled languages. *Pages 430–447 of :* Mitkov, Ruslan (ed), *The Oxford Handbook of Computational Linguistics*. Oxford University Press.

Krämer, Nicole C., Tietz, Bernd, & Bente, Gary. 2003. Effects of Embodied Interface Agents and Their Gestural Activity. *Pages 292–300 of : Proc. of IVA'2003*, vol. 2792. Berlin : Springer.

Kučera, Henry, & Francis, W. Nelson. 1967. *Computational analysis of present-day English.* Providence, RI : Brown University Press.

Lamel, Lori, & Gauvain, Jean-Luc. 2003. Speech Recognition. *Pages 305–322 of :* Mitkov, Ruslan (ed), *The Oxford Handbook of Computational Linguistics.* Oxford University Press.

Landragin, Frédéric. 2003 (Apr.). *Modélisation de la communication multimodale : vers une formalisation de la pertinence.* Ph.D. thesis, Université de Nancy 2.

Landragin, Frédéric. 2004. Interface sémantique-pragmatique et domaines de référence. *In : Quatrièmes Journées d'Études Linguistiques de Nantes (JEL 2004).*

Landragin, Frédéric, Bellalem, Nadia, & Romary, Laurent. 2002. Referring to Objects with Spoken and Haptic Modalities. *Pages 99–104 of : Proc. of the Fourth IEEE International Conference on Multimodal Interfaces.* Pittsburgh, PA : IEEE Computer Society Press.

Larsson, Staffan, & Traum, David. 2001. Information state and dialogue management in the TRINDI dialogue move engine toolkit. *Natural Language Engineering*, **6**(3&4), 323–340.

Lassila, Ora, & Swick, Ralph R. 1999. *Resource Description Framework (RDF) Model and Syntax Specification.* W3C Recommendation.

Le Guern, Karl. 2004 (Sept.). *Définition d'une architecture de médiateur pour des agents conversationnels animés.* Masters thesis, Université Paris-Sud 11.

Leray, David. 2009 (Dec.). *D'une famille de composants dialogiques à une méthodologie de synthèse de modèles d'assistance pour un Agent Conversationnel Assistant.* Ph.D. thesis, Université Paris-Sud 11.

Leray, David, & Sansonnet, Jean-Paul. 2007 (Sept.). Assisting Dialogical Agents Modeled from Novice User's Perceptions. *Pages 1122–1129 of : Proceedings of KES'07.* LNAI, vol. 4693.

Lesk, Michael. 1986. Automatic sense disambiguation using machine readable dictionaries : How to tell a pine cone from an ice cream cone. *Pages 24–26 of : Proceedings of the 5th annual international conference on Systems documentation.*

Lester, James C., Converse, Sharolyn A., Kahler, Susan E., Barlow, S. Todd, Stone, Brian A., & Bhogal, Ravinder S. 1997. The persona effect : affective impact of animated pedagogical agents. *Pages 359–366 of : Proceedings of the SIGCHI conference on Human factors in computing systems.* Atlanta, Georgia, United States : ACM.

Levelt, Willem J.M. 1989. *Speaking : from intention to articulation.* MIT Press.

Luzzati, Daniel. 1995. *Le dialogue verbal homme-machine : études de cas.* Masson.

Mack, Robert L., Lewis, Clayton H., & Carroll, John M. 1983. Learning to use word processors : problems and prospects. *ACM Transactions on Information Systems*, **1**(3), 254–271.

Marcus, Mitchell P., Santorini, Beatrice, & Marcinkiewicz, Mary Ann. 1994. Building a large annotated corpus of English : The Penn Treebank. *Computational linguistics*, **19**(2), 313–330.

Marshall, I. 1987. Tag selection using probabilistic methods. *The Computational analysis of English : a corpusbased approach*, 42–65.

Maïs, Chantal. 1989. *L'adaptation de l'aide à l'utilisateur : aider les programmeurs occasionnels à opérationaliser leurs plans sous-optimaux*. Ph.D. thesis, Université Aix-Marseille 1.

McEnery, Tony. 2003. Corpus Linguistics. *Pages 448–463 of :* Mitkov, Ruslan (ed), *The Oxford Handbook of Computational Linguistics*. Oxford University Press.

McGuinness, Deborah L., & van Harmelen, Franck. 2003 (Aug.). *OWL Web Ontology Language Overview*. Candidate Recommendation. World Wide Web Consortium (W3C).

Mercer, James. 1909. Functions of positive and negative type and their connection with the theory of integral equations. *Philos. Trans. Roy. Soc. London*, **209**.

Merchant, Guy. 2001. Teenagers in cyberspace : an investigation of language use and language change in internet chatrooms. *Journal of Research in Reading*, **24**(Oct.), 293–306.

Miletto, Evandro Manara, Flores, Luciano Vargas, Pimenta, Marcelo Soares, Rutily, Jérôme, & Santagada, Leonardo. 2007. Interfaces for musical activities and interfaces for musicians are not the same : the case for codes, a web-based environment for cooperative music prototyping. *Pages 201–207 of : Proceedings of the 9th international conference on Multimodal interfaces*. Nagoya, Aichi, Japan : ACM.

Miletto, Evandro Manara, Sansonnet, Jean-Paul, Pimenta, Marcelo Soares, & Bouchet, François. 2009. Corpus-based design of a Web 2.0 Assisting Agent. *Pages 52–63 of :* Olsina, Luis, Pastor, Oscar, Schwabe, Daniel, Rossi, Gustavo, & Winckler, Marco (eds), *Proc. of the 8th International Workshop on Web-Oriented Software Technologies (IWWOST'2009)*, vol. 493. San Sebastian, Spain : CEUR Workshop Proceedings.

Minsky, Marvin. 1974. *A Framework for Representing Knowledge*. Technical Report AIM-306. Massachusetts Institute of Technology.

Molinsky, Steven J, & Bliss, Bill. 1994. Inventory of functions and conversation strategies : The Comprehensive course in functional English. Prentice Hall.

Morel, Mary-Annick. 1989. *Analyse linguistique de corpus*. Paris, France : Publications de la Sorbonne Nouvelle.

Moreno, Kristen, Klettke, Bianca, Nibbaragandla, Kiran, & Graesser, Arthur. 2002. Perceived Characteristics and Pedagogical Efficacy of Animated Conversational Agents. *Pages 963–971 of : Intelligent Tutoring Systems*.

Moreno, Roxana, & Mayer, Richard E. 2000. Life-Like Pedagogical Agents in Constructivist Multimedia Environments : Cognitive Consequences of Their Interaction. *Proceedings of the World Conference On Educational Multimedia Hypermedia and Telecommunications (ED-MEDIA)*, 741–746.

Morrell, Roger W., & Park, Denise C. 1993. The effects of age, illustrations, and task variables on the performance of procedural assembly tasks. *Psychology and Aging*, **8**(3), 389–99. PMID : 8216959.

Negron, J. A. 1995. The impact of computer anxiety and computer resistance on the use of computer technology by nurses. *Journal of Nursing Staff Development*, **11**(3), 172–175.

Newell, Allen. 1982. The Knowledge Level. *Artificial Intelligence*, **18**(1), 87–127.

Nicolle, Anne, Pierrel, Jean-Marie, Romary, Laurent, Sabah, Gérard, Vilnat, Anne, & Vivier, Jean. 1998. *Machine, Langage et Dialogue*. Paris : L'Harmattan.

Nigay, Laurence, & Coutaz, Joëlle. 1995. A generic platform for addressing the multimodal challenge. *Pages 98–105 of : Proceedings of the SIGCHI conference on Human factors in computing systems*. Denver, Colorado, United States : ACM Press/Addison-Wesley Publishing Co.

Norling, Emma, & Ritter, Franck E. 2004. Towards Supporting Psychologically Plausible Variability in Agent-Based Human Modelling. *In : Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems*.

Norman, Donald A. 1994. How might people interact with agents. *Commun. ACM*, **37**(7), 68–71.

Novick, David G., & Ward, Karen. 2006. Why don't people read the manual ? *Pages 11–18 of : Proceedings of the 24th annual ACM international conference on Design of communication*. Myrtle Beach, SC, USA : ACM.

O'Brien, P. D., & Nicol, R. C. 1998. FIPA — Towards a Standard for Software Agents. *BT Technology Journal*, **16**(3), 51–59.

Ortony, Andrew. 1993. *Metaphor and Thought*. 2nd edn. Cambridge University Press.

Ortony, Andrew. 2003. On making believable emotional agents believable. *Pages 189–211 of :* Trappl, Robert, Petta, Paolo, & Payr, Sabine (eds), *Emotions in humans and artifacts*. Cambridge, MA : MIT Press.

Ozkan, Nadine. 1994. *Vers un modèle dynamique du dialogue : analyse de dialogues finalisés dans une perspective communicationnelle*. Ph.D. thesis, INP Grenoble.

Padro, Lluis. 1998. *A Hybrid Environment for Syntax-Semantic Tagging*. Ph.D. thesis, Universitat Politechnica de Catalunya.

Paiva, Ana, & Self, John A. 1994. Tagus — A user and learner modeling workbench. *User Modeling and User-Adapted Interaction*, **4**(3), 197–226.

Palmer, James E. 1992. Medium of delivery and the design process. *Pages 23–40 of :* Duffy, Thomas M., Palmer, James E., & Mehlenbacher, Brad (eds), *Online Help : Design and evaluation*. Norwood, NJ : Ablex Publishing Corporation.

Palmiter, Susan, & Elkerton, Jay. 1991. An evaluation of animated demonstrations of learning computer-based tasks. *Pages 257–263 of : Proceedings of the SIGCHI conference on Human factors in computing systems : Reaching through technology*. New Orleans, Louisiana, USA : ACM.

Paraiso, Emerson Cabrera, & Barthès, Jean-Paul A. 2004. Une interface conversationnelle pour les agents assistants appliqués à des activités professionnelles. *Pages 243–246 of : Proceedings of the 16th conference on Association Francophone d'Interaction Homme-Machine*. Namur, Belgium : ACM.

Paroubek, Patrick. 2000 (June). Language Resources as by-Product of Evaluation : The MULTITAG Example. *Pages 151–154 of : Proceedings of the 2nd International Conference on Language Resources and Evaluation (LREC2000)*, vol. 1.

Paroubek, Patrick, Robba, Isabelle, Vilnat, Anne, & Ayache, Christelle. 2008. Easy, evaluation of parsers of french : what are the results. *Pages 2480–2486 of : Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC'08).* Marrakech, Morocco : European Language Resources Association (ELRA).

Patel, Ronakkumar, Leuski, Anton, & Traum, David. 2006. Dealing with Out of Domain Questions in Virtual Characters. *Pages 121–131 of : Intelligent Virtual Agents.*

Pauchet, Alexandre, Seghrouchni, Amal El Fallah, & Chaignaud, Nathalie. 2007 (May). A Computational Model of Human Interaction and Planning for Heterogeneous Multi-Agent Systems. *Pages 391–393 of : Proc. of the Sixth Intl. Joint Conf. on Autonomous Agents and Multiagent Systems.*

Pellom, B., Ward, W., Hansen, J., Hacioglu, K., Zhang, J., Yu, X., & Pradhan, S. 2001. University of Colorado dialog systems for travel and navigation. *In : Proc. HLT.*

Pereira, David, Oliveira, Eugenio, & Moreira, Nelma. 2008. Formal modelling of emotions in BDI agents. *Pages 62–81 of :* Sadri, F., & Satoh, K. (eds), *Proceedings of CLIMA-VIII.* LNAI, vol. 5056. Porto, Portugal : Springer-Verlag.

Peters, Carol, Jijkoun, Valentin, Mandl, Thomas, Müller, Henning, Oard, Douglas W., Peñas, Anselmo, Petras, Vivien, & Santos, Diana (eds). 2008. *Advances in Multilingual and Multimodal Information Retrieval - 8th Workshop on the Cross-Language Evaluation Forum (CLEF 2007).* Lecture Notes in Computer Science, no. 5152. Berlin / Heidelberg : Springer.

Pitel, Guillaume. 2004 (Sept.). *MICO : La notion de construction située pour un modèle d'interprétation et de traitement de la référence pour le dialogue finalisé.* Ph.D. thesis.

Pitrat, Jacques. 1981. *Réalisation d'un analyseur-générateur lexicographique général.* Rapport de recherche GR22 79/2. Université Paris 6, Paris, France.

Post, Brechtje. 2000. *Tonal and phrasal structures in French intonation.* Ph.D. thesis, Katholieke Universiteit Nijmegen.

Rao, Anand S., & Georgeff, Michael P. 1991. Modelling Rational Agents within a BDI Architecture. *Pages 473–484 of :* Fikes, R., & Sandewall, E. (eds), *Proceedings of Knowledge Representation and Reasoning.* San Mateo, CA, USA : Morgan Kaufmann.

Rastier, François. 1994. La microsémantique. *Pages 43–82 of : Sémantique pour l'analyse.* Paris : Masson.

Rathnam, Gene. 2005. Exploring the Perceived Usefulness of System Documentation. *International Journal of Information and Communication Technology Education*, **1**(1), 31–41.

Rayson, Paul, & Garside, Roger. 2000. Comparing corpora using frequency profiling. *Pages 1–6 of : Proceedings of the workshop on Comparing corpora - Volume 9.* Hong Kong : Association for Computational Linguistics.

Reeves, Byron, & Nass, Clifford. 1996. *The media equation : how people treat computers, television, and new media like real people and places.* Cambridge university press edn.

Ripoche, Gabriel. 2006 (June). *Sur les traces de Bugzilla.* Ph.D. thesis, Université Paris-Sud 11.

Roeck, Anne De, Kruschwitz, Udo, Scott, Paul, Steel, Sam, Turner, Ray, & Webb, Nick. 2000. The YPA - An Assistant for Classified Directory Enquiries. *Pages 239–258 of : Intelligent Systems and Soft Computing.*

Rogers, Carl R. 1957. The Necessary and Sufficient Conditions of Therapeutic Personality Change. *Journal of Consulting Psychology*, **21**(2), 95–103.

Sabouret, Nicolas. 2002 (Dec.). *Etude de modèles de représentations, de requêtes et de raisonnement sur le fonctionnement des comosants actifs pour l'interaction homme-machine.* Ph.D. thesis, Université Paris-Sud 11.

Sabouret, Nicolas, & Sansonnet, Jean-Paul. 2000. Un modèle de raisonnement sur les systèmes évolutifs fondés sur l'intensionnalisation de chroniques. *Pages 305–314 of : Actes du 12e Congrès Francophone AFRIP-RFIA*, vol. 2.

Salmon-Alt, Susanne. 2001 (May). *Référence et dialogue finalisé : de la linguistique à un modèle opérationnel.* Ph.D. thesis.

Samuelsson, Christer, & Voutilainen, Atro. 1997. Comparing a linguistic and a stochastic tagger. *Page 246–253 of : Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics.*

Sankoff, David, & Sankoff, Gillian. 1973. Sample survey methods and computer assisted analysis in the study of grammatical variation. *Pages 7–64 of :* Darnell, Regna (ed), *Canadian Languages in their Social Context.* Edmonton : Linguistic Research.

Sansonnet, Jean-Paul. 1999. *Description scientifique du projet Interviews - The Interviews project.* Technical Report 99-01. LIMSI-CNRS, Orsay, France.

Sansonnet, Jean-Paul, & Leray, David. 2007 (Apr.). Kiwi : An environment for capturing the Perceptual Cues of an Application for an Assisting Conversational Agent. *In : Proc. of Language, Speech and Gesture for Expressive Characters Symposium at AISB'07.*

Sansonnet, Jean-Paul, Miletto, Evandro Manara, Bouchet, François, & Pimenta, Marcelo Soares. 2009. Exploring the integration of teaching capabilities into a CSCP framework through help agents. *In : Proceedings of the 20th Brazilian Symposium of Computer Science in Education (SBIE 2009).* Florianópolis, SC, Brazil : (To be published).

Santhanam, Radhika, & Wiedenbeck, Susan. 1993. Neither novice nor expert : the discretionary user of software. *International Journal of Man-Machine Studies*, **38**(2), 201–229.

Schank, Roger C., & Abelson, Robert P. 1977. *Scripts, plans, goals and understanding : An inquiry into human knowledge structures.* Oxford, England : Lawrence Erlbaum Associates, Inc.

Schmid, Helmut. 1994. Probabilistic part-of-speech tagging using decision trees. *In : Proceedings of International Conference on New Methods in Language Processing*, vol. 12.

Searle, John Rogers. 1969. *Speech Acts : An essay in the Philosophy of language.* New edn. Cambridge University Press.

Sebillotte, Suzanne, & Scapin, Dominique L. 1994. From Users' Task Knowledge to High-Level Interface Specification. *International Journal of Human-Computer Interaction*, **6**(1), 1–15.

Sebrechts, Marc M., & Swartz, Merryanna L. 1991. Question asking as a tool for novice computer skill acquisition. *Pages 293–299 of : Proceedings of the SIGCHI conference on Human factors in computing systems : Reaching through technology.* New Orleans, Louisiana, United States : ACM.

Seneff, Stephanie, & Polifroni, Joseph. 2000. Dialogue management in the Mercury flight reservation system. *Page 1–6 of : Proc. ANLP-NAACL.*

Shneiderman, Ben. 1980. *Software psychology : Human factors in computer and information systems.* Winthrop Publishers.

Shneiderman, Ben. 1983. Direct Manipulation : A Step Beyond Programming Languages. *Computer*, **16**(8), 57–69.

Shneiderman, Ben. 1987. *Designing the user interface : strategies for effective human-computer interaction.* Addison-Wesley.

Simonin, Jérôme, & Carbonell, Noëlle. 2007. Interfaces Adaptatives - Adaptation dynamique à l'utilisateur courant. *In :* Saleh, Imad, & Regottaz, Djeff (eds), *Interfaces Numériques.* Information, hypermédias et communication. Hermès - Lavoisier.

Sloman, Aaron. 2000. Architectural Requirements for Human-like Agents Both Natural and Artificial. *In :* Dautenhahn, Kerstin (ed), *Human Cognition and Social Agent Technology (Advances in Consciousness Research).* John Benjamins Publishing Co.

Streitz, Norbert A. 1988. Mental models and metaphors : implications for the design of adaptive user-system interfaces. *Pages 164–186 of : Learning Issues for Intelligent Tutoring Systems.* New York : Springer-Verlag.

Thacker, Chuck, McCreight, Edward, Lampson, Butler, Sproull, Robert, & Boggs, David. 1979. Alto : A personal computer. *Pages 549–572 of :* Siewiorek, Daniel P., Bell, C. Gordon, & Newell, Allen (eds), *Computer Structures : Principles and Examples*, 2nd edn. New York, NY, USA : McGraw-Hill.

Thibault, Pierrette, & Vincent, Diane. 1990. *Un corpus de français parlé - Montréal 84 : historique, méthodes et perspectives de recherche.* Tech. rept. 1. CIRAL, Université de Laval, Québec, Canada.

Traum, David, Marsella, Stacy, Rickel, Jeff, Gratch, Jonathan, & Swartout, William. 2002. Toward a new generation of virtual humans for interactive experiences. *IEEE Intell. Syst.*, **17**(4), 32–38.

Trenner, Lesley. 1989. A Comparative Study of the Friendliness of Online "Help" in Interactive Information Retrieval Systems. *Information Processing and Management*, **25**(2), 119–136.

Trumbly, James E., Arnett, Kirk P., & Martin, Merle P. 1993. Performance effect of matching computer interface characteristics and user skill level. *International Journal of Man-Machine Studies*, **38**(4), 713–724.

Tulving, Endel. 1983. *Elements of episodic memory.* Oxford, England : Clarendon Press.

Tversky, Barbara, Morrison, Julie Bauer, & Betrancourt, Mireille. 2002. Animation : can it facilitate ? *International Journal of Human Computer Studies*, **57**(4), 247–262.

Twitchell, Douglas P., Adkins, Mark, Nunamaker, Jay F., & Burgoon, Judee K. 2004. Using Speech Act Theory to Model Conversations for Automated Classification and Retrieval. *Pages 121–129 of : Proceedings of the 9th International Working Conference on the Language-Action Perspective on Communication Modeling.* New Brunswick, NJ : Rutgers University Press.

Ummelen, N., & Neutelings, R. 2000. Measuring reading behavior in policy documents : a comparison of two instruments. *IEEE Transactions on Professional Communication*, **43**(3), 292–301.

Vanderveken, Daniel. 1990a. *Meaning and speech acts : Formal semantics of Success and Satisfaction.* Vol. 2. Cambridge University Press.

Vanderveken, Daniel. 1990b. *Meaning and speech acts : principles of language use.* Vol. 1. Cambridge University Press.

Vishwanathan, S.V.N., & Smola, Alexander Johannes. 2004. Fast Kernels for String and Tree Matching. *Pages 113–130 of : Kernel Methods in Computational Biology.* The MIT Press.

Vivier, Jean, & Nicolle, Anne. 1997. Questions de méthode en dialogue homme-machine : l'expérience Compèrobot. *Pages 249–305 of : Machine, Langage et Dialogue.* Paris : L'Harmattan.

Voorhees, Ellen M., & Buckland, Lori P. (eds). 2008. *The Seventeenth Text REtrieval Conference Proceedings (TREC 2008).* NIST Special Publication, nos. 500–277.

Vossen, Piek (ed). 1998. *EuroWordNet : a multilingual database with lexical semantic networks.* Kluwer Academic Publishers.

Voutilainen, Atro. 2003. Part-Of-Speech Tagging. *Pages 219–232 of :* Mitkov, Ruslan (ed), *The Oxford Handbook of Computational Linguistics.* Oxford University Press.

Véronis, Jean. 1998. A study of polysemy judgments and inter-annotator agreement. *In : Programme and advanced papers of the Senseval workshop.*

Walker, Marilyn A., & Rambow, Owen C. 2002. Spoken language generation. *Computer Speech and Language*, **16**(3), 273–282.

Wallace, Richard S. 2003. *The elements of AIML style.*

Wallace, Richard S. 2008. The Anatomy of A.L.I.C.E. *Pages 181–210 of : Parsing the Turing Test.* Springer.

Weizenbaum, Joseph. 1966. ELIZA : a computer program for the study of natural language communication between man and machine. *Commun. ACM*, **9**(1), 36–45.

West, Jamie L. 1995 (Dec.). *A Study and Application of Online Help for Novice Users.* Masters thesis, Texas Tech University.

Wittgenstein, Ludwig. 1953. *Philosophical investigations : the German text, with a revised English translation.* Blackwell Publishers.

Wobcke, Wayne, Ho, Van, Nguyen, Anh, & Krzywicki, Alfred. 2006. A BDI agent architecture for dialogue modelling and coordination in a smart personal assistant. *Pages 61–66 of : Proceedings of the 2005 NICTA-HCSNet Multimodal User Interaction Workshop - Volume 57.* Sydney, Australia : Australian Computer Society, Inc.

Wollermann, Charlotte. 2004. *Evaluierung der linguistischen Fähigkeiten von Chatbots.* Masters thesis, Rheinische-Friedrich-Wilhelms Universität Bonn.

Wollermann, Charlotte. 2006 (Sept.). Position paper. *Pages 75–76 of : Proceedings of Young Researchers' Roundtable on Spoken Dialogue Systems.*

Wærn, Yvonne. 1990. *Cognitive aspects of computer supported tasks.* New York, NY, USA : John Wiley & Sons, Inc.

Xiao, Jun, Stasko, John, & Catrambone, Richard. 2004. An Empirical Study of the Effect of Agent Competence on User Performance and Perception. *Pages 178–185 of : Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1.* New York, New York : IEEE Computer Society.

Xuetao, Mao, Sansonnet, Jean-Paul, & Bouchet, François. 2008 (Nov.). A corpus-based NLP-chain for a web-based Assisting Conversational Agent. *Pages 41–49 of : Actes du Troisième Workshop sur les Agents Conversationnels Animés (WACA 2008).*

Xuetao, Mao, Bouchet, François, & Sansonnet, Jean-Paul. 2009a. Impact of agent's answers variability on its believability and human-likeness and consequent chatbot improvements. *Pages 31–36 of :* Michaelson, Greg, & Aylett, Ruth (eds), *Proc. of the Symposium Killer Robots vs Friendly Fridges – The Social Understanding of Artificial Intelligence (AISB 2009).* Edinburgh, Scotland : SSAISB.

Xuetao, Mao, Sansonnet, Jean-Paul, & Bouchet, François. 2009b. Intelligent Assisting Conversational Agents viewed through Novice Users' Requests. *Pages 37–44 of :* Blashki, Katherine (ed), *Proc. of Interfaces and Human-Computer Interaction 2009.* Algarve, Portugal : IADIS Press.

Annexe A

Corpus *Daft*

Le corpus *Daft* étant constitué de 11 626 requêtes, il est trop important pour figurer même en annexe. De manière à donner tout de même un aperçu conséquent de celui-ci (en complément des exemples particuliers des tables 3.4, 3.5, 3.7, 3.8 4.1 et 7.3), nous présentons donc dans cette annexe une partie (le début, le milieu et la fin) du sous-ensemble de 1 074 requêtes utilisé pour les études menées dans le chapitre 7. Les phrases sont ordonnées par longueur croissante, puis par ordre alphabétique pour les phrases de même longueur.
Ce sous-ensemble est disponible dans son intégralité à l'adresse suivante :
http://fbouchet.vorty.net/projects/daft/corpus.1074.txt
Un autre sous-ensemble de 1 000 phrases sélectionnées aléatoirement est en accès libre sur le site de GRASP : http://www.limsi.fr/~jps/research/grasp/corpus/rawcorpus.1000.txt

Début

1	OK	ok	bon	Bye	bye !
Bon.	debut	joues	Bouge	ca va	merci
Marco	ok ok	bouge.	escape	Retour	hello !
Membres	Arrete !	membres	end svp	marco ?	ah bon ?
Ciao...	super !	arrête !	Augmente	Pouquoi ?	ou es tu
c'est ça	où es-tu	surement	parfait !	Au revoir	bien sur !
ferme la !	la ferme !	Ta gueule	vas t'en !	plus grand	éteind toi
montre toi	recommence	mon pauvre	ciao bella	au revoir !	Absolument
bien súr !	la ferme !	Tais toi !	que dis tu	au revoirq	j'approuve
très juste	description	que fais tu	ok pour moi	cest pas ça	non alors !
qui tu es ?	vraiement ?	modifie Jean	recommence !	enlever Bill	informations
excuse toi !	faut y aller	j'y suis pas	non ! annule	bon â rien !	es-tu prof ?
bon à rien !	Certainement	Carte du site	trier par nom	qui est Haifa	ca marche :-)
pas d'accord !	qui est tu ?	Que fais-tu ?	auf viedersen	ça me stresse	j'ai un doute
je proteste !	je t'aime pas	tu comprends ?	enlever "DAFT"	Change d'image	modifie Bellik
c'est quoi ça ?	je suis perdue	c'est quoi ça,	c'est probable	pas vrai lea ?	je veux partir
Je suis triste	merci, de rien	certainement !	pas question !	comment vas tu	que me veux tu
ça va pas non ?	c'est pas bien	Qui t'a fait ?	salut et merci	tu fais erreur	tu t'es trompé
nouvelle partie	retour au debut	joue à ma place	A quoi sers-tu ?	peux-tu m'aider	réponds-moi stp
ca m'etonnerait	dis moi ton âge	As-tu des amis ?	il est lisible,	j'y crois pas !	nouvel arrivant

pas trop mal...
c quoi le GT ACA
on peut tricher ?
je suis contre !
changer de langue
reinitialises toi
a quoi sert "raz"
sais-tu chanter ?
je n'y crois pas !
modifier Sansonnet
il y a quelqu'un ?
je suis Mr. Martin
explique moi le jeu
On peut décompter ?
ça se complique ...
ça tu peux le dire !
Je suis très triste
tu parles anglais ?
Arrête le compteur !
la vitesse de quoi ?
C'est pas très utile
as tu bien compris ?
Incrémente le chiffre
fiche le camp d'ici !
je me nomme Sansonnet
redemarre le compteur !
Comment marche ce jeu ?
etes vous un serveur ?
do you speak english ?
je refuse de te parler
les boutons du milieu ?
Ca veut dire quoi RAZ ?
le fond est trop foncé
je hais tes remarques !
puis-je compter su toi ?
j'aime bien ta figure !
tu répond toujours mal !
je veux trier par bureau
comment devenir membre ?
c'est totalement faux !
bon je dois filer lea...
tu l'aimes bien lea ? :)
Bouger de gauche a droite
Ca veut dire quoi rouge ?
cette action est un échec
racontre moi une histoire
fais une vérif du site svp
place Sansonnet en premier
pourquoi tout cet espace ?
j'aime pas que tu fasse ça
je me présente : Guillaume
c'est pas comme je pensais
remets Jean Pierre Fournier
revenons au sujet précédent
je me suis trompé de bouton
sais tu finir toute seule ?
la page de Pointal plante !
mignonne... mais efficace ?
bonjour je suis un étudiant
est-ce que tu sais parler ?
ne me force pas à te tuer !
aller a la page de Sansonnet
Compte a l'envers maitnenant
quels sont les participants ?
heyy qu'est ce que tu fais ?
non pas çui ci plutot cui la
t'as rien compris mon vieux !
J veux fermer ça
C'est quoi LEA ?
Ou des blagues ?
tu dors ou quoi ?
enlever "vernier"
Combien de themes
je me suis trompé
à quoi penses tu ?
je suis très déçu
c'est toi qui joue
J'adore ton accent
t'es fou ou quoi ?
modifier Sansonnet
que signifie le 1 ?
ca n'a pas marché !
c'est de ta faute !
je te crois pas pas
ajoute un membre stp
met Sansonnet en fin
pourquoi "faible" ?
c'est une première !
c'est la dactylo ...
ah, quel est ce jeux ?
Bon je me casse. Bye.
que penses tu de moi ?
j'ai peur de mal faire
peux tu me conseiller ?
as tu une conscience ?
je dois te laisser lea
juste un peu moins vite
ne me laisse pas tomber
c'est quoi, le GT ACA ?
qu'elle soit fructueuse
es tu bien sur de toi ?
bon on tourne en rond !
non, met plutot "pas" !
Arrête le compteur, stp.
Montre moi le logo LIMSI
Pourquoi tu le fais pas ?
quelle excellente idée !
est-ce que tout va bien
tu ne me mentirais pas ?
dépace de gauche à droite
as tu une autre solution ?
c quoi cette histoire ? ? ?
Ton look laisse à désirer
revenir à la première page
va à la page de son projet
souhaite tu que je t'aide ?
je ne te le fais pas dire !
je suis Jean-Pierre Martin
déplacer de droite a gauche
bouge gauche vers la droite
comment faire pour gagner ?
quel est le chef du groupe ?
Comment utilise t on ce jeu
le curseur est devenu fou !
pourquoi tu m'écoutes pas ?
je peux pas être d'accord !
salut Léa, à quoi sers tu ?
on ne peux pas aller dehors
clicker sur le drapeau rouge
je désire quitter maintenant
qui incrémente le compteur ?
je sais pas revenir au début
puis-je te faire confiance ?
y'a un mélange de franglais.
partie neuve svp
peut-on tricher ?
A quoi sers-tu ?
tu peux écrire ?
lance le compteur
je peux arrêter ?
Beaucoup de vide.
quel est ton but ?
n'est tu pas fou ?
pourquoi teletron ?
Qui t'a programmé ?
t'es quoi en fait ?
montre moi la carte
je suis perdu .....
as tu peur de moi ?
comment t'appele tu
je te laisse, salut
Désigne moi un objet
Trie par profession.
Qui est J-C Martin ?
qu'est ce que tu es ?
redemerre le compteur
coco est un tableau ?
hello comment ça va ?
je veux une autre page
peux tu re accélerer ?
Qu'est ce qu'une page ?
ce jeu me laisse froid
je m'appelle Jean-Paul
c est quoi le GT ACA ?
qu'est-ce que t'as dit ?
peux tu changer d'etat ?
racontre une histoire !
aime tu les étudiants ?
c'est pas très probable
pourquoi es tu navrée ?
montre moi le bouton raz
Comment devenir membre ?
qu'est ce que c'est aca ?
et si on se disait vous ?
j'en sais rien ma vielle
actionne lebouton du haut
bouger de droite a gauche
Le sigle AMI, c'est quoi ?
je n'ai pas besoin d'aide
tu es plus fort que moi !
je veux resizer la fenetre
a quoi sert cette fenêtre,
toi, tu ferais quoi là... ?
je n'ai plus besoin d'aide
Bonjour, quel est ton nom ?
Je veux revenir a l'accueil
deplacer de droite a gauche
existe t il un truc rouge ?
qu'est ce qui est en haut ?
Pourquoi tu t'appelle coco ?
comment je m'appelle hein ?
bonjour je suis un etudiant
je t'ai pas compris du tout
je suis Jean-paul Sansonnet
peut on te faire confiance ?
fais défiler l'ascenseur stp
page de Jean-pierre Fournier
c'est possible de s'abonner ?
où est-ce qu'on peut aller ?
mais un utilisateur-lambda ?
appuies sur le bouton quitter
voir les membres
Peux tu m'aider ?
c'est une erreur
tu peux marcher ?
page de sansonnet
qu'est-il arrivé ?
je suis en retard
Haaaa.... Merci.
Tu es très drôle.
ça va pas trop mal
ça c'est pas sûr !
effacer "jean-paul"
revenir a l'accueil
qui est Stéphanie ?
barre toi mon vieux
j'admire ton savoir
tu me suis ou pas ?
donne moi le premier
C'est quoi une page ?
qu'est ce qu etu es ?
que pense tu de moi ?
atteind ton maximum !
c'est pas de chance !
je suis Mr. Sansonnet
montre toi mon vieux !
peux tu t'en charger ?
je m'appelle Sansonnet
c'est à toi de décider
je ne te le dirais pas
je voudrais plus d'info
a quoi sert cette case ?
Que veut dire annuler ?
j'ai plus besoin de toi
comment t'appelles tu ?
et si on se disait tu ?
t'es à la limite chiant
je veux changer la ligne
A quoi sert la vitesse ?
le fond est un peu moche
as tu peur des humains ?
t'es dur de la feuille !
affiche la page Sansonnet
clickersur le bouton back
j'en suis pas trop sûr...
moi Tarzan ... toi Jane ?
tu n'es pas très fort....
montre moi le bouton "raz"
Bonjour,que puis je faire ?
ca ne va pas si mal que ça
Bonjour,que puis je faire ?
ça me convient tout à fait
montre moi la carte du site
enlever un membre au hasard
j'ai bien peur d'être perdu
rien n'empêche de tricher ?
qui fait partie du gt aca ?
j'ai pas de chance avec toi
salut Léa, à quoi sers tu ?
Léa, es tu contente de moi ?
je trouve que tu es inutile
t'es pas dans ton assiette ?
le disque de droite à gauche
réinitialise le compteur stp
C'est quoi la règle du jeu ?
je viens relever le compteur
répète un peu ça pour voir !
bon pi on reviens case depart

Milieu

Combien de doctorants dans ce groupe ?
puis-je actualiser le site moi-même ?
qu'est-ce qu'il y a sur ton tableau ?
Cette page a l'air trés intéressante.
as tu bien entendu ce que j'aid dit ?
il n'y a qu'a poser le dernier disque
change la couleur du fond du compteur,
Donne moi le numero entier pour Sarkis
je sais pas si il y a ou pas des liens
soit plus précis sur les règles du jeu
excuses moi, peux-tu me ramener page 1
sais tu si je peux gagner facilement ?
y aura til un jour une page de démos ?
là c'est ma faute, j'ai appuyé sur RAZ
est-ce qu'on pourrait pas se dire tu ?
est-ce que tu peux te mettre en jupe ?
de mon point de vue, t'es un peu niais
il faut que je vois la page des projets
ca change le pas du compteur c'est ca ?
Je voudrai découvrir ce qui est proposé
quelles sont les couleurs disponibles ?
comment faire pour arreter le compteur ?
quelles sont les propriétés de la page ?
y a t il une limite de temps pour jouer
Je souhaite faire un don cognitif à JPS
cette réponse ne m'emballe pas vraiment
tu n'aurais pas dû me parler sur ce ton
je voudrait aller vers la page des démos
explique moi un peu ce que je peux faire
peux tu grossir le texte dans la fenêtre
quels sont les éléments de l'application
combien y a t-il de liens dans la page ?
je peux pas changer la date du séminaire
le logo dans le coin du haut c'est quoi ?
sais tu ce qu'il faut faire maintenant ?
qu'arrive t-il si je te dis un gros mot ?
je ne peux pas accepter de tels propos !
t'as beau dire, je comprend toujours pas
je veux la documentation de l'application
Peux-tu dire a coco d'arreter de compter ?
le bouton RAZ marche pas comme il devrait
je suis en faveur d'une solution négociée
déplace le disque de gauche vers la droite
A quoi servent les boutons ok et quitter ?
elle est où exactement la page des demos ?
je peux changer le fond de l'application ?
que proposes tu pour résoudre le problème ?
je vois pas du tout pourquoi ça marche pas
on dirait que les liens ne sont pas actifs
si on le désire, on doit pouvoir tricher !
stp ne pourrait-on pas plutôt se vouvoyer ?
sors tes mains des poches, tu m'enerves ! !
je voudrais juste enlever la première ligne
dis moi ce qui arrive si j'appuie sur "RAZ"
je suis sûre que tu peux m'aider pour jouer
Peux tu changer la couleur du papier peint ?
comment faut faire pour arréter le compteur
est ce que je peux visualiser les projets ?
peut-on faire des mises-à-jour sur le site ?
je parie que je vais gagner du premier coup
maintenant je vais me débrouiller tout seul
as-tu des compétences en gestion de sites ?
il faut absolument arrêter ce compteur fou !
qu'est-ce qui ne va pas avec le gros disque ?
je pourrais modifier la table si je voulais ?
qu est-ce qui est écrit au dessus de "start"
c'est certain que le bouton "next" est buggé
la liste n'a pas été updatée correctement !
c'est très énéervant tout le temsp "externe"
je suppose que tu es contente de ta réponse ?
je trouve que le look des pages AMI est bien
Existe t il un formulaire d'acccueil ?
qu'est-ce que tu proposes maintenant ?
le bouton NEXT est probablement buggé
oui mais elle a toujours la meme voix
il est déconseillé de me contrarier !
préfère tu les garçons ou les files ?
prend celui du haut et met le à droite
Remet le compteur à zero s'il te plait
je voudrais savoir si il y a des leins
Comment s'appelle les élèments du jeu ?
peux tu déplacer les disques toi-même ?
Une page dédiée à l'accueil, pourquoi ?
c'est impossible d'arrêter le compteur
laisse moi au moins utiliser la souris
est-ce que tu peux bouger plus que ça ?
cool non ?ce n'est pas grand chose ...
je te promet que je vais m'appliquer !
Télécharger les autres exposés récents.
est-ce que Durand fait partie du GT ACA
ou est disponible la nouvelle version ?
as tu un tuyau pour gagner facilement ?
dis moi ce que je peux faire maintenant
qui a modifié la valeur de la vitesse ?
la table n'a pas été triée correctement
Ca ne te fais rien d'être une machine ?
Merci de m'avoir supporté si longtemps,
met Sansonnet juste en dessous de Bellik
aide moi a utiliser Coco le compteur stp
j'ai modifié la table sans faire expres !
quelle est la valeur du nombre du milieu
à ton avis, que veux dire le mot "coco" ?
euh... ca sert a quoi la scrollbar la ? ?
l'acces à certaines pages est-il limité ?
peux tu me dire si le GT ACA est ancien ?
c'est difficile de naviguer dans ce site
c'est fatiguant de taper des phrases ...
le design des pages laisse à désirer ...
ça te dirais de visiter la page de DAFT ?
peut-tu t'arreter de compter un instant ?
je ne sais pas si la page a des liens ...
je suis venu te dire que je m'en vais ...
veux-tu que je me déplaces dans le site ?
fais une sélection des pages intéressantes
au sujet de cette page, que peux tu dire ?
est-il possible des supprimer des lignes ?
peut tu me recommender une page à visiter ?
a mon avis, il manque un bouton de "reset"
j'imagine que tu connais les règles du jeu
pas moyen d'updater la liste des projets !
oui, je devrais envisager de te quitter...
je garde à l'esprit la remarque précédente
j'aurais voulu aller à la page de Sansonnet
...et qu'est-ce que ça veut dire "compte" ?
il n'y a rien à faire, je n'y arrive pas !
L'impression d'une page est-elle activable ?
quand on bouge le curseur, il compte quoi ?
comment je fais pour quitter le programme ?
le temps de la partie est-il limité ou pas ?
peut-on manipuler les disques à la souris ?
Le premier c'est celui strictement à gauche
je vous interdit formellement de me tutoyer
les rigolos dans ton genre me font pas peur
il faut que tu ailles à la page de Sansonnet
est-ce qu'on peut tricher un tout petit peu ?
pourquoi la page de Sansonnet est invisible ?
y a t-il beaucoup de fonctions disponibles ?
je ne suis pas à même de savoir les règles !
le bouton RAZ est probablement mal programmé
histoire de rendre cette fenêtre attractive !
Es tu jaloux de ma condition d'être humain ?
je vais essayer de me conformer à tes désirs

Fin

J'ai une question sur la vitesse à te poser : comment agit-elle sur Coco ?
peux tu me dire si les boutons "next" et "back" marchent en sens inverse
pourquoi le fond du tableau est-il noir alors que le compteur est gris ?
peux-tu rajouter d'autres éléments dans la fenêtre de Coco le compteur ?
il ne fait aucun doute que la page des membres n'est plus du tout à jour
il est tellement bête qu'il n'est même pas capable de dire quel disque !
il n'y a aucune chance de disposer d'un moyen de gérer la carte du site ?
d'après toi, y a t-il des fonctions d'annulation dans cette application ?
peux tu me rappeler comment on fait pour insérer une ligne dans une liste
ça serait quand même mieux si on pouvait se déplacer directement au début
il serait souhaitable que la fenêtre principale soit beaucoup plus grande
je ne suis pas du tout intéressé par tes explications un trop laborieuses
nous ne sommes pas du tout du même avis sur le fonctionnement du compteur
ffff... en tant qu'informaticien, je comprend ce que fait cette commande,
je ne partage pas du tout ton point de vue sur le look de la page de garde
le drame vient de ce que je n'ai pas réussi à changer le champ "profession"
j'aurai aimé que ça soit un peu plus rapide pour aller d'une page à l'autre
je suis très déçu qu'il n'y ai pas de commandes pour éditer toute les pages
il y a une ressemblance entre la photo de Jean-Paul et celle de Jean-Claude
cette page est très bien, j'irai même jusqu'à dire qu'elle est magnifique !
je ne suis pas sûr qu'il y ait un forum sur ce site, ou bien est-ce le cas ?
le drame vient de ce que je ne comprend pas comment la vitesse est contrôlée
je te serais reconnaissante si tu me disais quoi faire pour bouger un disque
par chance, y a t-il une possibilité de créer sa propre page et de l'ajouter
pourquoi le compteur refuse t-il de se mettre à zéro quand on fait "reset" ?
j'aime cette page mais je préférerais revenir à la page d'accueil si possible
si on change une ligne maintenant est-ce qu'on peut revenir en arrière après ?
je doute fort qu etu sois capable de modifier la police et la taille du texte
peux tu me dire à nouveau comment il faut faire pour remmettre le compteur à 0
je ne demande pas mieux que de jouer mais qu'elles sont excatement les règles ?
je ne vois pas l'avantage qu'i y a à utiliser la parole plutôt que les boutons
tu m'as permis de changer les listes et maintenant le site est tout détraqué !
j'approuve sans réserve l'idée de gérer le séminaire via une page web éditable
si j'étais toi, je resterais dans mon coin au lieu de toujours répondre a coté
selon moi, la page de Sansonnet est de très loin supérieure à celle de Pointal
de ton point de vue, y a t-il des bugs connus dans le fonctinnement du compteur
je ne sais pas comment faire pour changer le contenu d'un des champs de la table
je voudrais juste être assuré que les modifications de la table sont réversibles
le bouton "fermer" et le bouton "quitter" ont exactement le même fonctionnement ?
malgré mes efforts je n'arrive pas à touver la page correspondant au projet DAFT
il me semble que la page des membres ressemble fortement à celle des projets ...
j'aimerais pouvoir avoir des informations sur le petit logo bleu en haut à gauche
si tu ne veux pas faire ce que je te dis, très bien mais alors dis moi quoi faire
Etant donné que les pages des membres ne sont pas éditables, ya plus rien à faire
je m'oppose absolument à l'utilisation de popups pour effectuer les mises à jours
je voudrais m'expliquer : j'avais cru bien faire mais j'ai bouzillé le compteur !
oui j'ai cassé le compteur mais je t'assures que je ne pouvais pas faire autrement
si j'ai un conseil à te donner c'est de faire ce que je te dis au lieu de discuter
peux tu me dire pourquoi ça bippe quand je clicke sur premier lien en haut a gauche
on dirait que les boutons "raz" et "stop" ont pratiquement le même effet, pourquoi ?
si j'avais pu faire autrement, je t'assures que je n'aurais pas bouzillé le compteur
j'ai été vraiment surpris de constater qu'il manque une fonction d'annulation globale
Si seulement tu savais quoi faire pour restaurer la table comme elle était juste avant
je ne suis pas sûr qu'il y ait une page des démonstrations du groupe, ou c'est le cas ?
il a fait une faute d'orthographe dans "l'équipe de recherche qui a encadrée ma thèse"
je crois que la page des séminaire est loin d'être aussi jolie que la page des membres
je ne peux pas changer le champ "n^o de bureau" dans la liste des membres, comment faire ?
le site AMI est très bien dans l'ensemble mais certains détails mériteraient d'être revus
si je peux me permettre une suggestion, je crois qu'il faudrait jouer avec plus de disques
je pense que ce qui distingue le bouton "start" du bouton "reset" c'est la réinitialisation
il n'est guère probable qu'on puisse choisir les couleurs des disques sinon comment faire ?
Genre, changer la couleur de fond, la couleur des rondelles, la couleur des bandeaux, etc...
je ne savais pas qu'on avait le droit d'insérer ou bien de supprimer des lignes à cette table
Il me faut te poser cette question : est-tu un homme ou bien une femme ou autre chose encore ?
le petit le moyen et le gros ne sont pas bien discernables, surtout dispersés dans les trois tiges
est-ce que cest vraiment indispensable d'afficher une fenêtre popup pour modifier une simple ligne ?
La première constatation qui s'impose c'est qu'il manque cruellement d'information sur les fonctions
il est fort probable que tu ne pourra pas m'aider mais dis moi si il y a une fonction "new" pour les pages
peut-être que c'est déjà trop tard, mais je vais tout de même dire que le look du site AMI est très moche !
j'ai peur qu'il n'y ait pas moyen de changer la taille de la police de caractères qui est bien trop petite ...

Annexe B

Clés sémantiques ($\mathbb{K}_G$)

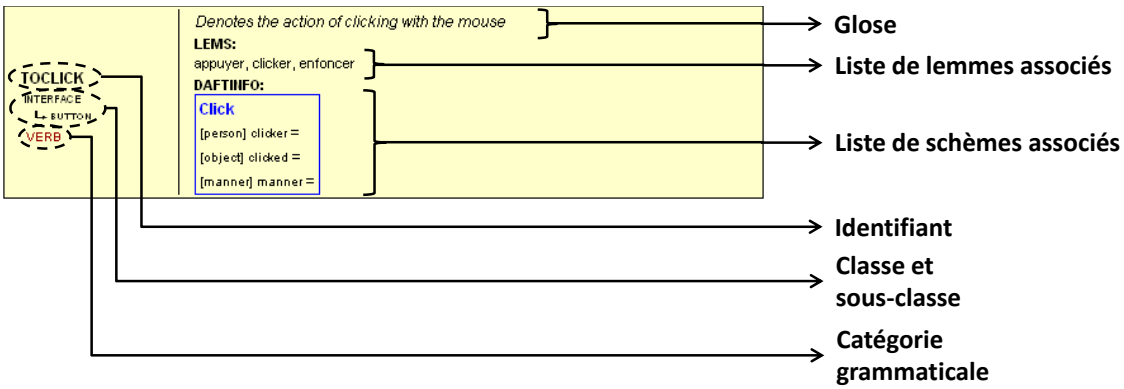
Les 1 294 clés formant l'ensemble $\mathbb{K}_G$ ont été converties au format XML et sont disponibles à l'adresse :

http://fbouchet.vorty.net/projects/daft/demo/DICO.keys.xml

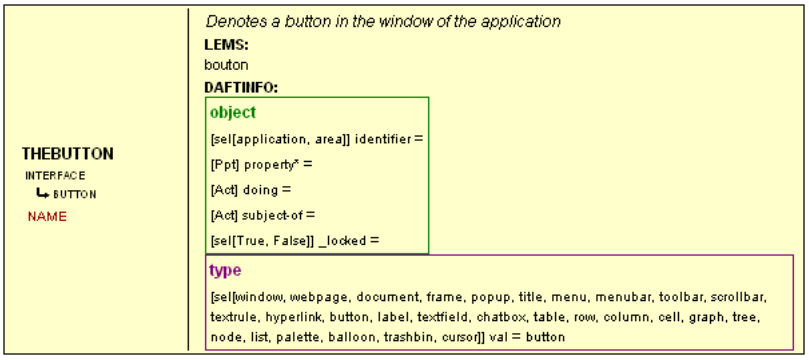
Ce fichier peut être visualisé sur la page :

http://fbouchet.vorty.net/projects/daft/demo/keys.html

Les clés y sont représentées graphiquement comme suit :



Quelques exemples de clés suivant ce format sont donnés ci-dessous :



THEINCREASE
PROCESS
↳ TUNING
NAME

Denotes the name of the action/the process of increasing the intensity of an attribute

LEMS:
augmentation, increment

DAFTINFO:

concept
[sel[game, step, example, rule, project]] identifier =
[Ppt] property* =

Modify
[{object, concept, person}] element =
[{manner}] manner =

manner
[sel[+, -]] val = +

[Ppt] property =

[{Act}] doing =

[Act] subject-of =
[sel[True, False]] _locked =

TOMODIFY
STRUCTURE
↳ MODIFYING
VERB

Denotes the action of modifying/altering something

LEMS:
adapter, affecter, alterer, changer, editer, mettre, modifier, remettre

DAFTINFO:

Modify
[{object, concept, person}] element =
[manner] manner =
[Ppt] property =

TOACCELERATE
PROCESS
↳ SPEED
VERB

Denotes the action of accelerating a procedure/a process

LEMS:
accelerer, depecher[SELF], foncer, hater, manier[SELF], presser[SELF]

DAFTINFO:

Modify
[{object, concept, person}] element =
[{manner}] manner =

manner
[sel[+, -]] val = +

[{speed}] property =

speed
[Interval[{-1, 1}]] val =

TOEXPLAIN
HELPING
↳ INFORMING
VERB

Denotes the action of explaining/teaching something to somebody

LEMS:
enseigner, expliquer

DAFTINFO:

TELL
[person] speaker =
[{person, object}] interlocutor =
[{object, concept, Act}] about =
[Act] todo =
[time-instant] time =
[Interval[{0, 1}]] insistance = 0.4

TODISAGREE
ARGUING
↳ FIGHTING
VERB

Denotes the mental action of disagreeing with somebody

LEMS:
attaquer, bouder, contester, contre, contredire, critiquer, desapprouver, engueuler, nier, protester, raler, rechigner, refuser, rejeter, reserve, reserver[PPL], resister, revolter[SELF]

DAFTINFO:

NEG
[{Act, object, concept}] of =

Accept
[{Act, concept}] accepted =

TOQUESTION
INTERACTING
↳ QUESTIONNING
VERB

Denotes the action of asking a question to somebody

LEMS:
demander[SELF], documenter[SELF], informer[SELF], interroger, poser, questionner, renseigner[SELF]

DAFTINFO:

INFOS
[Ref] about =
[sel[True, False]] more =

TOSURPRISE
OPINION
↳ REGULARITY
VERB

Denotes the action of surprising somebody or of being surprised

LEMS:
etonner, surprendre

DAFTINFO:

SURPRISE
[person] agent =
[object] about =

Annexe C

Entrées du lexique ($\mathbb{L}_G$)

Le lexique $\mathbb{L}_G$ est composé de 4 266 entrées selon le format défini en section 4.2.1.1. Elles sont réparties 9 dictionnaires $\mathbb{D}_G$:

– LEXIC est le dictionnaire principal (regroupant 85% des entrées) contenant le vocabulaire que l'on pourrait qualifier de générique dans le cadre de l'assistance à l'utilisation d'une application : nous donnons ci-dessous 30 entrées correspondant au début, au milieu et à la fin de ce dictionnaire (ordonné par ordre alphabétique).

– LOCUTION est le dictionnaire contenant les expressions constituées de plusieurs mots ne devant pas être séparés lors de la lemmatisation. Parmi les 385 entrées qu'il contient (9%), nous donnons ci-dessous également 12 entrées correspondant au début, au milieu et à la la fin de ce dictionnaire (ordonné par ordre alphabétique).

– ENGLISH et FOREIGN regroupent les expressions d'origines étrangères identifiées dans les requêtes (2% des entrées) : correspondant au début, au milieu et à la la fin de ENGLISH (ordonné par ordre alphabétique) sont présentées ci-dessous.

– WORDTEX, WEBSITE, HANOI, GTACA et COCO, dont les noms évoquent l'application (*cf.* chapitre 3) où les mots ont été collectés, contiennent des lemmes spécifiques au domaine de l'application assistée, et qui pourraient ne pas être chargés dans d'autre contexte. À titre d'exemple, nous donnons les 6 entrées du dictionnaire COCO.

```
{
  LEM["a"],
  POS[GG],
  FLEX["à"],
  KEYS[{"$A"}]
}

{
  LEM["aah"],
  POS[GG],
  FLEX["aah"],
  KEYS[{"TOSAYHAPPY"}]
}
{
  LEM["abaisser"],
  POS[VV],
  FLEX[CONJ["abaisser", 7]],
  KEYS[{"TODECREASE", "TOMOVEDOWN"}]
}

{
  LEM["abandonner"],
  POS[VV],
  FLEX[CONJ["abandonner", 7]],
  KEYS[{"TOABORT", "TOABANDON"}]
}
{
  LEM["abondamment"],
  POS[RR],
  FLEX["abondamment"],
  KEYS[{"QUANTLARGE"}]
}

...
{
  LEM["initialiser"],
  POS[VV],
  FLEX[CONJ["initialiser", 7]],
  KEYS[{"TORESET"}]
}
{
  LEM["initier"],
  POS[VV],
  FLEX[CONJ["initier", 16]],
  KEYS[{"ISCOMPETENT"[PPL], "TOSTART"}]
}
```

```
{
  LEM["abonnement"],
  POS[NN],
  GEND[M],
  FLEX[{MS["abonnement"], MP["abonnements"]}],
  KEYS[{"TOSUBSCRIBE"}]
}
{
  LEM["abonner"],
  POS[VV],
  FLEX[CONJ["abonner", 7]],
  KEYS[{"TOSUBSCRIBE"}],
}
{
  LEM["abord"],
  POS[RR],
  FLEX["abord"],
  DICOCOM["d'abord"],
  KEYS[{"NOKEY"}]
}
{
  LEM["aborder"],
  POS[VV],
  FLEX[CONJ["aborder", 7]],
  KEYS[{"TOFOCUSON"}],
}
{
  LEM["aboutir"],
  POS[VV],
  FLEX[CONJ["aboutir", 20]],
  KEYS[{"TOSUCCEED", "TOFINISH", "TORESULTIN"}]
}

...
{
  LEM["inquieter"],
  POS[VV],
  FLEX[CONJ["inquiéter",11], CONJ["inquièter",11]],
  KEYS[{"TOWORRY"[SELF]}]
}
{
  LEM["inquietude"],
  POS[NN],
  GEND[F],
  FLEX[{FS["inquiétude"], FP["inquiétudes"]}]
}
```

```
{
  LEM["inoperant"],
  FLEX[{MS["inopérant"], MP["inopérants"],
        FS["inopérante"], FP["inopérantes"]}],
  POS[JJ],
  KEYS[{"ISNOTOPERATIONAL"}]
}
{
  LEM["inopinement"],
  POS[RR],
  FLEX["inopinément"],
  KEYS[{"ISSUDDEN"}]
}
{
  LEM["inquiet"],
  FLEX[{MS["inquiet"], MP["inquiets"],
        FS["inquiète"], FP["inquiètes"]}],
  POS[JJ],
  KEYS[{"ISWORRIED"}]
}

...
{
  LEM["zapper"],
  POS[VV],
  FLEX[CONJ["zapper", 7]],
  KEYS[{"TOSKIP"}]
}
{
  LEM["zazou"],
  FLEX[{MFS["zazou"], MFP["zazous"]}],
  POS[JJ],
  KEYS[{"ISINCOMPETENT"}]
}
{
  LEM["zero"],
  POS[CD],
  FLEX[{MS["zéro"], MP["zéros"]}],
  KEYS[{"AKOCARDINAL"}]
}
{
  LEM["zinzin"],
  FLEX[{MFS["zinzin"], MFP["zinzins"]}],
  POS[JJ],
  KEYS[{"ISINCOMPETENT"}]
}
{
  LEM["zone"],
  POS[NN],
  GEND[M],
  FLEX[{MS["zone"], MP["zones"]}],
  KEYS[{"THESURFACE"}]
}
```

```
{
  LEM["inscrire"],
  POS[VV],
  FLEX[CONJ["inscrire", 86]],
  KEYS[{"TOWRITE", "TOSUBSCRIBE"[SELF]}]
}

{
  LEM["insecable"],
  FLEX[{MFS["insécable"], MFP["insécables"]}],
  POS[JJ],
  KEYS[{"ISINDIVIDABLE"}]
}
{
  LEM["inserer"],
  POS[VV],
  FLEX[CONJ["insérer",11], CONJ["insèrer",11]],
  KEYS[{"TOFILL"}]
}

...
{
  LEM["zoomer"],
  POS[VV],
  FLEX[CONJ["zoomer", 7]],
  KEYS[{"TORESIZE"}]
}
{
  LEM["zut"],
  POS[GG],
  FLEX["zut"],
  KEYS[{"TOSAYOUPS", "TOSAYINSULT"}]
}
{
  LEM["$excl$"],
  POS[KK],
  FLEX["$excl$"],
  KEYS[{"AKOPUNCTUATION"}]
}
{
  LEM["$intg$"],
  POS[KK],
  FLEX["$intg$"],
  KEYS[{"QUEST"}]
}
{
  LEM["$sadey$"],
  POS[KK],
  FLEX["$sadey$"],
  KEYS[{"TOSAYSORRY"}]
}
```

LOCUTION

```
{
  LEM[".a.bientot"],
  LOC[{"a", "bientot"}],
  POS[GG],
  FLEX["-"],
  DICO[LOCUTION],
  KEYS[{"TOSAYBYE"}]
}
{
  LEM[".a.cause.de"],
  LOC[{"a", "cause", "de"}],
  POS[GG],
  FLEX["-"],
  DICO[LOCUTION],
  KEYS[{"BECAUSE"}]
}
...
{
  LEM[".le.meme.chose"],
  LOC[{"le", "meme", "chose"}],
  POS[GG],
  FLEX["-"],
  DICO[LOCUTION],
  KEYS[{"ISSAME"}]
}
{
  LEM[".le.inverse.le.un.de.le.autre"],
  LOC[{"le", "inverse", "le", "un", "de",
       "le", "autre"}],
  POS[JJ],
  FLEX["-"],
  DICO[LOCUTION],
  KEYS[{"ISCONTRARY"}]
}

...
{
  LEM[".valeur.limite"],
  LOC[{"valeur", "limite"}],
  POS[NN],
  GENDER[F],
  FLEX["-"],
  DICO[LOCUTION],
  KEYS[{"THEMAXIMUM"}]
}
{
  LEM[".vieux.branche"],
  LOC[{"vieux", "branche"}],
  POS[PP],
  FLEX["-"],
  DICO[LOCUTION],
  KEYS[{"ABOUTSYSTEM"}]
}
```

```
{
  LEM[".a.chaque.fois"],
  LOC[{"a", "chaque", "fois"}],
  POS[GG],
  FLEX["-"],
  DICO[LOCUTION],
  KEYS[{"EACHTIME"}]
}
{
  LEM[".a.cote.de.le.plaque"],
  LOC[{"a", "cote", "de", "le", "plaque"}],
  POS[JJ],
  FLEX["-"],
  DICO[LOCUTION],
  KEYS[{"ISBEWILDERED"}]
}
...
{
  LEM[".le.moins"],
  LOC[{"le", "moins"}],
  POS[DD],
  FLEX["-"],
  DICO[LOCUTION],
  KEYS[{"THELEAST"}]
}
{
  LEM[".le.moins.que.on.pouvoir.dire"],
  LOC[{"le", "moins", "que", "on" || "je",
       "pouvoir", "dire"}],
  LOCLEM[".en.fait"],
  POS[GG],
  FLEX["-"],
  DICO[LOCUTION],
  KEYS[{"ACTUALLY"}]
}
...
{
  LEM[".vis.a.vis"],
  LOC[{"vis", "a", "vis"}],
  POS[GG],
  FLEX["-"],
  DICO[LOCUTION],
  KEYS[{"ABOUT", "ISPOSOPPOSITE"}]
}

{
  LEM[".week.end"],
  LOC[{"week", "end"}],
  POS[GG],
  FLEX["-"],
  DICO[LOCUTION],
  KEYS[{"SUNDAY"}]
}
```

ENGLISH

```
{
  LEM["abstract"],
  POS[NN],
  GEND[M],
  FLEX[{MS["abstract"], MP["abstracts"]}],
  DICO[ENGLISH],
  KEYS[{"THEOUTLINE"}]
}
{
  LEM["back"],
  POS[VV],
  FLEX[CONJ["back", 0]],
  DICO[ENGLISH],
  KEYS[{"TOGOBACK"}]
}

...
{
  LEM["next"],
  POS[VV],
  FLEX[CONJ["next", 0]],
  DICO[ENGLISH],
  KEYS[{"TOGONEXT"}]
}
{
  LEM["no"],
  POS[GG],
  FLEX["no"],
  DICO[ENGLISH],
  KEYS[{"TOSAYNO"}]
}

...
{
  LEM["week"],
  POS[NN],
  GEND[M],
  FLEX[{MS["week"], MP["weeks"]}],
  DICO[ENGLISH],
  KEYS[{"THEWEEK"}]
}
{
  LEM["what"],
  POS[GG],
  FLEX["what"],
  DICO[ENGLISH],
  KEYS[{"WHAT"}]
}
```

```
{
  LEM["background"],
  POS[NN],
  GEND[M],
  FLEX[{MS["background"], MP["backgrounds"]}],
  DICO[ENGLISH],
  KEYS[{"ISPOSREAR"}]
}
{
  LEM["backslash"],
  POS[NN],
  GEND[M],
  FLEX[{MS["backslash"], MP["backslashs"]}],
  DICO[ENGLISH],
  KEYS[{"AKOPUNCTUATION"}]
}

...
{
  LEM["nope"],
  POS[GG],
  FLEX["nope"],
  DICO[ENGLISH],
  KEYS[{"TOSAYNO"}]
}
{
  LEM["ok"],
  POS[GG],
  FLEX["ok"],
  DICO[ENGLISH],
  KEYS[{"TOSAYOK"}]
}

...
{
  LEM["yes"],
  POS[GG],
  FLEX["yes"],
  DICO[ENGLISH],
  KEYS[{"TOSAYOK"}]
}

{
  LEM["you"],
  POS[PP],
  FLEX["you"],
  DICO[ENGLISH],
  KEYS[{"THESYSTEM"}]
}
```

COCO

```
{
  LEM["arrosoir"],
  POS[NN],
  GEND[M],
  FLEX[{MS["arrosoir"], MP["arrosoirs"]}],
  DICO[COCO],
  KEYS[{"ADHOCCOCO"}]
}
{
  LEM["cafe"],
  POS[NN],
  GEND[M],
  FLEX[{MS["café"], MP["cafés"]}],
  DICO[COCO],
  KEYS[{"ADHOCCOCO"}]
}
{
  LEM["cafetiere"],
  POS[NN],
  GEND[F],
  FLEX[{FS["cafetière"], FP["cafetières"]}],
  DICO[COCO],
  KEYS[{"ADHOCCOCO"}]
}
{
  LEM["lampe"],
  POS[NN],
  GEND[F],
  FLEX[{FS["lampe"], FP["lampes"]}],
  DICO[COCO],
  KEYS[{"ADHOCCOCO"}]
}
{
  LEM["planificateur"],
  POS[NN],
  GEND[M],
  FLEX[{MS["planificateur"], MP["planificateurs"]}],
  DICO[COCO],
  KEYS[{"ADHOCCOCO"}]
}
{
  LEM["radiateur"],
  POS[NN],
  GEND[M],
  FLEX[{MS["radiateur"], MP["radiateurs"]}],
  DICO[COCO],
  KEYS[{"ADHOCCOCO"}]
}
```

Annexe D

Règles de réécriture de GRASP ($\mathbb{R}_G$)

On a vu dans la section 4.4.2 que GRASP dispose d'un ensemble de 240 règles $\mathbb{R}_G$ permettant l'analyse grammaticale des requêtes. Cette annexe regroupe donc l'ensemble des entêtes de règles, c'est-à-dire l'identifiant de la règle et le schéma de nœuds déclenchant la réaction associée à la règle. La réaction n'est toutefois pas donnée ici.

Au sein de ces règles, on fait appel aux sous-ensembles de lemmes suivants qui peuvent être appelés à différentes reprises, ce qui permet donc d'alléger la notation :

ABOUTVERBSPPL = associer|lier|relier|dedier|consacrer
ABOUTVERBSPPZ = correspondre
ADHOCLABEL = echap|enter|entrée|entree|ok|speed|start|stop|stopper|raz|quitter|fermer|compter|back|next|annuler|help|new|home|bad|reset|retour|top|undo|ami|stoppe|on|verbatim|manuel|automatique|survol|page|mathematique|plan|revision|echo|normal|manuel|auto|texte|graphique|lecture|commande|qwerty
GLfromtoverb = amener|bouger|decaler|demenager|deplacer|draguer|emmener|glisser|mouvoir|porter|transferer|transporter
GLlocPPZ = occuper|resider|exister|trouver
GLlocverb = abaisser|afficher|ajouter|aligner|aller|amener|apparaitre|approcher|arriver|baisser|bouger|centrer|coller|creer|decaler|definir|demenager|deplacer|deposer|descendre|disposer|distribuer|draguer|ecrire|egarer|elever|eloigner|enlever|emmener|etaler|faire|glisser|hausser|inserer|installer|ajouter installer|jouer|justifier|lever|loger|marquer|mettre|monter|mouvoir|nager|orienter|parvenir|placer|porter|poser|positionner|ranger|rapprocher|recentrer|rechercher|remonter|remplacer|renforcer|repartirPLACE|replacer|reporter|repousser|resserrer|retrouver|revenir|situer|soulever|superposer|taper|transferer|transporter
LABELPRED = bouton|touche|commande|champ|case|logo|fonction|action|mode|symbole|mot|version|affichage|outil
PPZTOJJ = etudier|flotter|suivre|assister|derouler|exister|courir|decroitre|croitre|exposer|entrer|defiler|correspondre
PREP = a|en|avec|par|pour|surG|deC|.ainsi.que|.a.le.aide.de|.a.le.dela|.a.le.lieu.de|a.le.oppose|.a.le.place.de|.a.le.suite.de|.a.partir.de|apres|.a.propos.de|.aupres.de|avant|cf|chez|contre|dans|dedans|dehors|depuis|derriere|dessous|dessus|devant|durant|.en.arriere|.en.cas.de|.en.deca|.en.fonction.de|.en.guise.de|.en.partant.de|.en.tant.que|entre|envers|.grace.a|inter|loin|.par.le.biais.de|parmi|pendant|pres|.relatif.a|selon|sous|.suite.a|.sur.le.base.de|vers|via|.sous.forme.de
PREPCONJ = ou1|et|.plutot.que|puis|.et.ou
PREPINF = de|pour|a|surG|.a.le.lieu.de|.a.part|.en.train.de|.en.vue.de|hormis|sans|.sans.jamais|.afin.de|aussi|.quand.meme|.si.possible|.tout.de.meme|.plutot.que

PREPNN2	=	entre|comme|puis
PRONAVOIR	=	il|on|tu|me|tt|lui|ca|ne|que|quel|lequel|qui
QUESTPRON	=	je|tu|il|on|ce
RCPA	=	semblable|similaire|comparable|egal|identique|superieur|inferieur
RFX1	=	tu|te|toi|ce|ca|lui
RFX2	=	je|me|moi|ce|ca|lui
RFX3	=	je|me|moi|tu|te|toi
RRGG	=	deja|encore|pourtant|souvent|non|peu|plutot|plus|meme|a.le.rigueur|voire
VERBDEFINE	=	preciser|definir|detailler|justifier|expliquer|agrandir
VERBSTATE	=	etre|paraitre|apparaitre|sembler|rester|devenir|avoirlair
VERBSTATERFX	=	trouver|retrouver|voir|sentir|averer

N^o	Identifiant	Schéma de nœuds
1	maisquoi1	{a :mais,b :quoi,q :$intg$,y :BB}
2	maisquoi2	{a :et,b :toi|moi,q :$intg$,y :BB}
3	non1	{x :non|.ne.etre.ce.pas,q :$intg$,y :BB}
4	oui1	{x :oui|ok,q :$intg$,y :BB}
5	oui2	{x :oui|ok,q :$excl$,y :BB}
6	?1	{x :$intg$,y :BB}
7	!1	{x :$excl$,y :BB}
8	susp1	{x :$susp$,y :BB}
9	smiley1	{x :$smiley$,y :BB}
10	hein1	{x :hein,y :BB}
11	pourquoi1	{x :pourquoi,y :BB}
12	abientot1	{x :.a.bientot,y :BB}
13	salut1	{u :!(DD|PP),x :salut,y :BB}
14	dot1	{x :$sent$|$virg$,y :BB}
15	foule1	{x :un,y :foule,z :de}
16	partie1	{x :un,y :FLEX[partie],z :de}
17	loin1	{x :loin,y :de,z :valoir|etre}
18	loin2	{v :etre,SPAN[s,3],x :de,y :loin}
19	loin3	{x :loin,y :de}
20	en1	{x :en,y :haut|bas}
21	stp1	{x :stp}
22	stp2	{x :.par.exemple}
23	air1	{x :avec,y :ton,z :air}
24	air2	{x :donner,y :le,z :air}
25	air3	{x :de,y :le,z :air}
26	air4	{x :de,y :air}
27	de1	{p :de,c :CD}
28	tel1	{x :de,y :tel,z :!que}
29	quelle1	{x :|,y :que,z :FLEX[elle|elles]}
30	peutetre1	{z :GG|VV|KK|BB|RR,x :FLEX[peut],y :etre}
31	detonpart1	{x :de,y :ton,z :part|partir}
32	onsediretu1	{p1 :je|tu|il|on,OPT[z1,ne],p2 :te|me|se,v :dire,OPT[z2,pas|plus],p3 :tu}
33	a1	{x :FLEX[a|à],y :mon,z :avis}
34	a2	{x :FLEX[a|à],y :ton|son,z :avis}
35	PPLa1	{x :ALT[ABOUTVERBSPPL]&&TENSE[PPL],y :a}
36	PPZa1	{x :ALT[ABOUTVERBSPPZ]&&TENSE[PPZ],y :a}
37	concernant1	{x :concerner&&TENSE[PPZ]}
38	a3	{y :!VV,x :ALT[PRONAVOIR],a :a&&!FLEX[à]}
39	a4	{x :que,a :a&&!FLEX[à],p :PP}
40	a5	{x :y,a :a&&!FLEX[à]}
41	a6	{a :a&&!FLEX[à],y :TENSE[PPL|APPL]}
42	a7	{a :a&&!FLEX[à],y :tu}
43	a8	{a :a}
44	tt1	{t :tt,v :VV,x :tu}
45	tt2	{x :(VV|PP)||(de|qui|ne),t :tt}
46	tt3	{t :tt}
47	meme1	{d :moi|je,x :meme,y :!(si|pour)}
48	meme2	{d :toi|tu,x :) | meme,y :¬(pour si}
49	meme3	{d :lui|il,x :meme,y :!(si|pour)}
50	meme4	{d :soi,x :meme,y :!(si|pour)}

N^o	Identifiant	Schéma de nœuds
51	meme5	{d :DD,x :meme,y :que}
52	meme6	{d :DD,x :meme}
53	meme7	{x :meme}
54	tout1	{x :tout,v :TENSE[PPL]}
55	tout2	{x :tout,v :VV&& !TENSE[PPL]}
56	tout3	{x :tout,y :NN|DD}
57	tout4	{x :tout,y :RR|JJ}
58	tout5	{x :tout,y :en}
59	certain1	{x :certain,y :RR|JJ|NN}
60	certain2	{x :certain}
61	deletout1	{x :plus,SPAN[s,5],y :de,z :le,u :tout}
62	plus1	{x :ne,FREE[s,pas,5],y :plus}
63	plus2	{x :VV&& !ALT[VERBSTATE]&& !ALT[VERBDEFINE],y :plus,z : !de}
64	plus3	{x :plus,y :de}
65	jamais1	{x :jamais}
66	moinsde1	{x :moins,y :de}
67	si1	{x :si,y :seulement}
68	si2	{x :si,v :TENSE[PPZ]}
69	si3	{x :si,r :RR|JJ}
70	aussi1	{x :aussi,r :RR|JJ,SPAN[s,5],y :que}
71	aussi2	{x :aussi&& !FLAG[ASMUCH],v :TENSE[PPZ]}
72	aussi3	{x :aussi&& !FLAG[ASMUCH],r :RR|JJ}
73	aussi4	{x :aussi&& !FLAG[ASMUCH]}
74	no1	{x :no,y :NN|JJ|RR}
75	celui1	{x :celui}
76	cedont1	{x :ce,y :dont}
77	ce1	{x : !etre,y :ce,z :NN|JJ}
78	PRCLIT1	{x :le|leur|en,v :VV&& !TENSE[PPZ]}
79	deletout2	{x :pas,SPAN[s,5],y :de,z :le,u :tout}
80	pas-tout1	{x :pas,y :tout|encore}
81	pas-rien1	{x :ne,y :rien}
82	surtout1	{x :rien|surtout|meme|toujours,y :ne|pas}
83	combien1	{x :dans|depuis,y :combien}
84	RCP1	{x :ALT[RCPA]||FLEX[ressemblant],a :a}
85	RCP2	{x :pareil,y :que}
86	RCP3	{x :aussi|plus|moins,j :JJ,y :que}
87	RCP4	{x :comme}
88	Rc1	{r1 :RR,r2 :RR,r3 :RR,r4 :RR}
89	Rc2	{r1 :RR,r2 :RR,r3 :RR}
90	Rc3	{r1 :RR,r2 :RR}
91	Rrrcp1	{r :RR,x :RCP}
92	Rgj1	{r :ALT[RRGG],j :JJ}
93	Rrpasr1	{r1 :RR||ALT[RRGG],x :pas,r2 :RR}
94	Rpasr1	{x :pas,r :RR||ALT[RRGG]}
95	Rvpr1	{v :VV,p :je|tu|il|on|moi|toi|lui,r :RR||ALT[RRGG]}
96	Rvrv1	{v1 :VV,r :RR||ALT[RRGG],v2 :VV}
97	Rrv1	{r :RR||ALT[RRGG],v :VV}
98	Rvr1	{v :VV,r :RR||ALT[RRGG]}
99	RsiR1	{g :GG&& !(de|que|si|aussi),r :RR||ALT[RRGG]}
100	Rrj1	{r :RR,j :JJ}
101	ENFAIT1	{v :VV,SPAN[s,5],g :.en.fait}
102	ENFAIT2	{g :.en.fait,SPAN[s,5],v :VV}
103	ENFAIT3	{x :.en.fait}
104	Nce1	{x :etre,y :ce}
105	PPZ1	{x :NN|JJ|DD,z :TENSE[PPZ]&&ALT[PPZTOJJ]}
106	NE1	{t :monsieur|madame|mademoiselle,e1 :NE,e2 :NE}
107	NE2	{t :monsieur|madame|mademoiselle,e :NE}
108	NE3	{e1 :NE,e2 :NE}
109	NEetouNE1	{e1 :NE,p :ALT[PREPCONJ],e2 :NE}
110	NEni1	{x :ni,e1 :NE,y :ni,e2 :NE}
111	NEsoit1	{x :soit,e1 :NE,y :soit,e2 :NE}
112	CD1	{x :numero,c :CD}
113	CD2	{c1 :CD,rel :ALT[PREPCONJ],c2 :CD}
114	CDni1	{x :ni,c1 :CD,y :ni,c2 :CD}
115	CDsoit1	{x :soit,c1 :CD,y :soit,c2 :CD}

N^o	Identifiant	Schéma de nœuds
116	DET1	{d1 :DD,ou1,d2 :DD}
117	DET2	{d :un,n :NN,x :de,y :RR}
118	DET3	{d1 :un,x :de,d2 :le,z :JJ|NN|RR}
119	DET4	{d1 :DD,d2 :un|autre|moindre|meme}
120	DET5	{d :DD,c :CD,z :JJ|NN|RR}
121	DET6	{q :peu|beaucoup|nombre|certain|seulement|$subpart$,x :de,d :DD|CD,z :JJ|NN|RR}
122	DET7	{q :.un.certain.nombre|peu|beaucoup|nombre|plus|moins|autant|$subpart$,x :de, z :JJ|NN|RR}
123	DET8	{q :pas,x :de,z :JJ|NN|RR}
124	DET9	{q :tout|seulement|seul|juste|excepte|.a.part|outreG|sans|sauf|.ne.importer.quel|meme, d :DD|CD,z :JJ|NN|RR}
125	DET10	{q :certain,z :JJ|NN|RR}
126	PPL1	{x :DD|NN,v :TENSE[PPL]}
127	Jpas1	{x :(DD|JJ|NN)||(et|ou1),y :non|pas,j :JJ}
128	JJetouJ1	{j1 :JJ,j2 :JJ,rel :ALT[PREPCONJ],j3 :JJ}
129	JJJ1	{j1 :JJ,j2 :JJ,j3 :JJ}
130	JJ1	{j1 :JJ,j2 :JJ}
131	JetouJ1	{j1 :JJ,rel :ALT[PREPCONJ],j2 :JJ}
132	Jni1	{x :ni,j1 :JJ,y :ni,j2 :JJ}
133	Jsoit1	{x :soit,j1 :JJ,y :soit,j2 :JJ}
134	Nlabel1	{d :DD,n :ALT[LABELPRED],lab1 :FLEX[ALT[ADHOCLABEL]]||QQ, rel :ALT[PREPCONJ],lab2 :FLEX[ALT[ADHOCLABEL]]||QQ}
135	Nlabel2	{d :DD,n :ALT[LABELPRED],lab :FLEX[ALT[ADHOCLABEL]]||QQ}
136	Nlabel3	{d :le,n :mot,lab :!(qui|que|de|)||QQ}
137	Ndjnj1	{d :DD|CD,j1 :JJ,n :NN,j2 :JJ}
138	Ndjn1	{d :DD|CD,j :JJ,n :NN}
139	Ndnj1	{d :DD|CD,n :NN,j :JJ}
140	Ndnc1	{d :DD,n :NN,c :CD}
141	Ndne1	{d :DD|CD,n :NN|NE|QQ}
142	Ndc1	{d :DD,c :CD}
143	Ndj1	{d :DD|CD,j :JJ}
144	Nnj1	{n :NN,j :JJ}
145	Njn1	{j :JJ,n_/ ;§GNis[n,NN]&& !§GNhas[n,DET]}
146	Nnc1	{n :NN,c :CD}
147	Nne1	{n :NN,e :NE|QQ}
148	Nen1	{e :NE|QQ,n :NN|QQ}
149	Nfin1	{x :NE|QQ|DD}
150	NetouN1	{n1 :NN,rel :ALT[PREPCONJ],n2 :NN}
151	Nni1	{x :ni,n1 :NN,y :ni,n2 :NN}
152	Nsoit1	{x :soit,n1 :NN,y :soit,n2 :NN}
153	QNEG1	{x :ne,p1 :PP,v :VV,p2 :ALT[QUESTPRON],y :pas|plus}
154	QNEG2	{x :ne,v :VV,p :ALT[QUESTPRON],y :pas|plus}
155	Qa1	{a :a,q :quoi,v :VV,p :ALT[QUESTPRON]&& !QUEST[INVERSION]}
156	Qb1	{a :a,q :quoi,p :ALT[QUESTPRON]&& !QUEST[INVERSION],v :VV}
157	Qc1	{q :que,v :VV,p :ALT[QUESTPRON]&& !QUEST[INVERSION]}
158	Qd1	{x :te|me|se|y,v :VV,p :ALT[QUESTPRON]&& !QUEST[INVERSION]}
159	Qe1	{n :NN,v :VV,p :(tu|il)&& !QUEST[INVERSION]}
160	Qf1	{x :!PP&& !sans,v :VV,p :ALT[QUESTPRON]&& !QUEST[INVERSION]}
161	nepasa1	{x :ne,y :pas,z :PP,v :VV}
162	nepasb1	{x :ne,y :pas,v :VV}
163	nepasc1	{x :ne,p1 :PP,p2 :PP,v :VV,y :pas}
164	nepasd1	{x :ne,p :PP,v :VV,y :pas}
165	nepase1	{x :ne,v :VV,y :pas}
166	aucun1	{x :aucun,z :de,n :NN,SPAN[s,5],y :ne,v :VV}
167	aucun2	{x :aucun,n :NN,SPAN[s,5],y :ne,v :VV}
168	aucun3	{x :aucun,y :de,n :NN,SPAN[s,5],v :VV}
169	aucun4	{x :aucun,n :NN,SPAN[s,5],v :VV}
170	ninine1	{n :NN&&FLAG[NINI],SPAN[s,2],y :ne,v :VV}
171	pas-PPV1	{x :pas,p1 :PP,p2 :PP,v :VV}
172	pas-PV1	{x :pas,p :PP,v :VV}
173	pas-Vpas1	{v :VV,x :pas}
174	pas-pasV1	{x :pas,v :VV}
175	ne1	{x :ne,p :PP,v :VV,y :rien|aucun|que|plus|guere|personne}
176	ne2	{x :ne,v :VV,y :rien|aucun|que|plus|guere|personne}
177	ne3	{x :ne,y :rien|plus|guere,v :VV}
178	ne4	{y :rien|aucun|personne|pas,x :ne,v :VV}

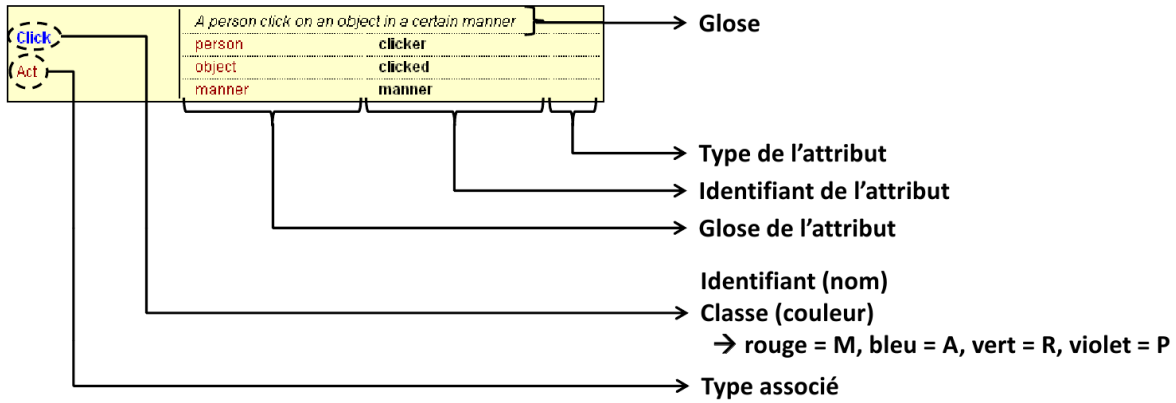
N^o	Identifiant	Schéma de nœuds
179	PAST1	{x :avoir,y :etre,v :TENSE[PPL|APPL]}
180	PAST2	{x :avoir,v :VV}
181	PPL2	{x :etre,v :TENSE[PPL|APPL]}
182	RFXa1	{p :je,FREE[s,ALT[RFX1],2],x :me|je,v :VV}
183	RFXb1	{p :tu,FREE[s,ALT[RFX2],2],x :te,v :VV}
184	RFXc1	{p :il|on|ca|comment,FREE[s,ALT[RFX3],2],x :se,v :VV}
185	RFXd1	{p1 :je,FREE[s,ALT[RFX1],2],x :me|je,p2 :PP,v :VV}
186	RFXe1	{p1 :tu,FREE[s,ALT[RFX2],2] (x :te),p2 :PP,v :VV}
187	RFXf1	{p1 :il|on|ca|comment,FREE[s,ALT[RFX3],2],x :se,p2 :PP,v :VV}
188	RFXg1	{x :POS[BB|KK],v :VV,p :toi}
189	RFXh1	{x :POS[BB|KK],p :te,v :NEG[]}
190	RFXi1	{p :se,v :VV}
191	GOINGTO1	{x :aller,p :PP,v :VV}
192	GOINGTO2	{x :aller,v :VV}
193	PRCLIT2	{p :te|me|se|y|le,v :VV}
194	air5	{x :avoir|a,FREE[s,de,2],y :le,z :air}
195	VERBSTATE1	{x :ALT[VERBSTATE],v :TENSE[PPL|APPL|PPZ]}
196	VERBSTATE2	{x :ALT[VERBSTATERFX]&&RFX[],v :TENSE[PPL|APPL|PPZ]}
197	VetouV1	{v :VV,x :ALT[PREPCONJ],d :de,w :VV}
198	VetouV2	{v :VV,x :ALT[PREPCONJ],w :VV}
199	Vsoit1	{x :soit,v1 :VV,y :soit,v2 :VV}
200	ETRECEQUE1	{q :que,x :FLEX[est|es],y :ce,z :que,n :NN|PP,v :VV}
201	ETRECEQUE2	{q :que,x :FLEX[est|es],y :ce,z :que,n :NN,e :!VV}
202	ETRECEQUE3	{e :!(PP||NN),x :FLEX[est|es],y :ce,z :que,n :NN|PP,v :VV}
203	ETRECEQUE4	{e :!(PP||NN),x :FLEX[est|es],y :ce,z :que,span____,b :BB}
204	ilya1	{x :il,y :avoir,z :y}
205	mettre1	{x :mettre,y :en,z :evidence}
206	idee1	{x :avoir,y :idee,z :de}
207	pleurer1	{x :pleurer,y :de,z :rireN}
208	dire1	{a :.le.moins,b :que,c :PP,d :pouvoirV,e :dire}
209	pouvoirfairepour1	{x :pouvoirV,y :faire}
210	vouloirdire1	{x :vouloir,y :dire}
211	aquoi1	{x :a,y :quoi,v :VV,n :NN}
212	PREP1	{x :a,y :le,p :ALT[PREP]}
213	PREP2	{x :en,p :ALT[PREP]}
214	GNPa1	{p :ALT[PREP],n1 :NN,x :de,n2 :NN,y :de,n3 :NN}
215	GNPb1	{p :ALT[PREP],n1 :NN,x :de,n2 :NN}
216	GNPc1	{p :ALT[PREP],x :de,n :NN}
217	GNPd1	{p :ALT[PREP],n :NN}
218	GNPni1	{x :ni,g1 :GNP,y :ni,g2 :GNP}
219	GNPsoit1	{x :soit,g1 :GNP,y :soit,g2 :GNP}
220	PCD1	{p :a|en|de,n :CD}
221	PCDetouPCD1	{p1 :PCD,rel :ALT[PREPCONJ],p2 :PCD}
222	GNPdede1	{n1 :NN,x :de,n2 :NN,y :de,n3 :NN}
223	GNPde1	{n1 :NN,x :de,n2 :NN}
224	GNPde2	{x :de,n :NN}
225	NetouN2	{n1 :NN,rel :ALT[PREPCONJ],n2 :NN}
226	quix1	{x :ce,v :etre,SPAN[s,2],n :NN|GNP,y :qui|dont,u___,z :BB}
227	quiz1	{n :NN|GNP,x :(qui|dont)&& !WSUB,u___,z :BB}
228	WSUBppz1	{n :NN|GNP,v :TENSE[PPZ],u___,z :BB}
229	verbcount1	{x :BB,ph_____,y :BB}
230	SPprepinf1	{p :ALT[PREPINF],v :TENSE[INF],FREE[u,POS[VV],5],z :BB}
231	SPinf1	{v :TENSE[INF],FREE[u,POS[VV|SP],5],z :BB}
232	SPenppz1	{e :en,v :TENSE[PPZ],FREE[u,POS[VV|SP],5],z :BB}
233	SPppz1	{v :TENSE[PPZ],FREE[u,POS[VV|SP],5],z :BB}
234	SPppl1	{v :TENSE[PPL|APPL],FREE[u,POS[VV|SP],5],z :BB}
235	SPquiV1	{y :qui,v :VV,FREE[u,POS[VV|SP],5],z :BB}
236	SPquiV2	{y :que,v :VV,FREE[u,POS[VV|SP],5],z :BB}
237	SPnpV1	{y :PP|NN|GNP,v :VV,FREE[u,POS[VV|SP],5],z :BB}
238	SPrest1	{v :VV,FREE[u,POS[VV|SP],10],z :BB}
239	stp3	{x :$openpth$,y___,z :$closepth$}
240	KK1	{x :KK,y_}

Annexe E

Schèmes de *DAFT* (𝕊)

Les 237 schèmes de 𝕊, dont les identifiants sont listés dans les tables 5.1, 5.2, 5.3, 5.4 et 5.5, ont chacun un type associé, une glose et au minimum un attribut. La liste complète de ces schèmes avec leurs différents attributs sont disponibles au format XML à l'adresse suivante :
`http://fbouchet.vorty.net/projects/daft/demo/DAFT.schemes.xml`
Ce fichier peut être visualisé sur la page :
`http://fbouchet.vorty.net/projects/daft/demo/schemes.html`

Les schèmes y sont représentés graphiquement comme suit :



Quelques exemples de schèmes de chaque classe suivant ce format sont donnés ci-dessous :

Modalités

POSSIBILITY concept	*Possibility given to someone by an authority (to do something for someone / about an object)*		
	person	**of**	*the person who we consider the possibilities*
	person	**offered-by**	*the person giving the possibilities*
	{Act, object, concept}	**concerning**	*the considered action or object*
	time-instant	**time**	*the considered time*

HELP concept, Act	*A seeker asks help from an helper (in order to do an action / about something) who helps by doing something.*		
	person	**seeker**	*the person asking help*
	person	**helper**	*the person who is required to help*
	Act	**for**	*the action needing help (goal to reach)*
	Act	**by**	*the action to do to help*
	{object, concept}	**about**	*the thing about which help is needed*

ASK §1	*Interrogation marker : for every question which is not a check, ie in which there is no supposition about the result of the question. It is always wrapping the rest of the request and can't be wrapped.*		
	{Act, object, concept, person}	**about**	

REFERENTIAL !1location-relative	*The link with a conceptual (i.e. not physical like PLACE) container*		
	{object, concept, person}	**to**	*the object or person taking value of referential*
	sel[True, False]	**_locked**	*lock*

INFOS	*Informations concerning a topic*		
Act, concept	Ref	**about**	*the considered topic*
	sel[True, False]	**more**	*more informations?*

UNION	*Union of N entity*		
§1	{object, concept, person, Act}	**members***	*the entities member of the union*
	sel[True, False]	**_locked**	*lock*

Actions

Show	*A presenter shows something to an audience, in a certain manner in a given place*	
Act	person	**presenter**
	person	**audience**
	object	**object**
	manner	**manner**
	location	**place**

Move	*To move an object or a person from a place to another (or inside a given place) in a certain manner*	
Act	object	**object**
	person	**person**
	location	**origin**
	location	**destination**
	location	**in**
	manner	**manner**

Modify	*To modify an object or a property, in a certain manner, to have a given value*	
Act	{object, concept, person}	**element**
	manner	**manner**
	Ppt	**property**

Go	*To go to a certain place at a given moment (ex: 'right now') in a certain way (ex: 'quickly') OR to go to a given past instant (ex: 'the previous move')*	
Act	location	**place***
	time-instant	**instant**
	manner	**manner**

Références

person	*A reference to person (human being or system).*		
person	sel[system, user, interlocutor, other, any]	**identifier**	*first category entities - 'any' for impersonal form*
Ref	Ppt	**property***	*the properties verified by the person*
	Act	**doing**	*the action done by the person*
	Act	**subject-of**	*the action which subject is the person*

object	*A reference to a physical element.*		
object	sel[application, area]	**identifier**	*first category entities*
Ref	Ppt	**property***	*the properties verified by the object*
	Act	**doing**	*the action done by the object*
	Act	**subject-of**	*the action which is subject is the object*
	sel[True, False]	**_locked**	*lock*

concept	*A reference to a more or less virtual concept.*		
concept	sel[game, step, example, rule, project]	**identifier**	*first category entities*
Ref	Ppt	**property***	*the properties verified by the concept*
	Act	**doing**	*the action done by the concept*
	Act	**subject-of**	*the action which subject is the concept*
	sel[True, False]	**_locked**	*lock*

Propriétés

name			
name	String	**val**	
Ppt			

quantity			
quantity	sel[0, 1, 1+, 2+, N-, N, N+, Few, Some, Most, All]	**val**	*aucun*
Ppt			

location-absolute-vertical	*Position verticale*		
location-absolute-vertical	Interval[{-1, 1}]	**val**	*-1=down, 1=up*
location-absolute			

type			
type	sel[window, webpage, document, frame, popup, title, menu, menubar, toolbar, scrollbar, textrule, hyperlink, button, label, textfield, chatbox, table, row, column, cell, graph, tree, node, list, palette, balloon, trashbin, cursor]	**val**	*Type d'un élément graphique de l'interface (widget)*
Ppt			

Annexe F

Règles de combinaison de DIG ($\mathbb{R}_{\mathcal{M}}$)

Les 24 règles de combinaison de DIG sont employées avant la règle de combinaison par défaut, pour gérer les cas où celle-ci ne fournit généralement pas de résultats satisfaisants. Une règle est composée de deux parties :

- une en-tête définissant les requêtes sur lesquelles elle peut s'appliquer, de la forme : $\{\_\_\_\_, e_1, e_2, e_3, \_\_\_\_\}$
  où ____ signifie "n'importe quel nombre d'entités" et où les e_i correspondent à des entités de $DAFT$ référencées sous la forme d'un sextuplet $\{c, h, rh, pos, key, pt\}$, où :
  - c est la classe de l'entité dans $\Gamma = \{M, A, R, P\}$;
  - h est l'identifiant d'un schème de $DAFT$;
  - rh est l'identifiant du schème, sauf si celui-ci est un marqueur (type ASK), auquel cas c'est l'identifiant de son contenu ;
  - pos est la nature correspondant à l'entité ;
  - key est la clé sémantique associée à l'entité ;
  - pt est le type de l'élément parent de l'entité dans l'arbre des types.
- un corps, constitué d'une séquence d'opérations élémentaires à effectuer sur la requête pour la modifier. La liste des opérations élémentaires est donnée dans le tableau F.1.

Les règles ci-dessous sont données au format Mathematica.

Suppression des successions de ASK : **ASK ASK → ASK**

```
{ {___,{_,_,ASK,_,_,_},___,{_,_,ASK,_,_,_},___},
  { new=req;
    If[§DRisempty[req[[x]],1],
      new=§DRdelete[req,x];,
      If[§DRisempty[req[[y]],1],
        new=§DRdelete[req,y];
      ];
    ]; new}}
```

Suppression UNION inutiles : **UNION ppt1 ppt2 → ppt1 ppt2**

Valeur de retour	Opération élémentaire	Description
Booléen	`§DRisempty[entite, numeroAttribut, ignoreMarqueurs, sousElement]`	vérifie que l'attribut d'une entité n'a pas de valeur associée. On peut passer outre les modalités de type marqueur (ASK ou NEG) et vérifier la valeur d'une entité contenu dans un champ multiple (par exemple une propriété d'un objet).
Booléen	`§DRcontains[entite, numeroAttribut, valeur]`	vérifie qu'un attribut donné (désigné par son rang dans l'schème) d'une entité a pour valeur associée celle passée en dernier paramètre.
Booléen	`§DRhasfield[entite, nomAttribut]`	vérifie si l'entité a un attribut du nom passé en paramètre.
Booléen	`§DRisbefore[entiteA, entiteB]`	vérifie si une entité A apparaît avant une entité B dans la requête fournie par `GRASP` (d'après le numéro de nœud Ω)
Booléen	`§DRissame[entiteA, entiteB]`	vérifie si deux entités A et B proviennent du même nœud dans la requête fournie par `GRASP`
{entités}	`§DRdelete[requete, entites]`	supprime d'une requête une ou plusieurs entité(s) passées en paramètre
entité	`§DRnew[entite]`	instancie une nouvelle entité telle que passée en paramètre
{entités}	`§DRmerge[requete, indexEntiteA, indexEntiteB]`	combine l'entité B avec un attribut libre de l'entité A selon la règle de combinaison par défaut, et retourne la requête d'origine où A a été remplacé par la combinaison
{entités}	`§DRsubst[requete, indexEntiteA, entiteB]`	substitue l'entité B à l'entité A dans la requête en cours de traitement
{entités}	`§DRfillfield[requete, indexEntiteA, attributARemplir, entiteB]`	combinaison (forcée) d'une entité B avec un attribut donné d'une entité A
{entités}	`§DRfillfield[requete, indexEntite, attributARemplir, valeur]`	association (forcée) d'une valeur à un attribut d'une entité
entité	`§DRunion[entiteA, entiteB]`	effectue l'union de deux entité ayant le même identifiant
entité	`§DRunlock[entite]`	passe l'attribut spécial de "blocage" d'une entité à faux, ce qui la rend disponible pour la combinaison avec une autre entité (*cf.* section 6.4.4)
{entités}	`§DRgetcondeltsaft[requete, indexEntite, typeContrainte, valeurContrainte]`	récupère les entités situées après celle dont l'index est donné dans la requête si et seulement si elles remplissent la condition spécifiée
String	`§DRrealhead[entite]`	retourne l'identifiant "réel" d'une entité, c'est-à-dire l'identifiant simple ou l'identifiant de l'entité servant de valeur pour un marqueur (NEG ou ASK)

Tableau F.1 Opérations élémentaires utilisées pour la combinaison d'entités *DAFT* par `DIG` (en italique : les champs optionnels)

```
{ {{_,UNION,_,_,_,_},{P,_,_,_,_,_},___,{P,_,_,_,_,_}},
  { new=req;
    If[§DRisempty[req[[x]],1],
      new=§DRdelete[req,x];
    ]; new}}
```

Articulation LINK du type “X est autre Y” :
LINK obj1 obj2 → LINK[ref = obj1, isa = obj2]

```
{ {___,{_,LINK,_,_,_,_},{R,_,_,_,_,_},{R,_,_,_,_,_},___},
  { new=req;
    If[§DRisempty[req[[x]],1,True],
      new=§DRunlock[§DRmerge[§DRmerge[req,x,y],x,z]];
      new=§DRdelete[new,{y,z}];
    ]; new}}
```

Articulation LINK du type “X a les propriétés Y et Z” :
LINK obj1 ppt* → LINK[ref = obj1, isa = obj2[ppt=...]]

```
{ {___,{_,LINK,_,_,_,_},___,{R,_,_,_,_,_},{P,_,_,_,_,_},___},
  { new=req;
    If[§DRisempty[req[[x]],1,True],
      new=§DRmerge[req,x,y];
      pptl=§DRgetcondeltsaft[req,z,class,P];
      new=§DRunlock[§DRmerge[new,x,§DRmerge[new,§DRnew[object[]],pptl]]];
      new=§DRdelete[new,pptl];
      new=§DRdelete[new,{y}];
    ]; new}}
```

Articulation LINK du type “la propriété X de Y a pour valeur Z” :
LINK VALUE name → LINK[ref = VALUE, isa = obj2[ppt=...]]

```
{ {___,{_,LINK,_,_,_,_},___,{_,VALUE,_,_,_,_},{P,_,_,_,_,_},___},
  { new=req;
    If[§DRisempty[req[[x]],1,True],
      If[§DRcontains[req[[y]][[1]][[3]],2,Head[req[[z]]]],
        If[!§DRisempty[req[[z]],1,True],
          new=§DRmerge[req,x,y];
          new=§DRunlock[§DRmerge[new,x,§DRmerge[new,§DRnew[concept[]],z]]];
          new=§DRdelete[new,{y,z}];
        ];
      ];
    ]; new}}
```

Subordonnées avec “être” : **qui LINK ppt → qui LINK obj[ppt]**

```
{ {___,{_,META,_,_,{___,\$QUI,___},_},{_,LINK,_,_,_,_},{P,_,_,_,_,_},___},
  {§DRsubst[req,z,§DRmerge[req,§DRnew[object[]],z]]}}
```

Subordonnées avec une action : **qui Act Ref → Ref[doing=Act[...]]**

```
{ {___,{_,META,_,_,{___,\$QUI,___},_},{A,_,_,_,_,_},{R,_,_,_,_,_},___},
  { If[§DRisbefore[req[[z]],req[[x]]],
      new=§DRfillfield[req,z,doing,y];
      new=§DRdelete[new,{x,y}];
      new    ,
      req
]}}
```

Verbes à références spatiale introduite par "à" : **à Act Ref→ Act[[loc]X=Ref[...]]**

```
{ {___,{A,_,_,_,{___,TOGO,___},_},___,{_,object,_,_,_,_},___},
  { If[§DRisbefore[req[[x]],req[[y]]],
      new=§DRmerge[req,x,§DRmerge[req,§DRnew[PLACE[position=,
            §DRnew[location-relative-inclusion[val=,True]]]],y]];
      new=§DRdelete[new,y];
      new    ,
      req
]}}
```

Formes passives : **LINK act Ref → Ref[subject-of=Act[...]]**

```
{ {___,{_,LINK,_,_,_,_},{A,_,_,_,_,_},{R,_,_,_,_,_},___},
  { new=req;
    If[§DRisbefore[req[[z]],req[[y]]],
      If[§DRhasfield[req[[z]],subject-of],
        new=§DRfillfield[req,z,subject-of,y];
        new=§DRdelete[new,{x,y}];
      ];
    ]; new}}
```

Conjonctions de références : **UNION Ref* → UNION[Ref, Ref, ...]**

```
{ {___,{_,UNION,_,_,_,_},{R,_,_,_,_,_},{R,_,_,_,_,_},___},
  { new=req;
    If[§DRisempty[req[[x]],1],
      new=§DRmerge[req,x,y];pptl=§DRgetcondeltsaft[req,z,class,R];
      new=§DRmerge[new,x,pptl];
      new=§DRdelete[new,pptl];
      new=§DRdelete[new,y];
  ]; new}}
```

Conjonctions d'actions : **UNION Act* → UNION[Act, Act, ...]**

```
{ {___,{_,UNION,_,_,_,_},{A,_,_,_,_,_},{A,_,_,_,_,_},___},
  { new=req;
    If[§DRisempty[req[[x]],1],
      new=§DRmerge[req,x,y];
      pptl=§DRgetcondeltsaft[req,z,class,A];
      new=§DRmerge[new,x,pptl];
      new=§DRdelete[new,pptl];
      new=§DRdelete[new,y];
    ]; new}}
```

"comment" sur la valeur d'une propriété : **ASK[WAY[]] ppt → ASK[] ppt**

```
{ {___,{_,WAY,ASK,_,_,_},___,{P,_,_,_,_,_},___},
  { new=req;
    If[§DRisempty[req[[y]],1],
      new[[x]]=§DRdelete[req[[x]],1,WAY];
    ]; new}}
```

Suppression des faux quantifieurs : **quantity=1 ppt → ppt**

```
{ {{_,quantity,_,_,{___,THE,___},_},{P,_,_,_,_,_},___},
  {§DRdelete[req,x]}}
```

Suppression d'éléments EXISTENCE inutiles : **CARD EXISTENCE → CARD**

```
{ {___,{_,CARD,_,_,_,_},___,{_,EXISTENCE,_,_,_,_},___},
  {§DRdelete[req,y]}}
```

Suppression d'éléments REASON inutiles : **WAY REASON → WAY**

```
{ {___,{_,WAY,_,_,_,_},___,{_,REASON,_,_,_,_},___},
  { new=req;
    If[!§DRisempty[req[[y]],1],
      new=§DRsubst[req,y,req[[y]][[1]][[3]]];    ,
      new=§DRdelete[req,y];
    ]; new}}
```

Combinaison des possessifs : **person ownedby → ownedby[person]**

```
{ {___,{_,person,_,_,_,_},___,{_,ownedby,_,_,_,_},___},
  { new=req;
    If[§DRisempty[req[[y]],1],
      new=§DRmerge[req,y,x];
      new=§DRdelete[new,x];
    ]; new}}
```

Forme figée d'interrogation sur une entité nommée :
ASK qui est Ref ? → INFOS[about=Ref[]]

```
{ {{_,_,ASK,_,_,_},{_,META,_,_,{___,\$QUI,___},_},{_,LINK,_,_,_,_},{R,_,_,_,_,_}},
  { new=§DRmerge[req,§DRnew[INFOS[]],p]}}
```

Forme figée d'interrogation sur une entité nommée réalisant une action :
qui ASK[Act[...]] ? → ASK[person[doing=Act[...]]]

```
{ {___,{_,META,_,_,{___,\$QUI,___},_},___,{A,_,ASK,_,_,_},___},
  { new=req;
    new[[y]]=§DRmerge[§DRnew[ASK[]],§DRfillfield[§DRnew[person[]],doing,req[[y]][[1]][[3]]]];
    new=§DRdelete[new,x];
  new}}
```

Valeur d'une propriété d'un objet :
de Ref1[...] Ref2[...] → Ref1[ppt*=REFERENTIAL[of=Ref2[...]],...]

```
{ {___,{_,META,_,_,{___,OF,___},_},___,{R,_,_,_,_,_,_},{R,_,_,_,_,_,_},___},
  { If[§DRisbefore[req[[y]],req[[x]]] && §DRisbefore[req[[x]],req[[z]]]
    && MemberQ[{object,concept,person},Head[req[[y]]]]
    && MemberQ[{object,concept,person},Head[req[[z]]]],
      new=req;
      new=§DRmerge[req,y,§DRmerge[req,§DRnew[REFERENTIAL[]],z]];
      new=§DRdelete[new,{x,z}];
      new    ,
      req]}}
```

Gestion du rapport locuteur/interlocuteur avec “dire” :
person1 person2 TELL → TELL[speaker=person1,interlocutor=person2]

```
{ {___,{_,TELL,_,_,_,_},___,{_,person,_,_,_,_},___,{_,person,_,_,_,_},___},
  { new=req;
    If[§DRisbefore[req[[y]],req[[x]]] && §DRisbefore[req[[z]],req[[x]]],
      If[§DRisbefore[req[[y]],req[[x]]],
        new=§DRmerge[§DRmerge[req,x,y],x,z];    ,
        new=§DRmerge[§DRmerge[req,x,z],x,y];
      ];
      new=§DRdelete[new,{y,z}];
    ]; new}}
```

Gestion du rapport locuteur/interlocuteur (2) avec “dire” :
TELL person1 → TELL[interlocutor=person1]
person1 TELL → TELL[speaker=person1]

```
{ {___,{_,TELL,_,_,_,_},___,{_,person,_,_,_,_},___},
  { new=req;
    If[§DRisbefore[req[[y]],req[[x]]],
      If[§DRisempty[req[[x]],1],
        new=§DRfillfield[req,x,speaker,y];
        new=§DRdelete[new,y];    ,
        If[§DRisempty[req[[x]],2],
          new=§DRfillfield[req,x,interlocutor,y];
          new=§DRdelete[new,y];
        ];
      ];    ,
      If[§DRisempty[req[[x]],2],
        new=§DRfillfield[req,x,interlocutor,y];
        new=§DRdelete[new,y];
      ];
  ]; new}}
```

Question sur un moment :
ASK INSTANT[. . .] → ASK[about=concept[ppt*=INSTANT[. . .]]]

```
{ {___,{_,ASK,_,_,_,_},___,{_,INSTANT,_,_,_,_},___},
  { new=req;
    If[!§DRisempty[req[[y]],1],
      new=§DRmerge[new,x,§DRmerge[new,§DRnew[concept[]],y]];
      new=§DRdelete[new,y];
    ]; new}}
```

Question sur un lieu :
ASK PLACE[. . .] → ASK[about=concept[ppt*=PLACE[. . .]]]

```
{ {___,{_,ASK,_,_,_,_},___,{_,PLACE,_,_,_,_},___},
  { new=req;
    If[!§DRisempty[req[[y]],1],
      new=§DRmerge[new,x,§DRmerge[new,§DRnew[concept[]],y]];
      new=§DRdelete[new,y];
    ]; new}}
```

Annexe G

Questionnaires d'acceptabilité d'un agent conversationnel

Dans la section 8.1.2, nous souhaitons tester l'impact de la responsivité (R), de la variabilité linguistique (V) et de la dépendance entre les réponses (D) dans le cadre d'un agent conversationnel. Au sein de l'expérience qui y est décrite, nous avons eu recours à deux questionnaires (Q_1 et Q_2) identiques, correspondant au "questionnaire post-interaction" ci-dessous, et un questionnaire (Q_3) de comparaison des deux agents, qui correspond au "questionnaire final".
Ces questionnaires sont en anglais, dans la mesure où l'interaction avec l'agent se faisait dans cette langue, et qu'on suppose donc que les sujets en ont une maîtrise minimale. Toutefois, les noms des principaux paramètres testés étaient précisés en chinois (non montré ici). De plus, les sujets étaient libres de laisser un commentaire complémentaire en français, anglais ou chinois.

D'un point de vue technique, ils ont été réalisés à l'aide de l'option de création d'un formulaire sous *Google Docs*.

Questionnaire post-interaction :

You'll be presented several statements concerning the interaction you have just had with a conversational character : please give your level of agreement with them from 1 to 5, where :

- 1 means you totally disagree
- 2 means you rather disagree
- 3 means you don't know or have no opinion
- 4 means you rather agree
- 5 means you totally agree

For each question, you can also justify your ranking (in English, French or Chinese).

1. The character's answers were precise *(if you ask for hour, the answer is "17:02", not "around 5pm")*
2. The character's answers were relevant *(if you ask about musical tastes, the answer is about music and not about sports for example)*
3. The character's answers were believable *(if the agent told you it was a male, it is not believable according her look)*
4. The character's answers were human-like *(you believe a real human being could answer that way)*
5. The character's answers were changing *(if asking several times the same question, the answer is always the same, it is not changing)*
6. The character was reluctant to provide some information *(it refused to give some information, even if you were insisting)*
7. The character's answers were globally satisfying *(you have been generally happy with the answers given)*

Questionnaire final :

You have now finished interacting with both characters : please answer to those final questions to know a little bit more about you and your feelings about this experiment in general.

1. Email address
2. Gender
3. Age
4. Education level : primary, secondary, university undergraduate, university graduate, university PhD
5. Education specialization : computer science, science (except computer science), human sciences, economics, humanities, other
6. Knowledge in computer science (on a scale from 1 to 5)
7. Had you already interacted with a virtual character like the one used in this experiment ? *(yes or no)*
8. Which character was the most precise in its answers ? *(the 1^{st}, the 2^{nd} or no difference)*
9. Which character was the most relevant in its answers ? *(the 1^{st}, the 2^{nd} or no difference)*
10. Which character was the most believable in its answers ? *(the 1^{st}, the 2^{nd} or no difference)*
11. Which character was the most human-like in its answers ? *(the 1^{st}, the 2^{nd} or no difference)*
12. Globally I prefered interacting with... *(the 1^{st}, the 2^{nd} or no difference)*

Acronymes

ACA	Agent Conversationnel Assistant.
AIML	Artificial Intelligence Markup Language.
AJAX	Asynchronous JavaScript and XML.
AMI	Architectures et Modèles pour l'Interaction.
BDI	Beliefs Desires Intentions.
BNC	British National Corpus.
CFG	Context-Free Grammar.
CHS	Contextual Help System.
CLEF	Cross Language Evaluation Forum.
COCA	Corpus of Contemporary American English.
CODES	COoperative Music Prototype DESign.
DAFT	Dialogical Agent Formal Talk.
DAMSL	Dialog Act Markup in Several Layers.
DIVA	DOM Integrated Virtual Agents.
DOM	Document Object Model.
DRT	Discourse Representation Theory.
ECA	Agent Conversationnel Animé.
ELDA	Evaluations and Language resources Distribution Agency.
ELRA	European Language Ressources Association.
EVALDA	Infrastructure d'EValuation à ELDA.
FAQ	Foire Aux Questions.
GRASP	Générateur de Requêtes par Analyse Sémantique Partielle.
GTACA	Groupe de Travail sur les Agents Conversationnels Animés.
GUI	Graphical User Interface.

HDL	Heuristic Description Language.
HMM	Hidden Markov Model.
IHM	Interface Homme-Machine.
KPPV	K Plus Proches Voisins.
LEA	LIMSI Embodied Agent.
LEFFF	LExique des Formes Fléchies du Français.
LREC	Language Resources and Evaluation Conference.
MDL	Model Description Language.
MOZ	Magicien d'Oz.
MQL	Model Query Language.
NICE	Natural Interactive Communication for Edutainment.
NLI	Interface en Langue Naturelle.
OEC	Oxford English Corpus.
OWL	Web Ontology Language.
OZONE	Offering an Open and Optimal roadmap towards consumer oriented ambient intelligence.
PCFG	Probablistic Context-Free Grammar.
POS	Part of Speech.
PPV	Plus Proche Voisin.
QR	questions-réponses.
RDF	Resource Description Framework.
SDHM	Système de Dialogue Homme-Machine.
SDRT	Structured Discourse Representation Theory.
SPA	Smart Personal Assistant.
TALN	Traitement Automatique de la Langue Naturelle.
TREC	Text REtrieval Conference.
TTS	Text-To-Speech.

VDL	View Design Language.
WSD	Word Sense Disambiguation.
XML	eXtended Markup Language.
XRCE	Xerox Research Centre Europe.

Glossaire

acte de dialogue moyen employé par un locuteur pour agir sur son environnement par le biais des mots qu'il utilise, par exemple pour convaincre, informer, promettre...

acte de langage voir acte de dialogue.

analyseur morphosyntaxique système permettant de déterminer la nature et la fonction des constituants d'un texte.

bigramme sous-séquence de deux éléments (cas particulier de N-gramme).

clé sémantique au sein de `DAFT`, il s'agit d'un équivalent des synsets de WordNet utilisés pour représenter un sens particulier pouvant être lié à un ou plusieurs lemmes. Inversement, un lemme peut aussi être lié à plusieurs clés sémantiques.

corpus "large collection d'élements linguistiques typiquement composé d'usages attestés de la langue" [McEnery, 2003, p. 449], qui doit être organisée et "recueillie dans les limites d'un cadre d'échantillonnage conçu pour autoriser l'exploration de certaines caractéristiques linguistiques". A ce titre, de nombreux corpus sont annotés en fonction du cadre d'étude. L'utilisation d'un corpus en linguistique computationnelle s'oppose à l'approche générative consistant à considérer des phrases créées de toutes pièces.

désambigüisation lexicale identification du sens d'un mot polysémique dans le cadre donné par une phrase où il occurre.

Agent Conversationnel Animé ou Embodied Conversational Agent.

entité instanciation d'un schème, il s'agit de l'élément de base du langage *DAFT* permettant de constituer une requête formelle.

étiqueteur morphosyntaxique voir analyseur morphosyntaxique.

expert par opposition au novice, utilisateur doté de compétences en informatique et connaissant bien le fonctionnement de l'application considérée.

flexion ensemble des modifications d'un même signifiant, qui varient en fonction du genre et du nombre (pour un nom, un adjectif ou un pronom) mais aussi du temps, du mode et de la voix pour un verbe.

frame structure de données permettant de modéliser une association sémantique d'éléments attendus pour décrire une situation stéréotypique. Une frame a le même pouvoir représentatif que la logique du premier ordre.

glose commentaire expliquant en langue naturelle la signification d'un terme.

homonymie relation d'ambiguïté liant deux mots (homonymes) ayant un sens différent mais partageant la même graphie (homographes) et/ou prononciation (homophones). Dans un dictionnaire, chacun des homonymes a une entrée qui lui est propre..

langage contrôlé ensemble artificiellement restreint de la langue naturelle afin d'atteindre un objectif particulier (par exemple la rédaction d'un document technique) – *cf.* [Kittredge, 2003].

lemme unité autonome du lexique d'une langue constituant la forme canonique d'un mot, représentant ainsi toutes les formes qui en sont morphologiquement dérivées.

métonymie figure de style consistant à changer la désignation d'un substantif en le remplaçant par un autre entretenant une relation avec le premier (cause/effet, contenant/contenu, institution/lieu, artiste/oeuvre...

Magicien d'Oz expérience dans laquelle on fait croire à un sujet qu'il interagit avec une machine alors que celle-ci est en fait simulée par un expérimentateur (le "magicien"). L'objectif est de pouvoir obtenir un comportement représentatif de ce que sera celui d'un utilisateur confronté au système final. Il s'agit donc d'une expérience utile notamment pour la conception d'un logiciel pour modéliser les utilisateurs et déterminer leurs besoins.

nature catégorie grammaticale d'un mot (nom, verbe, adjectif...). Deux mots de même nature remplissent la même fonction syntaxique et peuvent se substituer l'un à l'autre.

N-gramme sous-séquence de N éléments issus d'une séquence donnée. Ce concept est généralement utilisé en Traitement Automatique des Langues pour estimer la probabilité d'apparition d'un élément en connaissant uniquement ses plus proches voisins, selon l'hypothèse markovienne.

Interface en Langue Naturelle système d'interaction homme-machine en langue naturelle dans lequel les requêtes reçues sont traitées de manière plus ou moins isolée, par opposition aux SDHM.

novice utilisateur occasionnel, n'ayant peu ou pas de compétences en informatique de manière générale, et plus particulièrement dans le cadre de l'utilisation de l'application considérée.

polysémie qualité d'un mot doté de plusieurs sens différents. Dans un dictionnaire, ces sens sont listés sous la même entrée.

POS Tagger voir analyseur morphosyntaxique.

Part Of Speech voir nature.

pragmatique pragmatique domaine de la linguistique s'intéressant aux phénomènes dont le sens ne peut être compris que dans le contexte d'énonciation.

schème modélisation d'une association sémantique d'éléments (*cf.* frame). Un schème peut être instancié pour donner une entité afin de modéliser une requête en *DAFT*.

sous-langage sous-ensemble d'expressions de la langue naturelle qui apparaît naturellement (par opposition aux langages contrôlés) dans le cadre de domaines sémantiques limités – *cf.* [Kittredge, 2003].

tour de parole segment de dialogue constitué par l'intervention d'un des locuteurs. Il s'achève lorsqu'un nouveau locuteur prend la parole, ou lorsque le dialogue prend fin.

validation croisée 10-fold technique d'évaluation de la performance d'un classifieur, basée sur une division préalable en 10 sous-ensembles aléatoires des données annotées disponibles : 9 forment l'ensemble d'apprentissage tandis que le dernier sert d'ensemble de test. On effectue ensuite la moyenne des résultats en prenant successivement chacun des 10 sous-ensemble comme ensemble de test.

validation croisée leave-one-out technique d'évaluation de la performance d'un classifieur dans laquelle l'ensemble on considère successivement (N fois) le singleton formé par chacun des N éléments annotés comme ensemble de test, les $(N-1)$ éléments restants constituant l'ensemble d'apprentissage.

Word Sense Disambiguation voir désambigüisation lexicale.