Un Langage de Description d'Agents dédié à l'Interaction Dialogique

J.-P. Sansonnet G. Pitel N. Sabouret
jps@limsi.fr pitel@limsi.fr nico@limsi.fr

LIMSI-CNRS BP 133
F-91403 Orsay

Résumé :
Dans cet article, nous discutons de l'interaction en langue naturelle entre des utilisateurs et des services ou composants logiciels de l'Internet vus comme des agents conversationnels. Nous proposons un langage de description d'agent et un langage de requêtes spécialement conçus pour l'introspection et le raisonnement sur la **structure** et le **fonctionnement** de ce que nous appelons des **agents**. Nous exposons ici un aperçu de ces langages avec un exemple l'illustrant, et nous abordons les techiques que nous mettons en œuvre pour utiliser les descriptions des agents pour le traitement de la langue. **dialogiques**.

Mots-clés : Agents Conversationnels, Requêtes en Langue Naturelle, Langages de Description d'Agents

Abstract:
This paper examines the natural language interaction between human users and web services or software components considered as conversationnal agents. The paper introduces an agent description language and a query language specially designed for introspection and reasonning about the **structure** and **behavior** of what we call **dialogical agents**. The paper exposes an overview of these languages with an example of their usage, and explains how we use agents descriptions for natural language processing.

Keywords: Conversationnal Agents, Natural Language Queries, Agent Description Language

1 Agents Dialogiques

Notre travail se situe à la frontière entre le Traitement Automatique de la Langue (TAL) et les Systèmes Multi-Agents (SMA), plus particulièrement les Agents Assistants d'Interface (AAI). Dans le domaine des SMA, le problème de l'interaction avec les humains n'a émergé que récemment. Pourtant l'Interaction en Langue Naturelle (ILN) est depuis quelques années considérée comme une priorité [6] dans le développement des services pour les *gens ordinaires*.

Le concept émergeant de *gens ordinaires* (*ordinary people*) qualifie la différence cognitive qui sépare les anciens utilisateurs spécialistes des systèmes informatiques des nouveaux consommateurs des services "grand public". Cela met en avant des problématiques nouvelles comme par exemple la notion de *bon sens* dans les questions (common sense requests).

C'est pour cette raison que nous proposons une approche permettant de rapprocher la conception des composants de la conception de leur Interface Linguistique.

Nous définissons les Agents Dialogiques (AD) comme des composants logiciels dotés des capacités d'interaction avec les utilisateurs par la voie de la langue naturelle. La figure 1 montre l'architecture d'un système d'interaction avec un Agent Dialogique, il est composé de trois parties : le composant effectif, le médiateur et l'interface.

1.1 Le composant effectif

Le composant effectif contient le processus logiciel ou matériel réel. Typiquement, il s'agira d'un composant logiciel doté d'une interface graphique. On considère qu'il est interopérable, c'est-à-dire qu'il expose une interface de programmation accessible à un processus extérieur permettant d'atteindre ses méthodes (pour contrôler son fonctionnement) et ses propriétés (pour obtenir des informations sur sa structure et son contenu).

1.2 Le médiateur

Il est décrit dans un Langage de Description d'Agent (LDA), fait l'interface entre la langue et le composant effectif. Jouant le rôle de *médiateur*, il possède une représentation du composant effectif du point de vue *supposé* [1] de l'utilisateur (agentification), lui permettant de répondre aux questions de l'utilisateur sur le composant et contrôler celui-ci. Une de ses tâches consiste à maintenir la cohérence de la représentation.

[1] Le modèle perçu pouvant être très différent de l'implémentation réelle, on ne peut pas supposer que l'utilisateur fera référence aux parties ou aux fonctions du composant en des termes réflétant sa réalité implémentatoire, mais plutôt avec sa représentation cognitive propre. Nous nous appuyons pour affirmer cela sur la notion d'*Affordances* proposée par Gibson [10] et plus tard Norman [5].

1.3 L'interface

L'interface reflète les changements intervenant dans le composant. C'est en étudiant cette interface qu'on doit trouver les catégories des éléments que l'utilisateur devrait conceptualiser, et donc utiliser dans l'interaction pour référer aux objets ou aux événements.

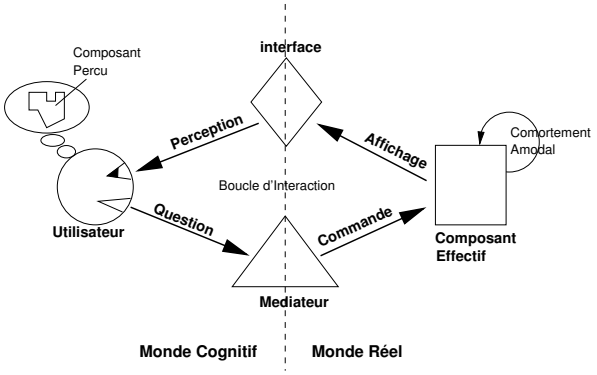

FIG. 1 – Boucle pour un Agent Dialogique.

2 Le Langage VDL

2.1 Définitions

Médiateur. Un médiateur M est composé de deux environnements[2] E_D et E_V. E_D est un ensemble de définitions de vues, aussi appelé définition du médiateur. E_V est un ensemble d'instanciations de vues, aussi appelée l'état du médiateur. Une fois lancé, l'état des vues évolue de manière synchrone au comportement du composant, et en fonction des requêtes de l'utilisateur.

Vue. L'entité de base de VDL est la vue, supportant la structure de la représentation. La définition d'une vue suit la syntaxe $DEF[D_i, \sigma, \{farg, ..\}, (rule|inst)^*; ..]$ avec :

- D_i : nom de la vue,
- σ : *schème* de la vue (cf. section 2.1).
- $fargs$: arguments de l'instanciation de la définition D_i,
- $rules$: utilisées pour attacher des définitions spécifiques à D_i (cf. section 2.2).
- $insts$; instances des sous-vues du domaine de D_i créées avec l'opérateur NEW (cf. section 2.3).

[2]D'une manière similaire à la distinction T_{box}/A_{box} des Logiques de Description [2].

A partir d'une définition de vue D_i, il est possible de créer une instance dans l'environnement d'exécution E_V à l'aide de l'opérateur NEW dont la syntaxe est $NEW[D_i, V_j, args, ..]$ avec :

- V_j ; nom de la nouvelle instance de vue,
- $args$; arguments qui doivent correspondre aux $fargs$ de la définition D_i.

Un ensemble E_G de définitions de vues, appelé GROUND, est inclus avec le système de programmation VDL.

Schèmes. Afin de construire des algorithmes travaillant sur la stucture du médiateur, il était nécessaire de restreindre les structures de base possibles à six schèmes très simples : $\Sigma = \{K, V, R, L, F, P\}$. Le programmeur est donc contraint de décrire le composant effectif en des termes appartenant exclusivement à Σ, mais cela apporte l'avantage de rendre systématique l'exploration de la représentation du composant.

Les schèmes sont présentés à la figure 2. Ils peuvent être catégorisés selon deux axes : 1) si leur structure est statique ou dynamique 2) si leur structure est scalaire, vectorielle ou encore fonctionnelle. Chacun de ces schèmes peut contenir des expressions terminales ou des vues récursivement.

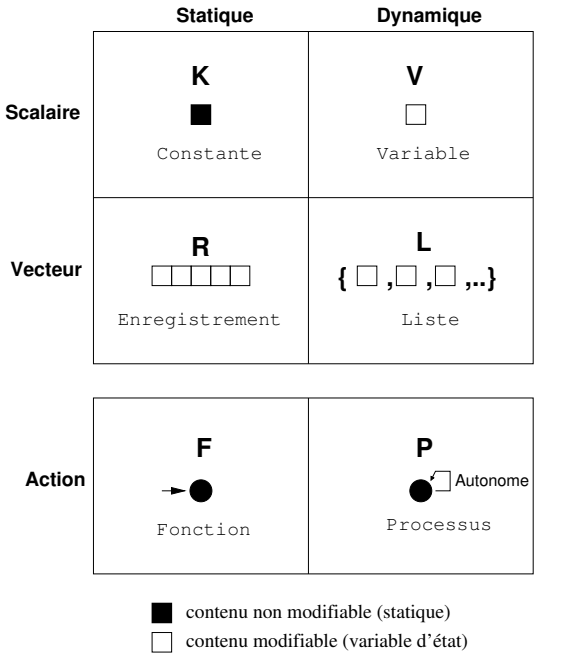

FIG. 2 – Les schèmes structurent l'information selon six formes primitives.

2.2 Effectuer une requête sur une vue

Nous définissons le langage VQL de requête sur les vues (View Query Language). En VQL, le mécanisme de requête est fondé sur la notion **d'observateur**.

Soit un type d'observateur $\varphi_i \in \Phi$, où Φ est l'ensemble prédéfini des types d'observateurs (au nombre de 35 dans notre implémentation). Un observateur $\varphi_i(D_j)$ est un opérateur applicable à **n'importe quelle** instance de vue définie par la vue D_j.

Les observateurs peuvent soit modifier l'environnement d'exécution du médiateur (ce sont les observateurs dynamiques, comme SET[V_1,V_1+1]) ou bien renvoyer une information sur la structure de celui-ci (observateurs statiques, par exemple SIZE[V_1]).

La sémantique opérationnelle des observateurs génériques (pour les vues basiques du GROUND) est définie par un tableau de sous-fonctions $\varphi_i(\sigma)$ où $\sigma \in \Sigma$. Les observateurs spécifiques à une vue sont définis grâce aux règles (*rules*) dans la définition de cette vue.

2.3 Exemples de programmation

Pour illustrer comment un programmeur peut en pratique définir la structure et le comportement d'un agent dialogique en VDL, nous montrons dans ce paragraphe un exemple partiel de programmation[3] sur le jeu de Hanoï.

Le jeu de Hanoï

```
DEF[move,F,{},
   NAME=LEX[SYNSET['move'],
    SYNSET['displace']];
   ARGS={{STACK,¬EMPTY},{STACK}};
   CHECK=(EMPTY[#2]∨SIZE[READ[#1]]<
    SIZE[READ[#2]])&;
   VAL=(POP[#1];    PUSH[#2,IT])&
  ]
DEF[hanoi,R,{},
  NEW[RED, red];
  NEW[RECTANGLE,rect,'disk'];
  NEW[FIGURE,d1,'disk',rect,red,
  NEW[BIG,z1]];
  ...
  NEW[STACK,s1,NIL,3];
  ...
  NEW[move, m];
  NAME=LEX[SYNSET['game']];
  VAL={s1,s2,s3};
  ...
]
```

[3]Le langage VDL étant actuellement implémentée en MATHEMATICA 4.0, les notations utilisées paraitront plus familières aux habitués des systèmes à base de règles de réécriture comme Mapple ou Mathematica.

Seules deux définitions doivent être programmées :

• **move** est associée avec l'opération de base du jeu. Son schème est une fonction (σ = F). Cette vue décrit le déroulement d'une opération de déplacement, en utilisant les observateurs ARGS et CHECK (Nous l'expliquons en détail à la section 3.3).

• **hanoi** est un vecteur d'éléments R contenant VAL = {s1,s2,s3}. Il est affiché avec l'orientation explicitement horizontale (HOR) ORIENT. La définition de vue hanoi instancie les sous-vues déjà prédéfinies dans le fichier GROUND : RED, RECTANGLE, BIG, MEDIUM, SMALL ainsi que des sous-vues définies explicitement.

3 Interaction en Langue Naturelle

3.1 Requêtes en Langue Naturelle

Les RLN écrites sont traitées par un analyseur syntaxico-sémantique qui produit une forme interne de la requête. On considère que de tels outils sont aujourd'hui impossibles à produire pour traiter la langue naturelle en général, mais certaines simplifications [1, 7] peuvent être envisagées dans le cadre d'une tâche donnée :

(a) La structure générale d'une requête est de la forme F(P) [9], où F est l'acte de langage [8], et P est une forme propositionnelle. Ceci nous a amené à concevoir l'analyseur sémantique afin qu'il extraie le Sytagme Verbal (SV) correspondant à l'action, le Syntagme Nominal (SN) correspondant aux arguments de cette action P et les prépositions correspondant aux relations.

(b) Le principe d'*Affordance* nous permet de prévoir que l'utilisateur réfèrera aux entités visibles selon la manière dont il les catégorise et selon les caractères que ceux-ci *fourniront* par leur apparence (paragraphe 3.2).

3.2 Recherche des référents

L'analyseur sémantique. L'analyseur sémantique met en œuvre une approche ascendante, orientée

règles, et de droite à gauche, avec l'aide d'une phase de balisage "Part of Speech" (TreeTagger [11]) pour l'analyse lexicale. Le niveau de sémantique lexicale est basé sur Wordnet [3] où les sens sont représentés comme des clefs uniques sous la forme de *synsets* $\phi_i \in \Psi$, reliés à des entrées lexicales (mots racines).

L'appariement d'entités lexicales. L'appariement d'entités lexicales consiste à établir une relation entre les synsets lexicaux des RLN (notés ϕ_i) et les vues définies dans E_V. Par exemple, dans `NG[N['disque'],Q[P]]` il faut attacher le *synset* ϕ_1=`'disque'` à {`d1,d2,d3`}, ce qui peut être réalisé simplement en observant chaque vue de E_V avec l'observateur $NAME$ et conserver toutes les vues répondant au synset ϕ_1. On se trouve face à trois possibilités de correspondance : aucune vue, une seule vue, ou plusieurs vues. Lorsqu'aucune solution n'est produite, nous sommes en présence d'un décalage cognitif. Les cas possibles sont discutés aux paragraphes suivants.

3.3 Heuristiques pour le traitement

Résolution du décalage sémantique La résolution du décalage sémantique fonctionne en élargissant le champ lexical trouvé. Pour une RLN de synset ϕ_0, il faut étendre le synset à son voisinage dans le réseau sémantique de Wordnet. L'exploration est exponentielle, mais dans la majorité des cas expérimentaux, une profondeur de 5 à 7 en moyenne amène à un résultat, ce qui reste très coûteux, mais devient abordable pour une station de travail actuelle.

Désambiguïsation adjectivale Cette résolution du décalage sémantique part de l'idée que la vue référée par le GN sera la plus petite vue dans le médiateur contenant tous les synsets du GN. Nous définissons une fonction $\Delta : E_V \longmapsto E_V$ qui retourne récursivement toutes les vues auquelles réfère une vue V_i. Ensuite, nous définissons la relation binaire sur E_V telle que $V_i \prec V_j \Leftrightarrow V_i \in \Delta(V_j)$ qui donne à E_V un ordre partiel. Nous pouvons alors construire l'algorithme qui donne la plus petite vue (au sens de l'opérateur $\prec$) contenant tous les synsets $\phi_{i=1,n}$ du GN.

Désambiguïsation pragmatique Le principe de désambiguisation pragmatique consiste à inverser le processus d'analyse de la RLN :

l'acte de langage de la RLN est extrait
SI *il peut être relié à une vue fonctionnelle* V
ALORS
POUR CHAQUE *argument* a_i DE V,
les vues candidates pour a_i *sont sélectionnées,*
SI *elles sont ambiguës* ALORS
les informations supplémentaires de la RLN sont utilisées

Afin de rendre possible cette utilisation des vues, le programmeur doit décrire, pour chaque vue fonctionnelle V dans le médiateur les contraintes de type sur les arguments de V et les préconditions [4] à l'exécution de la vue V représentées par un prédicat.

4 Conclusion et perspectives

Nous avons proposé un langage de conception de médiateur spécialement dédié à l'interaction en langue naturelle, afin d'apporter une réponse au besoin croissant d'interfaces plus naturelles entre les *gens ordinaires* et les services et systèmes en ligne.

Le langage proposé est encore primitif, mais nous pensons que les principes sur lesquels nous l'avons construit sont intéressants comparés d'une part aux langages à balises pour la description de contenu statique, et d'autre part aux approches Orienté-Objet des composants dynamiques, qui ne permettent pas de décrire des connaissances utiles pour assister les utilisateurs.

En limitant VDL volontairement à six schémas structurels simples et quelques observateurs prédéfinis, nous sommes capables de proposer des algorithmes de résolution de la référence relativement *génériques* pour assurer l'ILN pour n'importe quel composant.

Un important travail reste à faire afin d'augmenter l'expressivité du langage de description et de requête sur les agents, et pour élargir la gamme des fonctions de la langue naturelle prise en compte par ce langage.

Références

[1] J.F. Allen, D. K. Byron, M. Dzikovska, G. Ferguson, L. Galescu, A. Stent, Towards Conversational Human-Computer Interaction, *AI Magazine*, 2001.

[2] F. Baader, *Terminological knowledge representation : a proposal for a terminological logic*. TR - 90-04 DFKI, 1990.

[3] C. D. Fellbaum, Ed. *WordNet : An Electronic Lexical Database*, MIT Press, 1998.

[4] K. Lochbaum. The use of knowledge preconditions in language processing. *IJCAI'95*, p1260–1266, Montreal Canada, 1995.p.217-227. May2001

[5] D.A. Norman, *The psychology of everyday things*, Basic books Inc., New-York, 1988.

[6] J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, T. Carey, *Human-Computer Interaction*, Addison, Wesley, England, 1994.

[7] C. Rich, C. Sidner, When Agents Collaborate with People, *Proceedings of the First International Conference on Autonomous Agents*, 1997.

[8] J. R. Searle, *Speech acts*, Cambridge University Press, Cambridge, 1969.

[9] J.R. Searle, D. Vanderveken, *Foundations of illocutionary logic* , Cambridge University Press, 1985.

[10] J. J. Gibson, *The Ecological Approach to Visual Perception*, Houghton Mifflin, Boston, 1979.

[11] H. Schmid. Probabilistic Part–of–Speech Tagging Using Decision Trees. *Proceedings of the Conference on New Methods in Language Processing*. Manchester, UK., 1994