

Acquisition de connaissances perceptives pour un agent assistant

David Leray¹, Jean-Paul Sansonnet¹

¹ LIMSI-CNRS, Université de Paris XI,
{leray, jps}@limsi.fr

Résumé : L'un des enjeux importants du développement et de la conception d'une application est la production d'une assistance efficace. Or, force est de constater que cette assistance n'est pas adaptée aux usagers novices. Les causes d'une situation où l'utilisateur se retrouve en besoin urgent d'assistance sont multiples et nous retiendrons comme une cause fondamentale la perception de l'application possiblement faussée d'un utilisateur novice. Nous défendons dans cet article l'idée que la gestion d'une assistance répondant à cette problématique peut et doit être intégrée dès le début de la conception de l'application. Dans cette optique, en se basant sur des exemples de situations d'assistance, nous montrerons en quoi une méthodologie d'acquisition de connaissances fondée sur la perception est nécessaire et complémentaire aux approches de modélisation classiques.

Mots-clés : Communications de recherche, Agent Assistant, Extraction de connaissances d'assistance, Aide à la Conception, Connaissances perceptives.

1 Introduction

La pénétration de l'informatique dans les foyers domestiques a permis l'apparition d'une nouvelle classe d'utilisateurs grand public caractérisée par un nombre important d'individus et une méconnaissance des concepts fondamentaux de l'informatique : les **usagers novices**.

Les concepteurs de logiciels doivent s'adapter à ces nouveaux utilisateurs et notamment les outils d'assistance doivent évoluer. Or, il est évident que les outils actuels ne remplissent pas leur rôle, ce qui conduit à une sous-exploitation ou un abandon de logiciels pourtant garnis d'un nombre importants de fonctionnalités intéressantes (Fisher 85).

L'échec des solutions d'assistance trouvées dans les logiciels proposés au grand public actuellement tient à une caractéristique bien particulière : le *paradoxe de motivation* (Carroll 87) : contrairement à l'utilisateur expert, l'utilisateur novice ne prendra pas le temps nécessaire pour trouver une solution ou explorer l'ensemble des fonctionnalités proposées. Or l'ensemble des solutions d'assistance part du postulat inverse, forçant l'utilisateur à de grands efforts d'exploration, renvoyant ainsi l'utilisateur novice à sa frustration.

L'Assistance Contextuelle en Ligne (ou CHS – Contextual help Systems) (Cf. Capobianco et al, 01, 02 pour une synthèse) apparaît alors comme une alternative séduisante : consistant à ne faire apparaître que les informations utiles à l'utilisateur au moment où elles le sont, on évite le besoin d'exploration lié aux assistances usuelles. Toutefois, la plus grosse difficulté est alors de choisir l'information en extrapolant la tâche dans laquelle un usager novice est engagé. Si cette extrapolation est fautive, toute l'assistance échoue : les informations présentées étant inutiles cela provoque chez l'utilisateur un désarroi encore plus grand.

Il est intéressant de noter qu'à ce stade, l'utilisateur *verbalisera* (i.e l'exprimera en langue naturelle) spontanément son désarroi ou son problème (Roestler 95). Or, on peut alors remarquer que l'assistance contextuelle est en fait une assistance à sens unique : de l'application vers l'utilisateur sans prendre en compte un éventuel retour de l'utilisateur. Il devient évident qu'il faut rajouter une dimension *dialogique* à l'assistance contextuelle.

Or, l'ajout d'un système de traitement de la langue naturelle rajoute un effort de développement supplémentaire qui devrait être évité aux concepteurs d'une application de par sa complexité et son temps de mise en œuvre (Allen 95). Pour cette raison, il nous semble important d'intégrer la gestion de l'assistance à la phase de conception de l'application de façon transparente.

La section suivante présente le challenge de l'acquisition de connaissances sémantiques de haut niveau nécessaires à l'assistance. La section 3 présente l'architecture générale de notre solution d'assistance tandis que la section 4 présente l'environnement d'acquisition de connaissances sémantiques, illustré dans la section 5 par une étude de cas et discuté dans la section 6.

2 Fossé sémantique

2.1 Problématique du fossé sémantique

La difficulté principale de l'assistance aux utilisateurs novices c'est que la requête formulée pour exprimer le problème est l'expression de leur compréhension de l'application qu'ils utilisent. Des études ont pu montrer (Capobianco et al 01) que la plupart des requêtes sont des accès à des « concepts » de l'application, c'est-à-dire des références. Or les concepts référés dans la requête sont ceux que l'utilisateur imagine se rapporter à l'application. La fonction d'assistance consistera alors à mettre en correspondance la sémantique contenue dans la requête avec un objet de l'application disponible au regard de l'agent assistant (par exemple une classe dans un diagramme UML).

Mais si l'on peut dire que le concept référé a forcément une signification dans l'application (l'utilisateur a mentionné quelque chose qu'il voit ou qu'il associe à l'application), celui-ci ne se retrouve pas forcément dans une représentation formelle de l'application, clef de la production automatique d'une assistance. C'est ce que nous appelons *le fossé sémantique* : c'est la différence parfois énorme entre la représentation mentale de l'utilisateur et la réalité des fonctionnalités implémentées.

2.2 Exemple de fossé sémantique

La capture d'écran de la figure 1 présente un extrait du site web d'une agence de voyage bien connue (<http://www.voyage-sncf.com>). Il s'agit de la zone permettant de donner les renseignements utiles à la préparation du voyage (zone noyée parmi les offres promotionnelles).

Supposons un usager novice voulant se rendre de Paris à Berlin : il renseigne les différents champs, notamment ceux concernant les dates de départ et de retour. Mais il se trompe et, rentrant à la main la date de retour, il indique un mois antérieur à celui en cours sans changer l'année, ce que l'application détecte et corrige. L'usager ne comprend pas pourquoi le mois a été changé.

The screenshot shows a web form for booking a train ticket. At the top, there are five tabs: 'train', 'vol', 'hôtel', 'voiture', and 'séjour'. The 'train' tab is active. Below the tabs, there are four radio buttons for selecting the type of travel: 'Train seul', 'Train & Hôtel', 'Train & Voiture', and 'Train & Hôtel & Voiture'. The 'Train seul' option is selected. The form contains several input fields: 'Au départ de' (Paris), 'Départ (JJ/MM/AAAA)' (14/02/2007), 'à partir de' (12h), 'A destination de' (Berlin), 'Retour (JJ/MM/AAAA)' (14/01/2007), 'à partir de' (12h), 'Adultes' (1), and 'le classe' (2e classe). There are also buttons for 'Réserver votre billet' and 'Consulter les horaires'.

Fig. 1 – Vue extraite du site web <http://www.voyage-sncf.com>. La structure interne du site (i.e. définie par le code HTML) est encadrée en trait pointillé, la perception de l'usager est en trait plein.

Devant ce problème l'usager peut alors entre la requête suivante dans une boîte de dialogue : «*pourquoi jpeux pas rentrer de berlin le 1er janvier ??*»¹. L'usager se retrouve à appeler le champ « A destination de » par son contenu et de l'accoler avec le champ « retour ». Autrement dit, l'usager voit la ligne de composants graphiques (« a destination de », « retour » et « à partir de ») comme un ensemble homogène se rapportant au même concept « informations à propos de la destination ». Pourtant la structuration interne de cette zone dans la page web (i.e. définie par le code HTML visualisable dans le navigateur) ne reflète pas cette organisation sémantique, préférant regrouper « au départ de » et « à destination de » dans une même structure. Autrement dit un bloc se réfère à la *géographie* et l'autre au *temps*.

Un agent assistant n'exploitant que les informations accessibles et essayant de comprendre la requête usager se voit dans l'obligation de rechercher une structure

¹ La question de l'orthographe et de la syntaxe erronées, couramment rencontrée, ne sera pas discutée ici. Elle est traitée, dans une certaine mesure, par le module d'analyse sémantique des requêtes de DAFT (Cf. § 4.1).

correspondant à cet agglomérat et ne peut le trouver puisqu'il n'existe pas. L'agent assistant ne peut pas se reposer sur la représentation formelle de l'application (détaillée ci-dessous dans les sections 5-6), il faut donc introduire une représentation intermédiaire entre l'application et l'agent, représentation qui intègre un point de vue perceptif sur l'application de manière à repérer des structures qui échappent à l'agent.

Il devient primordial de traiter la question de l'acquisition de connaissances sémantiques de haut niveau sur une application destinée au grand public avec la contrainte d'une adaptation générique de l'assistance à n'importe quelle application. C'est cette problématique que nous étudions dans cet article. Nous proposons une méthodologie de construction de modèle pour l'assistance où la dimension perceptive du modèle est traitée de façon transparente et parallèlement à la conception usuelle d'une application à travers un environnement WYSIWYG (What You See Is What You Get).

3 Rôle et acquisition du modèle symbolique de l'application

Le modèle symbolique de l'application est l'interface entre les requêtes de l'utilisateur et l'agent assistant. Il sert de point d'entrée vers l'application pour résoudre les commandes de l'utilisateur et de base de connaissance pour formuler une réponse pertinente aux questions de l'utilisateur : une information d'assistance doit être attachée à chaque élément du modèle correspondant à un élément dans l'application sur lequel l'utilisateur pourrait avoir une question. Le modèle que nous proposons est un modèle exprimant trois dimensions de l'application :

- *Perceptive* (pour nous essentiellement en modalité visuelle) : quels sont les composants visibles avec lesquels un utilisateur peut interagir ;
- *fonctionnelle* : le modèle offre une représentation des différentes fonctionnalités accessibles pour un utilisateur ;
- *temporelle* : il s'agit de garder une trace de l'historique des interactions de l'utilisateur afin de pouvoir se situer dans une tâche, ou de diagnostiquer un échec possible.

La difficulté d'une telle entreprise, c'est l'adaptation de l'agent assistant aux différentes applications à assister. Nous visons une architecture qui soit autant que faire se peut *générique* : la chaîne de traitement doit s'adapter à une nouvelle application sans qu'un effort trop important soit demandé à celui qui la déploie et notamment sans avoir à lui demander une compétence en Traitement Automatique de la Langue Naturelle (TALN) ; de plus cela doit rester vrai pour la création du modèle symbolique représentant l'application. Malheureusement, le modèle médiateur étant fortement dépendant de l'application cible, on ne peut éviter d'avoir recours à un opérateur humain capable de fournir et de valider les informations sémantiques de haut niveau (les annotations) requises pour un bon fonctionnement de l'assistance (Fig. 2).

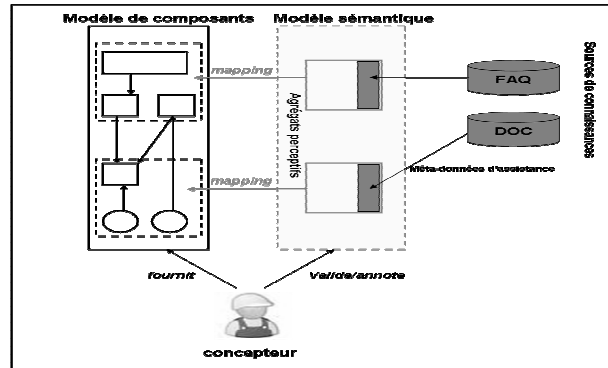


Fig. 2 – principe d'acquisition des connaissances dans le modèle Kiwi.

Plusieurs méthodologies d'acquisition de connaissances sont alors envisageables selon que l'on dispose ou non d'une représentation de l'application, préalable au codage. Nous voyons le processus d'intégration de ces connaissances comme *dynamique et continu* : il ne se limite pas à une phase de spécification au cours de l'élaboration de l'application ou du déploiement de celle-ci. Au contraire, la définition ou l'acquisition de connaissances au cours de cette phase n'est qu'un amorçage du processus d'intégration d'informations pour l'assistance : il s'agit de pointer les structures d'intérêt et les réifier dans le modèle pour pouvoir mapper des connaissances ultérieures vers les bonnes structures.

4 L'environnement Kiwi

4.1 Le projet d'Agent Conversationnel Assistant DAFT

Pour répondre au besoin d'assistance des usagers grand public, le projet DAFT (<http://www.limsi.fr/~jps/research/daft>) mené au LIMSI-CNRS (Sansoulet 2004), se propose de développer des Agents Conversationnels Assistants (ACA), capables d'analyser des requêtes en langue naturelle écrite non contrainte, provenant usagers novices en situation d'utilisation d'applications de complexité croissante (simples applets, pages web, sites actifs, traitements de texte, etc.).

La chaîne de traitement du système d'assistance DAFT a pour objectif à terme de caractériser les composantes génériques de la Fonction d'Assistance et de proposer un noyau d'agent rationnel capable d'assister les usagers novices dans le cas les plus fréquents. L'architecture du système d'assistance DAFT contient classiquement : le module d'analyse sémantique des requêtes en langue naturelle qui construit les requêtes formelles (Le mot 'formel' est employé ici pour distinguer la représentation interne des requêtes de leur forme en langue naturelle), le module de raisonnement sur le modèle de l'application qui retourne une réponse formelle. A son tour, le module de production est chargé d'exprimer cette réponse vers l'utilisateur de manière multimodale

(texte écrit ou parlé ; actions sur l'application; expressions gestuelles d'un personnage virtuel animé, etc.).

4.2 Principe de Kiwi

L'outil Kiwi est un environnement destiné à l'étude et à la construction de modèles symboliques d'assistance, exploitables dans le projet DAFT, cadre dans lequel s'inscrit cette étude. Kiwi est un environnement Internet 'live' permettant à un concepteur grand public (sous classe des usagers grand public qui produit du contenu actif sur le web – blogs, wikis, ...) de construire de manière interactive puis de déployer de petites applications ou services web. L'application déployée hérite ensuite de l'architecture d'assistance fournie par le projet DAFT.

Le principe retenu est de fournir une interface intuitive, de type WYSIWYG : l'interaction avec les composants disponibles se fait par manipulation directe (Drag&Drop). Le concepteur n'a qu'à assembler les composants désirés pour obtenir l'interface. La partie « définition des fonctionnalités » permet de spécifier les comportements et les effets des composants graphiques selon le même principe. La figure 3 donne un aperçu de l'interface Kiwi.

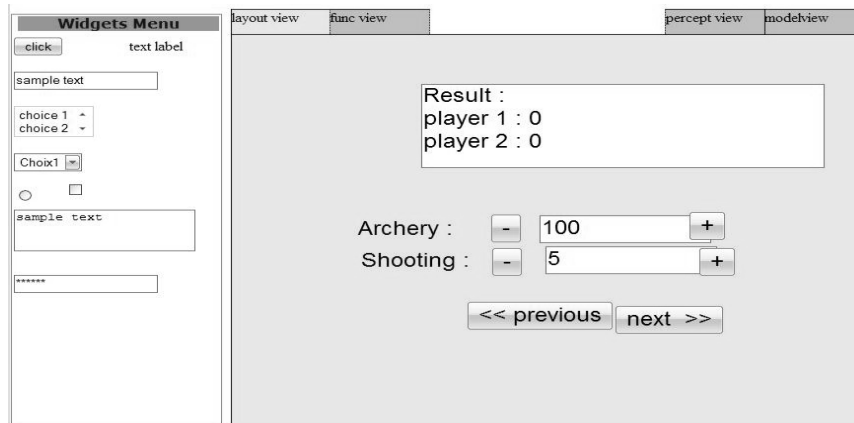


Fig. 3 – Aperçu de l'environnement web Kiwi. A gauche, on peut voir le menu des composants. La partie droite présente différents composants assemblés par un concepteur. Les onglets en haut à droite (layout/func view) permettent de passer de l'organisation graphique à la spécification des fonctionnalités. Ceux de gauche (percept/model view) permettent d'afficher les informations en provenance du serveur.

4.3 Architecture de Kiwi

L'architecture Internet de l'outil Kiwi repose sur une technologie Web 2.0. Elle est composée de trois modules principaux (Cf. fig 4) :

- *Modeleur* : le modeleur à pour objectif d'aboutir à la représentation de l'application. Il s'appuie sur une ontologie décrivant l'ensemble des composants utilisables. Parmi

ces composants, on distingue les composants perceptifs (widgets, évènements graphiques -fermeture d'un menu, clic sur un widget-), les composants structurels (structures de données, opérations sur les structures de données), les composants agrégés (fonctions, agrégats perceptifs – Cf. fig 1). L'ontologie décrit aussi les relations existant entre ces différents composants et leur structure interne (opérations accessibles, équivalence de composants). Cette ontologie est utilisée pour fournir à l'environnement de conception les informations à afficher à destination du concepteur. Cette ontologie est décrite dans un formalisme propre en langage Mathematica de Wolfram Research (elle peut toutefois être exprimée en XML, avec des équivalences en OWL).

- *Client* : le client se charge de fournir l'interface WYSIWYG pour le concepteur. Ses rôles sont de laisser le concepteur disposer des composants graphiques et d'afficher les retours de l'agent modelleur présent sur le serveur (évidence d'un groupe perceptif à annoter et demande d'annotation). Le client est écrit en Ajax (HTML, javascript et CSS).
- *Serveur* : le serveur est l'hôte de l'agent modelleur. L'agent modelleur reçoit les informations envoyées par le client, notamment les composants utilisés et leur placement sur le 'layout'. Sur cette base, il crée un modèle structurel naïf (qui ne contient que des informations liées au placement des composants) et tente d'inférer des agrégats perceptifs sur la base d'heuristiques perceptives. Les agrégats synthétisés sont alors renvoyés au concepteur pour annotation et validation. Le serveur est un serveur Apache Tomcat utilisant la technologie servlet pour faire le lien entre le client et l'agent modelleur.

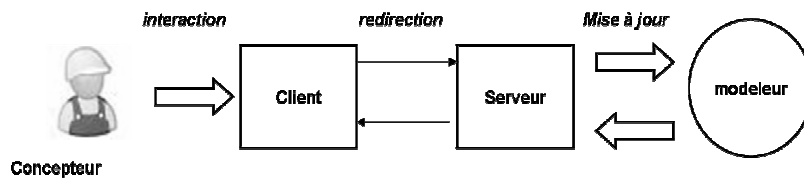


Fig. 4 – Architecture générale de l'environnement Kiwi.

Pour résumer, l'ontologie fournit les outils permettant d'aboutir à la structure du modèle, elle donne une syntaxe permettant de garantir une cohérence entre les différents modèles ainsi qu'une base finie d'attributs partagées par les composants. L'apport de connaissance se fait en trois étapes : 1) l'acquisition d'un modèle structurel naïf (à la charge d'un concepteur ou d'un processus automatique si l'application existe déjà), 2) la création de groupes perceptifs (à la charge de l'agent modelleur), 3) l'annotation des groupes trouvés (à la charge du concepteur).

On utilise ici l'expertise du concepteur envers son application pour apporter la sémantique de haut niveau (apport en langue naturelle) sur des structures pouvant être rapportées à des objets d'une représentation formelle.

La section suivante montre sur l'exemple du site de la SNCF, (Cf. fig. 1) le modèle auquel ce processus aboutit.

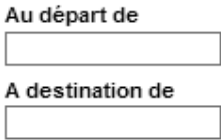
5 Etude de cas : voyage-sncf

5.1 Modèle structurel

Nous reprenons l'exemple de la section 2 pour illustrer les différentes étapes de l'enrichissement du modèle. Pour rappel, il s'agit de la partie du site web de la SNCF permettant à un usager de rentrer les informations lui permettant de réserver un billet de train.

Le modèle structurel simple peut être acquis à partir de la structure DOM de la zone correspondant à la capture d'écran. Pour aboutir à la structure voulue on construit une liste dans laquelle sont placées tous les composants visibles à l'écran. Les balises de type conteneur (<div>, <table>, ...) sont ajoutées à la liste uniquement si la propriété CSS 'border' est spécifiée. Le client enregistre également les informations liées au placement du composant, sa taille, son type et sa couleur. Le tableau 1 montre la structure initiale et la structure extraite.

Table 1. A gauche : structure issue du code HTML de la page voyage-sncf. A droite, structure associée, extraite par Kiwi, en vue de construire le modèle structurel naïf.

Structure DOM (HTML de la page)	Structure Extraite
<pre> ... <div id="od_train"> <p class="input required" id="fi_ORIGIN_CITY"> <label for="ORIGIN_CITY" title="départ"> Au départ de </label> <input name="ORIGIN_CITY" id="ORIGIN_CITY" tabindex="1" type="text" /> </p> <p class="input required" id="fi_DESTINATION_CITY"> <label for="DESTINATION_CITY" title="destination "> A destination de </label> <input name="DESTINATION_CITY" id="DESTINATION_CITY" tabindex="2" type="text" /> </p> </div> ... </pre> <p>Remarque : les balises <div> et <p> du code HTML originel ont été négligées car elles ne redéfinissent pas la propriété border du CSS.</p>	<pre> L = { label#1, ORIGIN_CITY, label#2, DESTINATION_CITY } </pre> 

5.2 Agrégation perceptive

Des études de psychologie (Wertheimer 23) ont pu mettre cinq critères sur lesquels est basée notre vision pour regrouper des formes dans des agrégats

perceptifs : « proximity, similarity, good continuation, symmetry, closure ». Dans le domaine de l'informatique, les algorithmes proposés pour implémenter ce phénomène ont surtout mis l'accent sur les trois premiers critères (Thorisson 94).

Nous utilisons un algorithme proche de Thorisson à la différence près que nous l'appliquons sur des composants d'interfaces graphiques qui sont porteurs d'une sémantique influant sur les possibles regroupement. L'algorithme en tient compte en utilisant des heuristiques tirant partie de ses spécificités.

Algorithme : Calcul des agregats perceptifs

Debut

```
V0 := {liste des composants}
V1 := calculsVoisins(V0)
i :=1
tant que Vi != Vi-1 faire
  pour tout vj dans Vi faire
    vj := appliqueHeuristique(CRITERE, vj)
    vj := appliquePattern(CRITERE, vj)
  fpour
ftq
fin
```

Le calcul des voisins `calculsVoisins` implémente le critère de proximité ; l'application des heuristiques `appliqueHeuristique` implémente le critère de similarité et le 'pattern' `appliquePattern` celui de bonne continuation. Les critères passés aux méthodes sont ceux apportés par les connaissances sur les composants : type, ou des primitives perceptives : couleur, taille. Le pattern consiste à chercher dans un groupe une répétition d'un motif de composants avec des priorités plus hautes pour certains types de composants précis (boutons, boutons radio, case à cocher, liens hypertextes). Deux types d'opérations sont alors possibles :

- *la fusion* : deux composants sont traités comme une même entité cognitive ;
- *l'agrégation* : deux composants sont regroupés dans un groupe.

6 Résultats

6.1 Structure générée

La table 2 présente les résultats obtenus pour cet exemple. Elle met en comparaison les structures synthétisées par l'algorithme lorsqu'un équivalent a pu être exhibé dans la structure DOM et lorsque une annotation sémantique a été considérée comme pertinente par le concepteur.

L'exemple (pris dans une application réelle) reste simple mais permet déjà de montrer des phénomènes intéressants : le fait de fusionner des composants entre eux est pertinent (c'est d'ailleurs fait dans le code source de l'application mais de façon hétérogène à la perception des usagers). Les agrégations de composants sont

pertinentes elles aussi dans la mesure où il a été possible d'attribuer un 'label' pour chacune des agrégations synthétisées automatiquement par Kiwi. Par contre, il n'a pas été possible de retrouver ces structures dans le code HTML. Inversement, certaines structures présentes dans le code HTML ne se retrouvent pas dans les agrégats synthétisés (c'est la source de l'erreur cognitive décrite à la section 2).

Table 2. Équivalence des structures DOM et des structures perceptives synthétisées. La colonne « occurrences » indique le nombre des structures synthétisées par Kiwi, « équivalence dans le fichier HTML » indique le nombre de structures synthétisées pour lesquelles une balise DOM a pu être trouvée (colonne « balises DOM équivalentes »). La colonne « structures annotées » indique le nombre de ces structures synthétisées pour lesquelles un label a pu être trouvé.

Opérations appliquées	occurrences	équivalence dans le fichier HTML	balises DOM équivalentes	structures annotées
Fusion de composants	13	13	<p>, , <label>	13
Agrégations de composants	5	3	<div>, 	5

Ces résultats portant sur un exemple, ne doivent pas être généralisés mais cela illustre un phénomène bien particulier : l'hétérogénéité des représentations formelles pour les des structures perceptives (visuelles) ne permet pas de se baser sur le seul code source de l'application pour anticiper les constructions perceptives, ce qui nous conforte dans notre volonté de porter une attention plus grande aux phénomènes perceptifs, problématique que nous évoquons à la section suivante.

6.2 Application à l'assistance

On peut le constater sur l'exemple du site de la SNCF, une telle méthodologie permet de faire ressortir des groupements d'objets porteurs d'une sémantique qui n'était pas clairement représentée dans la structure initiale et qui n'était pas exploitable par un agent assistant. Comment l'agent assistant peut-il exploiter ces informations pour augmenter sa compétence en assistance ? Il faut alors considérer la représentation perceptive comme complémentaire de la représentation classique qui permet déjà de répondre à une partie des requêtes. Sur l'exemple cité ci-dessus, la compétence de l'agent assistant se trouve t-elle améliorée ? La requête de l'utilisateur était : «*pourquoi jpeux pas rentrer de berlin le 1er janvier ??*». Dans cet article nous ne pouvons pas développer, même schématiquement, les techniques d'extraction sémantique mises en œuvre dans le projet DAFT (Cf. Sansonnet 04). Nous dirons simplement que pour la phrase ci-dessus, l'agent assistant reçoit une requête formelle ayant pour forme simplifiée :

```
WHY-FAIL (
  AGENT="user",
  ACTION="rentrer-de" (AGENT="user", LOC="#berlin", DATE="jan,1")
)
```

L'analyse sémantique renvoie deux informations importantes : l'utilisateur ne peut pas faire quelque chose (**WHY-FAIL**) et ce quelque chose est caractérisé par le label "#berlin" sur lequel est associé l'action "rentrer-de". La première partie permet de définir une stratégie dans la recherche de la référence : il existe un objet qui est contraint et cette contrainte est liée à la deuxième partie de la requête. Ces contraintes sont récupérables, elles se trouvent dans les instructions **IF** des scripts impliquant des identifiants d'objets se trouvant représentés à l'écran. Par contre, l'action "rentrer-de"[.. LOC=".."] ne peut être présente dans la structure de l'application, car elle n'est pas inhérente à un composant mais elle est consubstantielle d'un contexte linguistique, donc hors du champ d'étude.

Sur cette base d'extraction sémantique, l'agent doit rechercher un composant doté d'une contrainte et relatif à `LOC="#berlin"`. `LOC="#berlin"` est aisé à retrouver puisqu'il est tapé dans un champ texte (`textfield DHTML`), par contre ce champ de texte n'est doté d'aucune restriction. Et si l'agent tente de remonter la structure syntaxique pour retrouver une contrainte, il échoue : le contexte du champ de texte est celui que le concepteur a choisi, c'est-à-dire de regrouper les informations liées à la destination des trains d'un côté et les informations liées à la date du voyage de manière séparée.

Le modèle perceptuel apporte l'unification de ces deux contextes puisqu'il propose de voir les composants sous l'angle de l'unité sémantique opposant l'arrivée et le départ. Le composant portant sur la contrainte se retrouve dans le même contexte que le composant portant le marqueur `LOC="#berlin"` permettant à l'agent de formuler une réponse incorporant la contrainte `dateRetour > dateArrivée`. On pourra arguer, que dans le cadre de cet exemple, un simple examen des traces d'exécution aurait permis de constater que la dernière action de l'utilisateur aura été d'interagir avec un composant sur lequel une contrainte a été définie. Malheureusement il n'existe aucune certitude que la requête se rapportait effectivement à cette action, la seule certitude sur laquelle se baser est le compréhension sémantique de la requête de l'utilisateur, ce qui justifie notre approche.

7 Discussion

L'assistance étant une question vaste, nous avons mis l'accent sur la problématique du fossé référentiel comme étant source de difficultés pour traiter les requêtes d'un usager novice en pointant les défauts qu'avaient les modélisations actuelles dont on pourrait se servir comme base de connaissances. A travers l'exemple présenté ici, nous avons essayé de montrer l'importance de prendre en compte ce phénomène. Toutefois nous ne pouvons pour l'instant caractériser la fréquence à laquelle il apparaît dans une situation réelle d'assistance. Cela nous ouvre des perspectives d'évaluations intéressantes car elles reposent sur des phénomènes perceptifs assez fins que notre modèle cherche grossièrement à capturer.

Il se pose aussi la question de l'intégration du processus de production de l'assistance dans le cycle de conception et de développement d'un logiciel. Si les exemples que nous développons se prêtent bien à une application sur des web-

services, nous ne pouvons pas prétendre à un passage à l'échelle des grands logiciels monolithiques du commerce (par ex. Office, FireFox, ...). Il est à craindre que le processus que nous avons décrit engendre un nombre de composants trop important pour qu'il puisse être géré de façon efficace par un concepteur/annotateur humain.

Toutefois, la projection de cette méthodologie de conception sur l'espace web nous permet d'entrevoir des possibilités intéressantes en matière de compositions et d'enrichissement de composants : les développements récents et l'essor du web social pousse à transformer les usagers novices en *créateurs de contenu*. Et ce contenu doit pouvoir être médié et accessible aux autres usagers. La masse des créations résultant du nombre impressionnant d'utilisateurs justifie largement notre approche d'automatiser les processus de production d'assistance. On peut alors concevoir une base de composants intégrant des données d'assistance s'enrichissant au fur et à mesure de l'interaction que les usagers novices ont avec eux.

Références

- Allen J.F., Byron D.K., Dzikovska M.O., Fergusson G., Galescu L., and Stent A., Towards conversational Human-Computer Interaction, AI magazine, 2001.
- Allen, J.F. et al, "The TRAINS Project: A Case Study in Defining a Conversational Planning Agent", Journal of Experimental and Theoretical AI, 1995.
- Antonio Capobianco, Noëlle Carbonell, *Conception d'aides en ligne pour le grand public : défis et propositions*, Actes du 8ème Colloque Francophone, Ergonomie et Informatique Avancée (ERGO-IA'2002), Biarritz, 8-10 Octobre 2002, Bidart (64210) :ESTIA & ESTIA Innovation, pp. 309-335.
- Capobianco, A., Carbonell, N. (2000). *Aide en ligne contextuelle : stratégies d'experts humains*. In D. Scapin, E. Vergison (Eds.), Actes ERGO-IHM'2000, Biarritz, octobre 2000. Bidart, 64210 : CRT ILS & ESTIA, pp. 48-55.
- Carenini G., Moore J. D., Generating explanations in context. Int. workshop on Intelligent User Interfaces, ACM Press pp 175-182, Orlando, 1993
- Carroll, J.M., Smith-Kerker, P.L., Ford J.R., Mazur-Rimetz, S.A. (1987). *The minimal manual*. Human-Computer Interaction, 3(2), 123-153.
- Fisher, G., Lemke, A., Schwab, T. (1985). Knowledge-based help systems. In L. Borman, B. Curtis (Eds.), *Proc. CHI'85 (International Conference on Human Factors in Computing Systems)*, San Francisco, CA, April 1985. New York: ACM Press & Addison Wesley, pp. 161-167.
- Grady Booch, James Rumbaugh, Ivar Jacobson (2000). *Le guide de l'utilisateur UML*, ISBN 2212091036
- Leray, D, J-P. Sansonnet, *Librairie de Widgets Dialogiques pour un Agent Conversationnel Assistant*, Short paper at IHM05, Toulouse, 27-30 sept 2005.
- Leray, D, J-P. Sansonnet, *Ordinary user oriented model construction for Assisting Conversational Agents*, International Workshop on Communication between Human and Artificial Agents, CHAA'06 at IEEE-WIC-ACM Conference on Intelligent Agent Technology, Hong Kong, dec 2006
- Mack, R.L., Lewis, C., Carroll, J.M. (1983). *Learning to use word processors: problems and prospects*. In W.B. Croft (Ed.), *ACM Transactions in office information systems (TOIS)*, 1(3),254-271. (from [16])

Acquisition de connaissances perceptives pour un agent assistant

- Pilkington, R.M. (1992). *Question-answering for intelligent on-line help: the process of intelligent responding*. Cognitive Science, 16, (4), 455-491.
- Roestler, A.W., McLellan, S.G. (1995). *What help do users need? Taxonomies for on-line information needs and access methods*. Proc. CHI'95 (International Conference on Human Factors in Computing Systems), Denver, CO, May 1995. New York: ACM Press & Addison Wesley, pp. 437-441.
- Sabouret N., Étude de modèles de représentations, de requêtes et de raisonnement sur le fonctionnement des composants actifs pour l'interaction homme-machine, Thèse de l'Université de Paris XI, 2002.
- Sansonnet J-P. Composants dialogiques génériques : perspectives et méthodes pour une approche intégrée des outils assistants langagiers et de la programmation objet, Conférence LMO 04 Lille, 2004.
- Sansonnet J-P., Leguern K., Martin J-C., Une architecture médiateur pour des agents conversationnels animés, Workshop WACA'01, Grenoble, 2005.
- Thorisson K.R., Simulated Perceptual Grouping: An Application to Human-Computer Interaction., Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society, 1994. Atlanta, Georgia, August 13-16, 876-881.
- Vouligny, L. and Robert, J. 2005. Online help system design based on the situated action theory. In Proceedings of the 2005 Latin American Conference on Human-Computer interaction (Cuernavaca, Mexico, October 23 - 26, 2005
- Wertheimer, M. (1923). Untersuchungen zur Lehre von der Gestalt. Psychologische Forschung, 4, 301-50. Translation in W. D. Ellis (ed.), A Source Book of Gestalt Psychology. New York: H. B. J., 1938.