Interaction Multimodale et Monde Virtuel

Nicolas Saunier
Encadrant: Alain Grumbach

July 4, 2001

Contents

1 Présentation du sujet **4**
- 1.1 Introduction 4
- 1.2 Spécificité et complémentarité du langage et des interactions déïctiques 5
- 1.3 La problématique de la représentation du monde 7

2 Etude d'un cas, Ulysse **9**
- 2.1 Description 9
- 2.2 Evaluation 11
- 2.3 Algorithme de Résolution de Références 12
  - 2.3.1 Description de l'algorithme 13
  - 2.3.2 Comparaison avec d'autres algorithmes 14

3 Implémentation **16**
- 3.1 Présentation du système 16
- 3.2 Architecture 18
- 3.3 Critiques et améliorations envisagées 21

4 Conclusion **24**

5 Remerciements **24**

A Analyse du programmes **27**
- A.1 Description du monde 27
  - A.1.1 Classe World 27
  - A.1.2 Classe MyUniverse 30
  - A.1.3 Classe Landmark 31
  - A.1.4 Classe PickMouse 32
- A.2 Interface graphique 32
  - A.2.1 Classe WinOrder 32
  - A.2.2 Classe WinAnswer 33
  - A.2.3 Classe Help 34

1 Présentation du sujet

1.1 Introduction

Les mondes virtuels trouvent aujourd'hui de nombreuses applications dans l'industrie, en particulier pour la simulation, pour la CFAO (Conception Assistée par Ordinateur), mais aussi de façon tout à fait naturelle dans le domaine des jeux vidéos. Toutefois, l'interaction avec ces mondes virtuels pose encore de nombreux problèmes, par la variété des approches et des outils mis en oeuvre: gants, visiocasques, souris, outils de pointage, reconnaissance des mouvements, tapis roulants, microphones... il existe de nombreux outils d'interactions, plus ou moins adaptés suivant les tâches concernées. Ces outils font appel à différents modes de communication, et contribuent ainsi à créer des interfaces multimodales.

Figure 1: Ensemble visiocasque, gants et microphone.

Les interactions dans un monde virtuel sont de différentes natures. Elles concernent au premier plan la navigation dans ce monde. Tout aussi importante est la manipulation d'objets du monde. Pour cela, il faut être capable de désigner un objet, de s'en saisir. Il existe pour cela de nombreux outils, en particulier de pointage, des gants permettant de manipuler l'objet de façon naturel comme si il était réel. L'utilisateur va ainsi faire référence, dans le cours de ses actions, à des objets, et le système devra être capable de suivre les objets manipulés.

Cependant, on pense en général plus rarement à l'utilisation du langage dans les interfaces de mondes virtuels, et pourtant quoi de plus naturel que de dire 'Ouvre la porte de cette voiture' ! Dans cet ordre typique, la parole sera accompagnée d'un geste pour désigner la porte en question, ou d'un

autre outil de pointage permettant de la désigner. Il est indispensable de réfléchir et de pouvoir intégrer ces différents modes de communication afin de construire des interfaces efficaces, et naturelles pour l'être humain. Nous nous intéressons en particulier dans ce travail au langage et aux interactions déïctiques[1].

La prise d'importance des mondes virtuels a donné naissance et entraîné le développement de domaines nouveaux, comme par exemple la Théorie de l'Information Spatiale, thème sur lequel se tiennent les conférences bi-annuelles COSIT (Conference On Spatial Information Theory, cf [22]). Des chercheurs comme Barbara Tversky (cf [21] et [20]) s'intéressent aux capacités humaines de résolution de problèmes d'orientation et aux représentations cognitives des informations spatiales. D'autre part, de nombreux organismes, industriels, ont développé des projets utilisant le langage comme forme d'interaction. On peut citer Persona (cf [5] et [6]), le projet de Microsoft, dans lequel un perroquet virtuel nommé Peedy commande une base de données de disques et pent ensuite les jouer, Nautilus (cf [11], [12] et [13]), un projet de L'US Navy, permettant la navigation à la voix dans un sous-marin, Diverse (cf [7], [8], [9] et [10]), du Swedish Institute of Computer Science, dans lequel un agent vous accompagne et répond à vos requêtes, Ulysse(cf [17]; [18] et [19]), projet de Pierre Nugues, à l'université de Caen, permettant de se déplacer à la voix dans un monde virtuel, et Wordseye de ATT (cf [3] et [4]), dont le but est de convertir des textes en langue anglaise en images 3D supposées les illustrer. Ces projets explorent les possiblités offertes par l'ajout d'interfaces en langage naturel et nous étudierons en particulier le projet Ulysse dans la partie 2 de ce mémoire.

Mais nous allons dans un premier temps nous intéresser aux caractéristiques du langage et des interactions déïctiques, pour définir clairement la problématique de notre travail. Dans la partie 2, comme nous venons de le dire, nous verrons quelles sont les réponses apportées par le projet Ulysse. Enfin nous présenterons un exemple concret que nous avons développé, dans la partie 3, dont le but est d'illustrer et de répondre aux questions soulevées par notre sujet.

1.2 Spécificité et complémentarité du langage et des interactions déïctiques

Les caractéristiques du langage sont nombreuses, en particulier liées à sa fonction de référence symbolique. Il permet tout d'abord des raisonnements,

[1]Ce type d'interaction comprend en particulier les désignations par pointage. La deixis est définie précisément dans la partie 1.2

entre symboles désignant des objets du monde, mais aussi à des niveaux d'abstraction plus élevés. Il permet aussi de faire référence à des objets du monde, en particulier lorsqu'ils ne sont pas visibles, et cela de différentes façons, directe (étiquette unique pour un objet), indirecte ('la voiture rouge'), et par anaphore (utilisation de pronoms par exemple) (cf [14]). Il permet de désigner simplement un ensemble d'objets ('les voitures rouges') et de résumer un ensemble d'actions ('va chercher le journal !' = prendre la voiture rouge, aller au kiosque, acheter le journal, et le ramener).

D'autre part, Huls et al. [15] citent dans leur article la définition suivante de la deixis (Lyons, 1977): la localisation et l'identification des personnes, objets, évènements, processus et activités dont on parle ou auxquels on fait référence, en relation avec le contexte spatio-temporel créé et alimenté par l'acte d'énonciation et la participation d'un locuteur et d'au moins un interlocuteur. Huls et al. distinguent 3 types de deixis: personnelle, temporelle et spatiale. La deixis personnelle implique des pronoms à la première et seconde personne (je, nous, tu...). La deixis temporelle se réalise par le système des temps d'un langage ('il vit à Paris'), et par des modifieurs temporels ('dans une heure'). La deixis temporelle relie le temps du discours aux relations exprimées dans les énoncés. Enfin, la deixis spatiale implique des démonstratifs et d'autres références produites en combinaison avec des mouvements de pointage (cette — voiture, où — représente le mouvement effectué par le locuteur). Nous allons surtout nous intéresser à la deixis spatiale, et comment la résoudre dans des interactions déïctiques.

Les expressions déïctiques peuvent être mélangées avec des anaphores. A la différence des expressions déïctiques, les anaphores peuvent être interprétées indépendamment du contexte spatio-temporel de la discussion. Leur interprétation dépend uniquement des expressions linguistiques qui les précèdent dans le discours (exemple: 'Existe-t-il une voiture rouge ? Vas-y !'). Cependant, dans de nombreux langages, les mots utilisés dans les expressions déïctiques et les anaphores sont les mêmes, ce qui cause problème aux systèmes d'analyse du langage naturel.

Ces modes sont spécifiques et complémentaires. Spécifiques, et plus ou moins adaptés: il sera plus simple de dire 'les voitures rouges', que de cliquer sur toutes les voitures rouges. Ces modes sont aussi complémentaires, et peuvent se combiner, comme dans l'ordre 'mets-ça ici', où 'ça' peut être une anaphore, faisant référence à un objet précédemment cité dans le discours, et 'ici' être désigné par un geste, un clic de souris.

Nous présenterons le modèle de Huls et al., un modèle unique qui leur permet de résoudre les références déïctiques et anaphoriques, dans la partie 2.3.

1.3 La problématique de la représentation du monde

Mais, pour pouvoir répondre à ces questions, pouvoir résoudre les anaphores et les expressions déïctiques par exemple, il faut des structures adaptées . Tout dépend et découlera d'un choix d'une représentation du monde, qui se manifestera dans l'implémentation que l'on en fera. Cette structure découpée entre une représentation du monde, et des modules accédant cette représentation pour leurs besoins, comme la résolution de références, mais aussi le rendu 3D ou les interactions nous ont fait penser à une structure client-serveur, illustrée par le schéma 2.

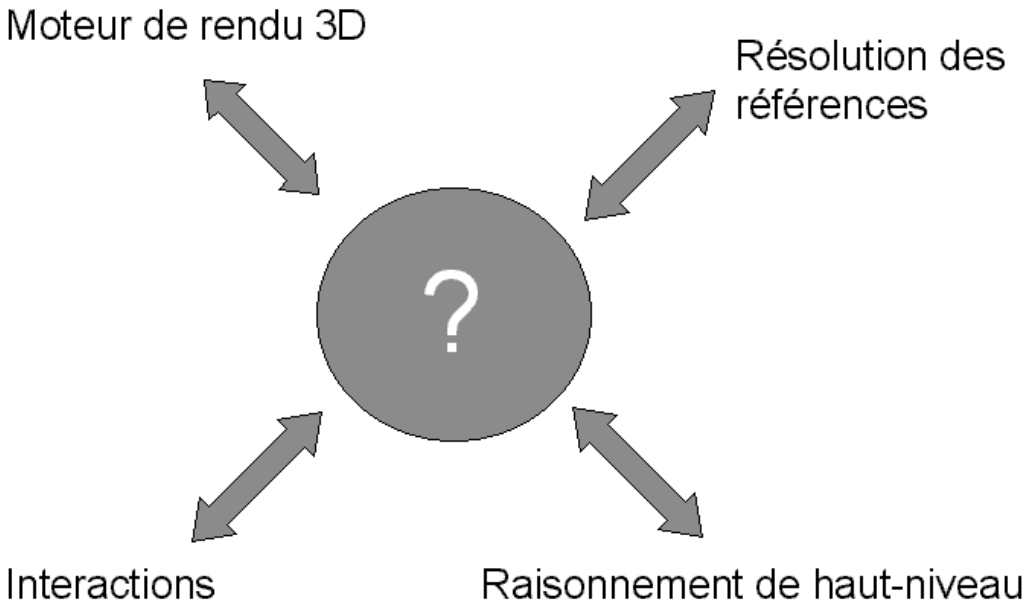

Figure 2: Modèle client-serveur de représentation du monde. Le '?' désigne la représentation à déterminer

Dans cette architecture, les clients seront par exemple le moteur de rendu 3D, dont les exigences en matière de format des données sont importantes, afin de pouvoir effectuer des calculs temps réel et générer un monde répondant rapidement aux interactions des utilisateurs. De même, on peut penser à des besoins cognitifs de plus haut niveau, demandant des capacités de raisonnement, et nécessitant des connaissances sur le monde, par exemple pour résoudre l'exemple précédemment cité: 'va chercher le journal !' = prendre la voiture rouge, aller au kiosque, acheter le journal, et le ramener. Le module chargé de ce travail devra accéder à des informations concernant la localisation du kiosque par exemple, mais il sera surtout nécessaire d'ajouter

des connaissances d'ordre général sur le monde, pour pouvoir déduire par exemple que le journal s'achète dans un kiosque. Cette représentation doit ainsi être en mesure de fournir les données demandées par tous les clients, et cela impose des contraintes importantes sur la représentation, ce qui constitue la problématique fondamentale à la base de ce travail.

2 Etude d'un cas, Ulysse

Comme nous en avons déjà parlé dans l'introduction, il existe de nombreux projets d'organismes variés concernant les interactions multimodales, mettant en jeu en particulier le langage, avec les mondes virtuels. Le projet Diverse du SICS (cf [7], [8], [9] et [10]) est l'un des plus intéressants. Il permet en particulier de manipuler des objets du monde, de les repeindre par exemple, et utilise un algorithme de résolution de références avec de nombreuses sources de connaissances, malheureusement non-détaillées dans les documents auquels j'ai pu avoir accès. Mais nous allons nous intéresser à un projet français dirigé par Pierre Nugues à l'université de Caen (cf [17]; [18] et [19]).

2.1 Description

Ulysse est un agent conversationnel permettant de naviguer à la voix dans des mondes virtuels VRML. La première version d'Ulysse a été réalisée avec DIVE du SICS et le système d'exploitation Unix. Quant à nous, nous n'avons pu étudier que la version VRML d'Ulysse, une adaptation simplifiée en VRML, Prolog et Java.

Comment se sert-on d'Ulysse ? On peut tout d'abord se déplacer avec les commandes par défaut dans le monde virtuel, ce qui est toujours possible dans un monde VRML visualisé grâce à un plug-in dans un navigateur internet. D'autre part, on peut lui donner des ordres en langage naturel par l'intermédiaire d'une fenêtre prévue à cet effet comme on peut le voir dans la figure 4. Détaillons ici les ordres de déplacement:

Ordres dépourvus de références : avance, recule, tourne à gauche, tourne à droite

Ordres faisant référence à des objets du monde : va vers / devant / derrière / à gauche / à droite de [objet du monde]

Il est à noter que la version VRML d'Ulysse ne traite que des références directes. Enfin, il est possible de rajouter des étiquettes à des objets existant déjà dans le monde, par une interface présentée dans la figure 5. On se place devant puis derrière l'objet que l'on veut nommer, afin de donner les coordonnées au système, et on choisit l'étiquette, avec deux adjectifs pour qualifier l'objet. Les étiquettes de ces objets nouvellement référencés peuvent ensuite être utilisées dans des ordres.

Ulysse est conçu sur le principe d'un système client-serveur, qui correspond justement au modèle que nous avons introduit au chapitre 1.3. L'architecture du système est décrite dans la figure 6.

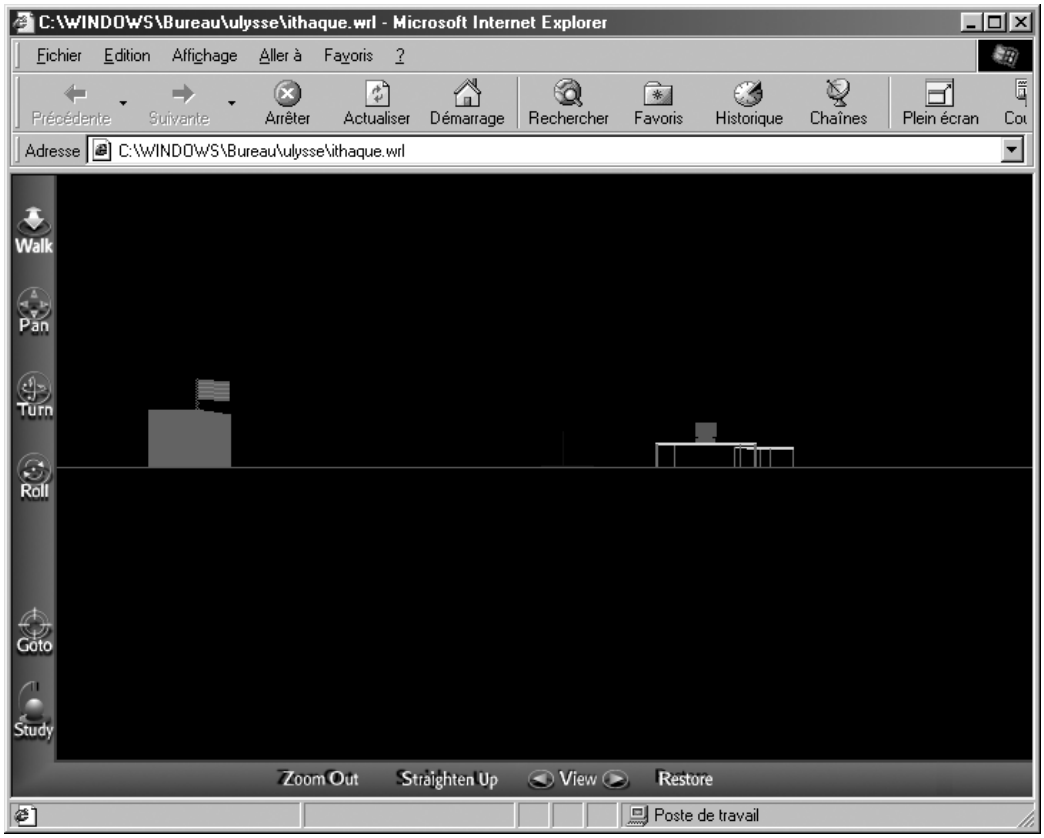

Figure 3: Monde d'Ulysse, visualisé grâce à un plug-in dans un navigateur internet.

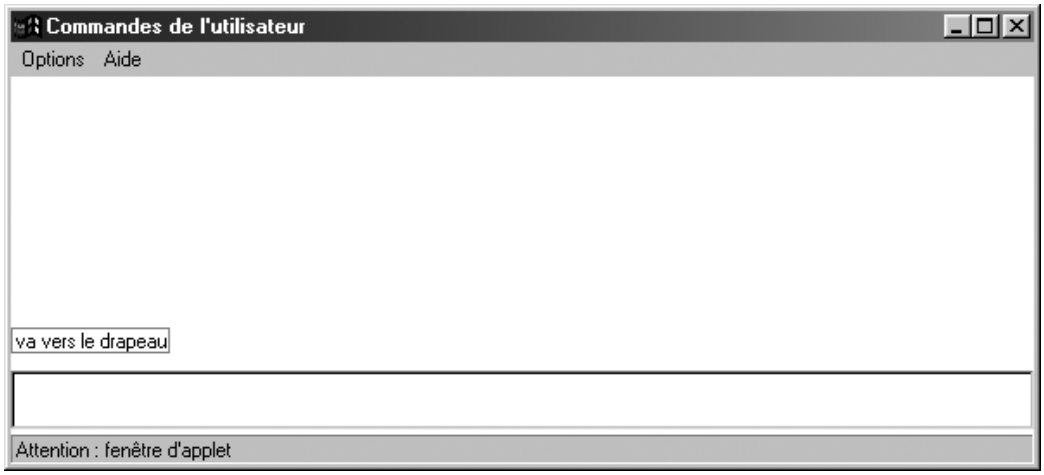

Figure 4: Interface de saisie des commandes en langage naturel.

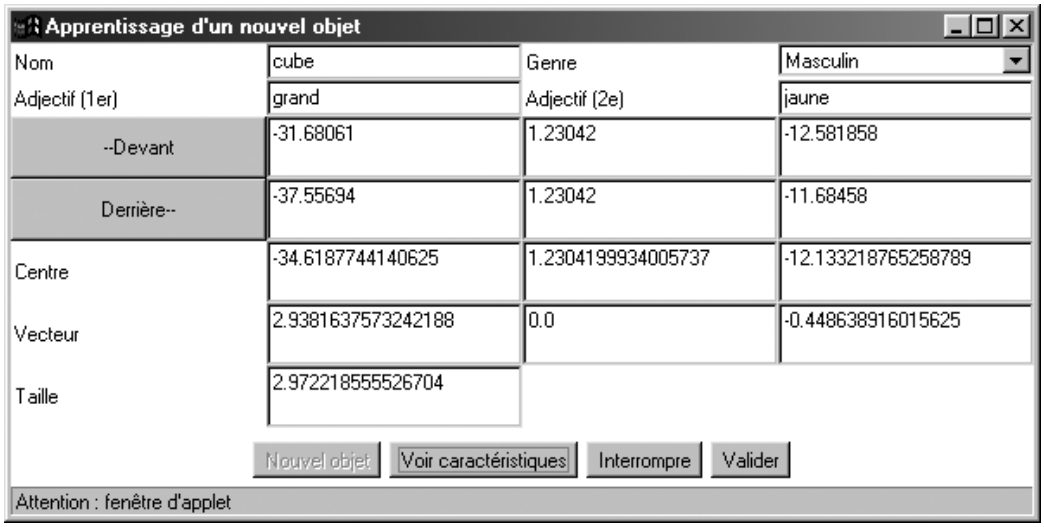

Figure 5: Interface de nommage des objets du monde.

Le moteur linguistique en prolog : il est constitué de plusieurs fichiers.

Le monde VRML : nommé Ithaque.

Une applet java : elle sert d'interface au monde VRML.

Le monde virtuel est lu par un client VRML, comme un navigateur internet avec un plug-in approprié. Le client est constitué d'une fenêtre d'entrée pour taper des commandes, afficher la position de l'utilisateur dans le monde, déclarer des nouveaux objets. Le serveur reçoit de l'utilisateur des commandes de navigation en français. Ces commandes sont interprétées par un moteur d'analyse linguistique en Prolog. Le serveur calcule le mouvement correspondant et renvoie au monde virtuel des ordres de déplacement. Le client déplace alors le point de vue de l'utilisateur dans le monde grâce à une interface Java incorporée au monde virtuel.

2.2 Evaluation

Le système Ulysse est intéressant pour de nombreuses raisons, la première étant que c'est un des premiers systèmes de navigation dans un monde virtuel avec une interface utilisant le langage naturel qui fonctionne. Les déplacements marchent bien, sont fluides et précis, en particulier en ce qui concerne les positions finales par rapport à la destination. Comme nous avons pu le voir, son architecture suit le modèle client-serveur présenté. Mais quelle est la

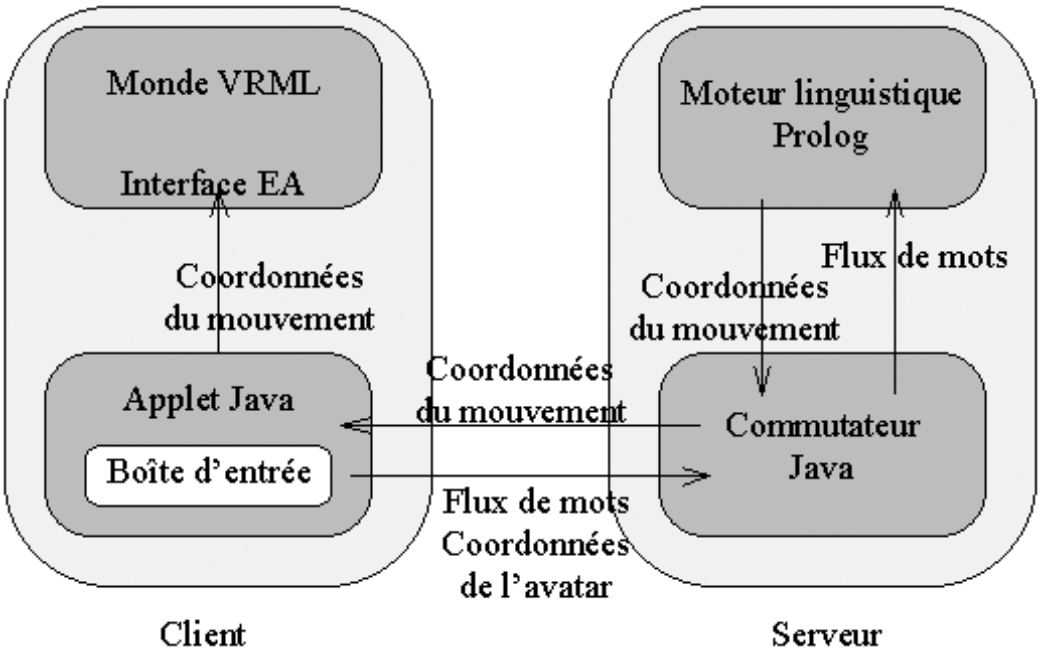

Figure 6: Architecture de la version VRML d'Ulysse.

représentation du monde utilisée dans cet exemple ? Il s'agit d'objets VRML décrits de façon symbolique dans un fichier prolog. Ce fichier effectue donc la correspondance entre les objets du monde et leurs étiquettes, qui seront utilisées dans des ordres et traitées par le moteur linguistique.

Il faut de plus reconnaître la performance du moteur linguistique, même si il y a parfois des problèmes avec le système d'adjectifs. On rencontre aussi des problèmes de nommage, en particulier lorsque l'on essaie de rentrer la position de l'objet nouvellement référencé. Enfin, cette version d'Ulysse ne comprend pas de système de résolution des anaphores du langage, et encore moins d'expressions déïctiques. Nous allons voir un Algorithme de Résolution de Références dans la partie 2.3.

2.3 Algorithme de Résolution de Références

L'algotithme utilisé dans la version complète d'Ulysse, développé dans l'environnement DIVE, a été mis au point par Huls et al. [15]. Nous allons le décrire maintenant.

2.3.1 Description de l'algorithme

Le but de cet algorithme est donc de résoudre les références dans une interface multimodale, aussi bien les anaphores que les expressions déïctiques. Il a été développé dans le cadre d'un projet intitulé EDWARD, dont l'une des applications concerne la manipulation de la représentation graphique d'un système de fichiers par des menus, le langage écrit ou parlé, et les gestes de l'utilisateur. EDWARD répond en langage naturel, parlé ou écrit, et graphiquement.

Pour résoudre les références, l'algorithme utilise deux sources de connaissance. Tout d'abord, il y a une base de connaissances générales, implémentée par un réseau sémantique. Les noeuds du réseau sont des instances de classes d'entités ou de relations. Par exemple, la classe 'personne' a pour sous-classe 'femme' et 'homme'. En outre, on lie un période de validité à chaque instance du réseau. Le fait de créer des classe d'objets permet de limiter les recherches à certaines classes. Cette base de connaissance sert pour les questions d'ordre général.

La seconde source de connaissance est constituée par un modèle du contexte de l'interaction. Pour cela, on introduit les notions de pertinence et de facteur contextuel (noté CF). L'idée à la base de ce modèle est que, à chaque recherche d'une référence, on va mesurer la pertinence de tous les objets susceptibles de résoudre la référence. Or, la pertinence d'un objet dépend de nombreux facteurs, d'importance variable. Par exemple, la récence de la mention d'un objet dans le discours influe sur sa pertinence. Pour mesurer cette pertinence, on va introduire des facteurs contextuels. Un CF est défini par l'élément du contexte qui le déclenche, et sa valeur au cours du temps, dont un exemple est donné dans le tableau 1. Il y a différents types de de CFs, linguistiques (on peut ensuite détailler suivant l'importance de la mention, si l'objet est le sujet, ou juste dans une expression prépositionnelle), perceptuels (comme le pointage vers un objet, sa visibilité, sa sélection)... On peut créer et rajouter des CFs à volonté. Il faut préciser la notion de 'temps' pour calculer les CFs: il s'agit en quelque sorte d'un temps des interactions, c'est-à-dire que l'on met à jour les CFs à chaque interaction significative. Certains CFs ne décroissent pas simplement au cours du 'temps', comme la visibilité d'un objet (cf le CF 'permanent' dans le tableau 1): quand l'objet apparaît, on crée un CF correspondant, avec une certaine valeur, qui n'évolue plus, jusqu'à ce qu'il ne soit plus visible; la valeur du CF devient alors nulle, et il est détruit.

A chaque interaction significative, on va donc créer les CFs correspondants pour les objets concernés, et mettre à jour les CFs déjà créés. Pour calculer la pertinence d'un objet, on va utiliser la formule 1 pour chaque

t	1	2	3	4
CF_t (ponctuel)	3	2	1	0
CF_t (permanent)	1	1	1	0

Table 1: Exemple de l'évolution de la valeur des CFs en fontion du 'temps'.

objet.

$$pertinence \ = \ \sum_{i=1}^{n} \ poids \ (\ CF_t^{\ i} \ ) \qquad (1)$$

Ainsi, l'objet résolvant une référence non-résolue sera donc celui de la bonne classe, dont la pertinence sera la plus élevée. Dans EDWARD, sept CFs ont été implémentés, dont quatre linguistiques et trois perceptuels.

2.3.2 Comparaison avec d'autres algorithmes

Les auteurs de cet algorithme, Huls et al., ont comparé leur algorithme à deux autres méthodes. La première semble très intuitive: elle consiste à résoudre une référence par la dernière entité citée de la bonne catégorie. La seconde méthode est celle de Grosz et Sidner [16]. Elle est contstituée de deux mécanismes séparés, chacun résolvant un type de références. Le premier est appelé 'focusing', et sert à limiter les informations qui doivent considérées pour identifier les référés de certaines classes d'expressions. Une pile est créée avec tous les éléments du discours et les entités qui y sont réliées. La pile est du type LIFO [2]. Le second mécanisme,intitulé 'centering', est utilisé pour la résolution des pronoms. A chaque expression du discours, on associe l'élément central de tous les éléments précédemment cités, selon des critères sémantiques, syntaxiques et concernant les informations du discours. Le fait qu'une entité soit reliée de cette façon à un élément du discours réduit l'espace de recherche des références d'un pronom dans une phrase consécutive. Malheureusement, à la différence du premier modèle simple à implémenter, ce modèle pré-suppose la connaissance d'un certain nombre d'informations qu'EDWARD est incapable de fournir. Le système et ses performances ont donc été simulées sur papier. Enfin, les tests ont été conduits avec cinq utilisateurs normaux, dont l'expérience avec le système était limitée. Ils devaient seulement effectuer un certains nombre de tâches fixées par les auteurs.

Comme vous pouvez le voir dans le tableau 2, le modèle de Huls et al. est le plus efficace, mais les deux autres ont de bons résultats, même le modèle

[2]LIFO: Last In First Out, dernier arrivé, premier sorti

EDWARD	125
Modèle simple	119
Grosz et Sidner	124

Table 2: Nombre de références correctement résolues par les 3 modèles (sur un maximum de 125).

simple. Il existe en effet des cas où l'entité auquel il est fait référence ne soit pas la plus récemment citée, comme dans l'exemple: 'Françoise a prit la voiture de Josiane pour aller à la pèche. Elle est revenu bredouille'. Quant à Grosz et Sidner, leur modèle a échoué sur un cas de double occurence de pronoms, qui n'avait pas été prévu dans le modèle.

Intéressons-nous enfin aux limitations intrinsèques du modèle. En effet, le modèle présenté de Huls et al., ainsi que le modèle simple, ne font aucune prédiction sur les intentions du discours, à la différence de celui de Grosz et Sidner. Or, les intentions du discours jouent un rôle-clef dans l'explication de la structure du discours, la définition de sa cohérence. En outre, tous ces modèles sont incapables de résoudre les 'cataphores', c'est-à-dire des références à des entités qui seront introduites plus tard dans le discours ou les interactions (par exemple dans la phrase 'il sera vainqueur, celui qui...').

Pour conclure sur l'algorithme de résolution des références que nous venons de présenter, et que nous allons utiliser dans notre implémentation (cf partie 3), il donne de bons résultats, est moins gourmant en ressources, et surtout demande une modélisation bien moins lourde que par exemple celui de Grosz et Sidner. De plus, il est plus modulaire, puisque prendre en compte de nouvelles interactions n'exige que la création de nouveaux CFs correspondants, le système de résolution de références à proprement parler restant le même, contrairement au modèle de Grosz et Sidner dans les règles de décisions duquel il faudrait faire des changement explicites.

3 Implémentation

3.1 Présentation du système

Le but du programme que nous avons impémenté est d'illustrer les possibilités des interactions multimodales avec les environnements virtuels. Les différents modes d'interaction que nous avons implémenté sont les suivants:

Le clavier : déplacement dans le monde virtuel.

La souris : outil de pointage.

Le langage : ordres de déplacement (en pseudo-langage,seuls certains mots sont interprétés), demande d'information sur le monde.

Le problème crucial qui nous a intéressé ici, comme nous avons essayé de le montrer dans les parties précécentes, est la question d'un modèle unique de la représentation du monde, accessible à différents clients.

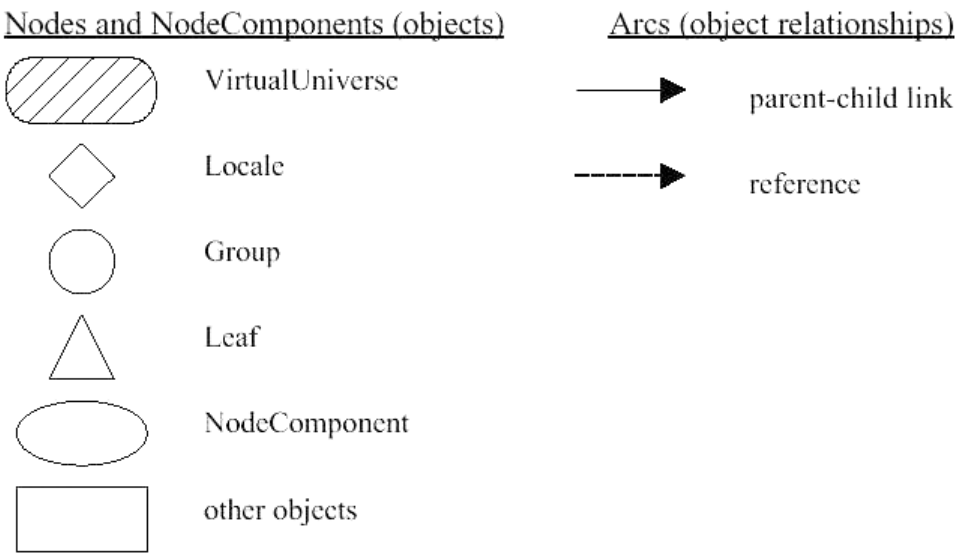

Figure 7: Légende des objets du graphe de scène Java3D de la figure 8.

Nous avons choisi le graphe de scène comme modèle du monde, car il nous semble répondre au moins en partie aux besoins que nous avons défini dans la partie 1.3. Un exemple tiré de la documentation de Java3D, que nous avons utilisé pour développé l'application, est donné par les figures 7 et 8.

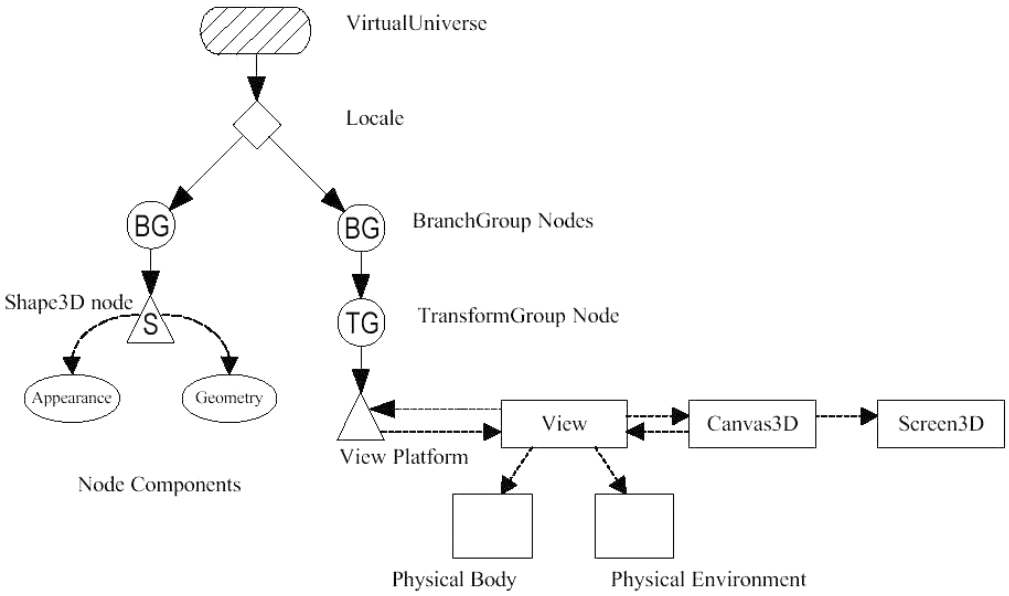

Figure 8: Graphe de scène typique d'un monde Java3D.

Tout d'abord, c'est la forme de construction de scène 3D que les langages de programmation utilisent en général. Il n'y aura donc pas à adapter le modèle du monde sous une autre forme pour le rendu, ce qui diminue les besoins en ressources informatiques de l'application. De plus, il est très simple d'associer une étiquette à chaque objet du graphe de scène susceptible d'être référencé par l'utilisateur. Par conséquent, il sera possible de résoudre les références concernant ces objets. L'inconvénient évident est que le graphe de scène ce saurait suffire pour répondre à des questions plus générales, faire des inférences sur les objets du monde... Il manque une base de connaissance sur le monde, pour laquelle il faudrait ajouter un autre modèle.

Grâce aux modes de communication mis en oeuvre, nous pouvons résoudre deux types de CFs.

CF linguistique : référence directe d'un objet du monde, grâce aux étiquettes associées.
ex: 'existe-t-il une maison ?', 'tourne vers la maison', 'va vers la maison'

CF déïctique : pointage sur un objet du monde.

Cela permet au système de résoudre des références par anaphore, et donc de comprendre des ordres comme 'vas-y' et 'tourne vers lui'. Nous n'avons

implémenté qu'un seul type d'objet susceptible de résoudre ces références, les lieux accessibles du monde.

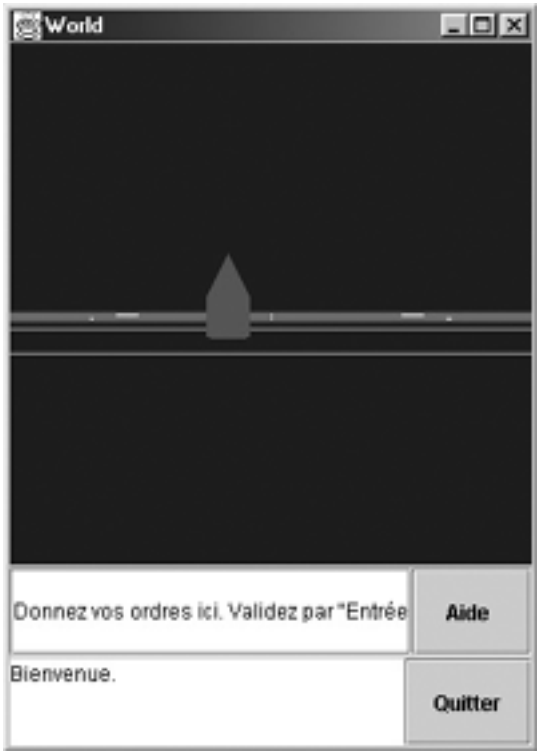

Figure 9: Interface de l'application développée.

3.2 Architecture

Dans notre système, les commandes en langage pseudo-naturel sont saisies au clavier, et non par un système de reconnaissance vocal. En effet, il existe maintenant sur le marché des systèmes de reconnaissance vocal tout à fait performants et ce problème de traitement du signal ne concerne pas le présent travail.

Une autre restriction concerne le fait que nous n'avons pas implémenté de véritable moteur linguistique, et que les ordres interprétés sont donc pseudo-naturels, du type 'va/tourne [préposition: vers] [objet du monde désigné par son étiquette]'... Les étiquettes sont supposées uniques.

Le programme est implémenté en Java, avec l'API Java3D (cf [1] et [2]) pour la partie réalité virtuelle, tout d'abord pour des raisons de portabilité et de rapidité de développement, de nombreuses classes relativement complètes

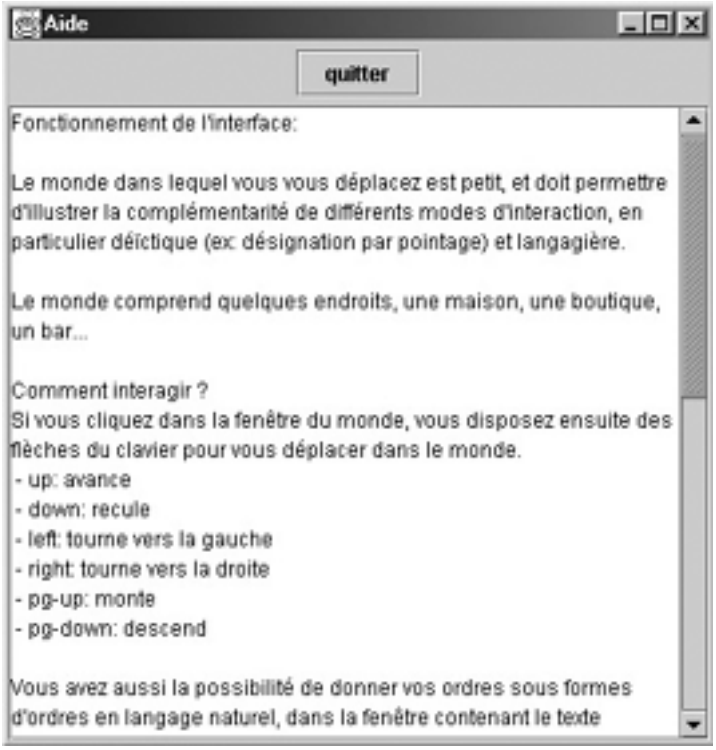

Figure 10: Fenêtre d'aide de l'application développée.

existant déjà, pour l'implémentation de la navigation par exemple, mais aussi du fait de la fidélité de l'implémentation du graphe de scène dans le langage. En outre, il est facile d'intégrer la fenêtre de visualisation dans une interface complète, en particulier avec l'API Swing. Il faut ajouter un bémol sur les classes de haut niveau: en effet, elles facilitent la tâche, car il suffit de les rajouter au bon endroit pour qu'elles fonctionnent, mais on ne peut accéder ensuite aux sous-structures qu'elles contiennent. Par exemple, en utilisant la classe qui implémente la navigation au clavier, on ne peut accéder à la position de la vue. Java3D utilise en particulier des droits sur les structures, et ces classes toutes faites ne permettent pas de lire les informations sur leurs structures. Si on veut être libre, il faut donc refaire ces classes soi-même, à partir des classes de base, car il n'y a pas de classes intermédiaires. Java3D est cependant tout à fait accessible malgré ces difficultés.

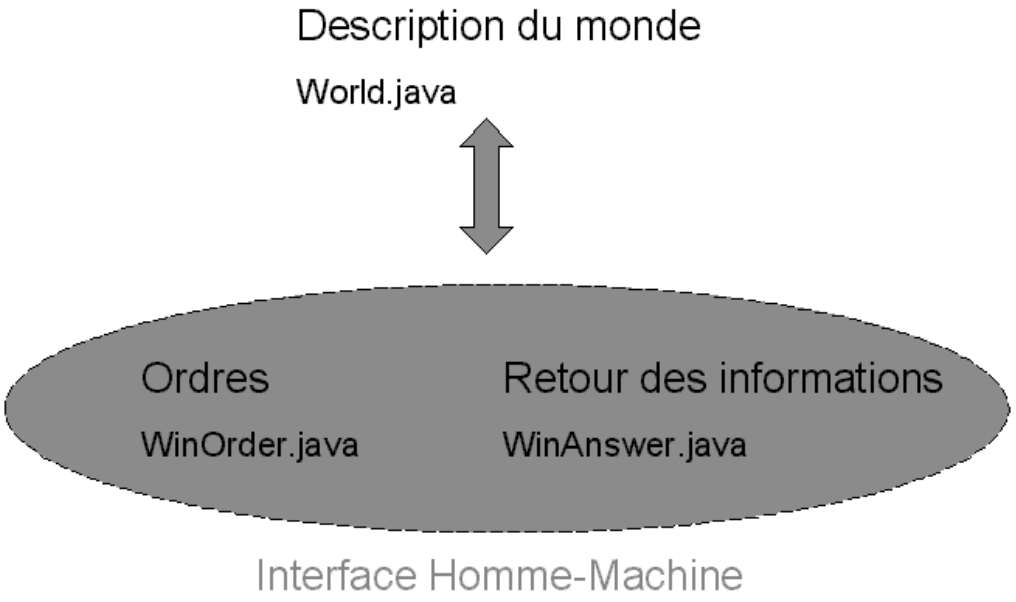

Figure 11: Architecture de l'application développée.

Décrivons maintenant l'architecture du système, présentée sur la figure 11. Elle est divisée principalement en deux parties, le monde, et l'interface graphique, que vous pouvez voir sur les figures 9 et 10:

Description du monde : elle est contenue dans le fichier World.java. Ce fichier est le fichier principal, et comprend le graphe de scène, ainsi que les méthodes permettant de déplacer le point de vue.

Interface graphique : Elle se divise en deux fichiers, WinOrder.java et WinAnswer.java. WinOrder gère la fenêtre de saisie des ordres, et effectue les traitements pseudo-linguistiques, avant d'appeler des méthodes de World pour effectuer le déplacement, et des méthodes de WinAnswer pour afficher dans la fenêtre inférieure des informations sur l'état du système. On a essayé de détailler et expliciter les réponses, en particulier d'erreur, afin que l'utilisateur ne se sente pas perdu en face du système. Enfin, en cliquant sur le bouton 'aide', on instancie un objet de la classe Help, ouvrant une fenêtre affichant le fichier texte d'aide README.txt.

Les détails de l'implémentation sont développés dans l'annexe A. Parlons des objets accessibles dans le monde. Ils instancient des objets de la classe Landmark, tous conçus sur le même modèle géométrique: on peut cependant en changer la couleur et le texte afficher en 3D au-dessus, et évidemment choisir leur position dans le monde, dans le plan horizontal passant par l'origine du monde. Ils comprennent aussi des champs contenant leur étiquette, pour les utiliser dans des ordres textuels, et un tableau contenant les différents CFs actifs concernant un objet. World comprend une méthode qui met à jour les CFs de tous les objets à chaque interaction significative.

3.3 Critiques et améliorations envisagées

Si vous voulez tester l'application, vous trouverez tous les renseignements nécessaires dans les références [23]. Le système implémenté marche de la façon que nous avons décrite. En particulier, il gère les références correctement, permettant d'enchaîner des ordres comme cela est décrit dans le tableau 3.

Utilisateur	Système (Fenêtre du monde)
Existe-t-il un bar ?	Oui, un bar existe. (rien ne se passe)
Tourne vers lui.	Vous etes tourné vers le bar. (rotation sur place vers le bar)
Vas-y.	Vous etes devant le bar. (déplacement devant le bar)
Tourne a gauche.	Déplacement effectue. (rotation de 90 degrés vers la gauche)
Tourne a gauche.	Déplacement effectue. (rotation de 90 degrés vers la gauche)
(clique sur un objet)	Vous avez clique sur la boutique. (rien ne se passe)
Vas-y.	Vous etes devant la boutique. (déplacement devant la boutique)

Table 3: Exemple typique de suites d'ordres et de réponses du système.

Le système est cependant limité. Tout d'abord, nous n'avons pas réussi à créer un déplacement fluide du point de vue, lors des mouvements. Ceci est dû aux problèmes de Java3D évoqués dans la partie 3.2. Le point de vue se retrouve ainsi téléporté d'un endroit à un autre de façon instantanée. De même, nous ne permettons pas la manipulation des objets du monde du fait de la difficulté de développer les classes correspondantes. D'autre part, le nombre de CFs considérés est faible comparé par exemple à EDWARD [15]. On pourrait ainsi en ajouter de nouveaux spatiaux et temporels. D'un point de vue spatial, on pourrait rajouter des CFs liés à la visibilité des objets et leur proximité. Il faudrait aussi permettre la manipulation des objets, la possiblité de les bouger, et permettre des ordres comme 'mets-ça ici !'. En tant que CF temporel, on tient déjà compte des ordres donnés, ce qui est équivalent à lier des CFs aux lieux visités. Il en serait de même si l'on permettait de manipuler des objets par des ordres linguistiques. Un ajout novateur concernerait la mise en place de profils utilisateurs. En effet, on pourrait envisager d'apprendre les habitudes d'une personnes, les lieux fréquemment visités, mais aussi, au lieu d'imposer des déplacements de longueurs fixes, par exemple pour les ordres 'avance' ou recule, régler ces déplacement en fonction de l'utilisateur, rajouter des modifieurs comme 'un peu', 'encore'.

Il y a en outre les problèmes liés au pseudo-moteur linguistique utilisé. Il marche en effet pour ce que nous avons implémenté, mais il serait souhaitable d'utiliser un vrai, car il est toujours frustrant d'utiliser vainement un synonyme de l'ordre compris par le système. Dans la même perspective, comme insistent les constructeurs du projet Diverse du SICS [7], il est important de développer des messages d'erreurs et plus généralement des messages sur l'état du système pour aider la compréhension de l'utilisateur. On ne peut prétendre développer un système cherchant à prendre en compte les capacités de l'utilisateur en utilisant le langage naturel, et le laisser face à une interface muette.

Enfin, le graphe de scène en tant que modèle du monde est un choix tout à fait justifiable, bien que limité. Pour implémenter des raisonnements de haut niveau, une solution naturelle serait un réseau sémantique modélisant des connaissances générales sur le monde. Pour résoudre des ordres complexes, comme 'va chercher le journal', on chercherait par exemple le lieu de déplacement comme l'entité de la catégorie 'lieu' la plus proche, c'est-à-dire dont l'évocation sera la plus élevée, dans le réseau sémantique activé par l'entité 'journal'. Nous pensons même que ce réseau sémantique pourrait faire partie du profil utilisateur dont nous avons parlé plus haut, modélisant ainsi les goûts de l'utilisateur.

Pour conclure, notre système, avec ses imperfections, constitue cependant une illustration cohérente des atouts d'une interface pour monde virtuel pro-

posant une interaction par le langage naturel, un premier exemple auquel il faudrait consacrer plus de temps pour développer de nouvelles capacités, parmi les pistes envisagées dans ce travail, et au-delà.

4 Conclusion

Comme nous venons de le montrer, les interfaces graphiques utilisant le langage naturel sont très prometteuses. Comme tout ce qui touche le traitement du langage naturel, elles sont aussi complexes à mettre en oeuvre, en particulier en ce qui concerne la résolution des réferences, même si des algorithmes efficaces et relativement simples ont fait leur apparition.

Le second point important implique la réflexion sur la représentation des mondes virtuels, dans un modèle de type client-serveur, une architecture ouverte permettant d'ajouter de nouveaux modules. Le graphe de scène que nous avons utilisé donne des résultats satisfaisants, mais est limité en ce qui concerne les possibilités de raisonnement de haut niveau. Il faut ajouter un modèle de connaissances générales. Ces questions constituent un enjeu capital pour de nombreuses applications de réalité virtuel, afin de rendre leurs interfaces plus simples et naturelles pour les utilisateurs.

5 Remerciements

Je tiens à remercier Alain Grumbach pour son encadrement, et Pierre Nugues pour ses réponses précises à mes nombreuses questions.

References

[1] Page de l'API Java3D, chez Sun, http://java.sun.com/products/java-media/3D/index.html.

[2] Getting Started with the Java 3D API, Dennis J. Bouvier.

[3] Page du projet Wordseye de ATT, http://www.research.att.com/projects/wordseye/

[4] Wordseye, an Automatic Text-To-Scene Conversion System, Bob Coyne, richard Sproat, ATT Labs-Research, Siggraph 2001.

[5] Page du projet Microsoft, Persona, http://www.research.microsoft.com/research/ui/persona/home.htm.

[6] Lifelike Computer Characters, The Persona Project at Microsoft Research, Gene Ball, Dan Ling, David Kurlander, John Miller, David Pugh, Tim Skelly, Andy Stankosky, David Thiel, Maarten Van Dantzich and Trace Wax, Microsoft Research, One Microsoft Way, Redmond, Software Agents, Jeff Bradshaw Ed., MIT Press, 1997.

[7] Page du projet Diverse (DIVE Real time Speech Enhancement), du SICS (Swedish Institute of Computer Science), http://www.sics.se/nlp/diverse.html.

[8] Interaction Models, Reference, and Interactivity in Speech Interfaces to Virtual Environments, Jussi Karlgren, Ivan Bretan, Niklas Frost, Lars Jonsson, Swedish Institute of Computer Science, KISTA, Stockholm, Sweden.

[9] Achieving Virtual Presence with a Semi-Autonomous Robot Through a Multi-Reality and Speech Control Interface, Kristian T. Simsarian, Jussi Karlgren, Ivan Bretan, Niklas Frost, Lars Jonsson, Lennart E. Fahlen, Emmanuel Frécon, Swedish Institute of Computer Science, KISTA, Stockholm, Sweden.

[10] Using Surface Syntax in Interactive Interfaces, Jussi Karlgren, Ivan Bretan, Niklas Frost, Swedish Institute of Computer Science, KISTA, Stockholm, Sweden.

[11] Natural Language Interfaces to Virtual Reality Systems, page principale du projet Nautilus, http://www.aic.nrl.navy.mil/~severett/vr.html

[12] Creating Natural Language Interfaces to VR Systems: Experiences, Observations and Lessons Learned, Stephanie S. Everett, Kenneth Wauchope, Manuel A. Pérez-Quiones, Navy Center for Applied Research in AI, Naval Research Laboratory, Washington, publié initialement dans Future Fusion: Application realities for the virtual age, Proceedings of VSMM98, 4th International Conference on Virtual Systems and Multimedia, Vol. 2, pp. 469-474.

[13] A Natural Language Interface for Virtual Reality Systems, Stephanie S. Everett, Kenneth Wauchope, Manuel A. Pérez, Navy Center for Applied Research in Artificial Intelligence, US Naval Research Laboratory, Washington.

[14] Introduction à la linguistique française II, Jacques Moeschler, Université de Genève.

[15] Automatic Referent Resolution of Deictic and Anaphoric Expressions, Carla Huls, Edwin Bos et Wim Claassen, Computational Linguistics, volume 21, numéro 1, 1991.

[16] Attention, intentions, and the structure of discourse, Barbara J. Grosz et Candace L. Sidner, Computational Linguistics, volume 12, 1986.

[17] Page du projet Ulysse, dirigé par Pierre Nugues, http://www.ensicaen.ismra.fr/ nugues/recherche.html.

[18] Agents informatiques interactifs, Applications linguistiques et biomédicales, Pierre Nugues, Dossier d'habilitation à diriger des recherches.

[19] Un agent conversationnel pour naviguer dans les mondes virtuels, Christophe Godéreaux, Pierre-Olivier El Guedj, Frédéric Revolta et Pierre Nugues, Caen.

[20] Cognitive Consequences of Spatial Language, Geoffrey S. Hubona, Stephanie Everett, Elaine Marsh, Kenneth Wauchope.

[21] Pictorial-verbal tools for conveying routes, Barbara Tversky, International Conference COSIT'99, Proccedings, p51-72.

[22] Spatial Information Theory, Cognitive and Computational Foundations of Geographic Information Science, International Conference COSIT'99, Proccedings, Springer.

[23] Les sources de l'application, ainsi que tous les documents, sont disponibles à l'adresse suivante: http://perso.enst.fr/~saunier/mem/.

A Analyse du programmes

Le système d'axe en Java3D est décrit dans la figure 12. Le graphe de scène de l'application est représenté dans la figure 13.

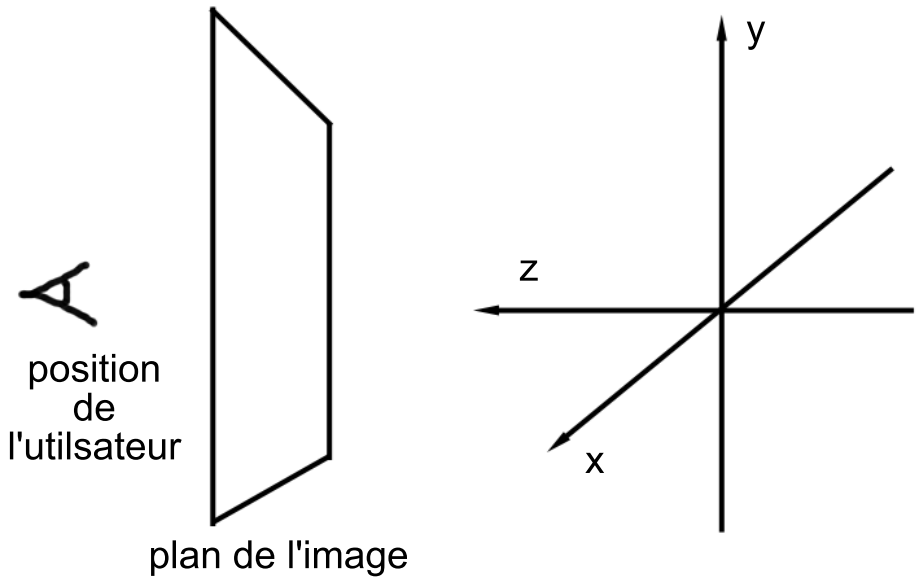

Figure 12: Orientation du repère et position du plan de l'image dans Java3D.

A.1 Description du monde

Le monde est essentiellement créé par la classe World, qui instancie des objets de la classe Landmark et MyUniverse, et utilise le comportement PickMouse pour permettre de cliquer sur des objets.

A.1.1 Classe World

Attributs

private MyUniverse avalon : cet attribut est de la classe MyUniverse qui contient les mêmes éléments que SimpleUniverse, hormis qu'on a les droits de déplacer le point de vue.

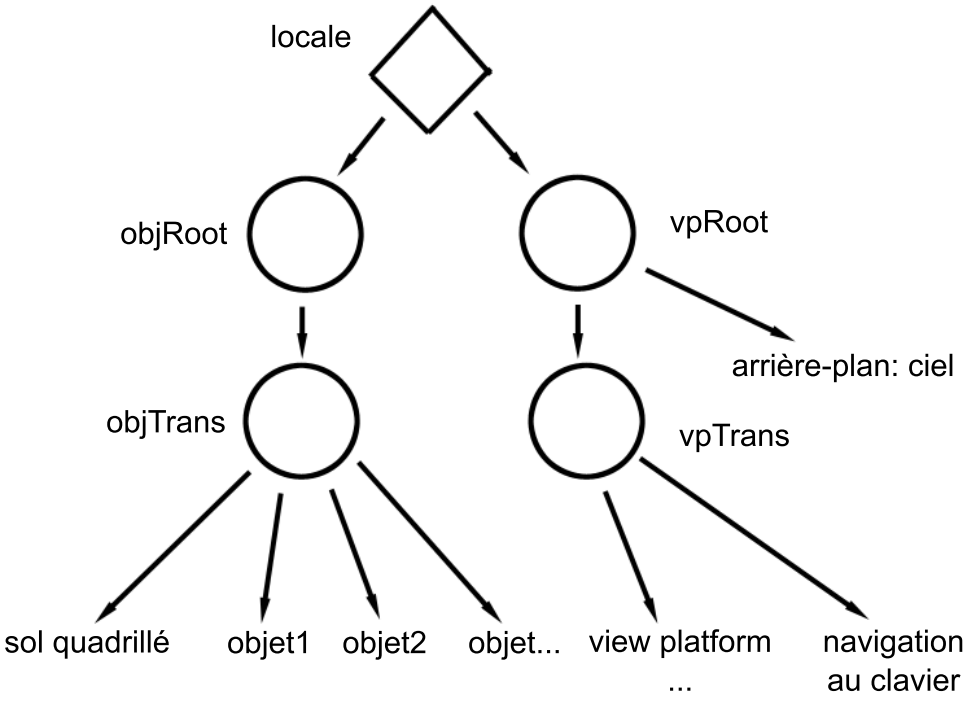

Figure 13: Graphe de scène de l'application.

Canvas3D can : élément indispensable d'un Univers Virtuel minimal.

WinOrder panel1 : cet attribut de la classe WinOrder fait partie de l'interface graphique.

WinAnswer panel2 : cet attribut de la classe WinAnswer fait partie de l'interface graphique.

KeyNavigatorBehavior keyNavBeh : cette classe implémente la navigation au clavier.

Landmark place[ ]: tableau des objets du monde.

static int nbPlaces :nombre d'objets dans le monde.

static int table[[]]: tableau contenant les valeurs des CFs en fonction du temps.

TransformGroup objTrans : TransformGroup auquel sont attachés tout les objets du monde (avec les attributs: Transform3D trans3D, Vector3f vect, double angle = 0).

TransformGroup vpTrans : TransformGroup de la plateforme (avec les attributs: Transform3D vpTrans3D, Vector3f vectView, double angleView = 0, Vector3f oldVectView).

Méthodes

Shape3D createLand() : cette méthode crée un sol quadrillé.

public BranchGroup createSceneGraph() : cette méthode crée le graphe de scène.

public int findPlace(String goal) : cette méthode retourne le numéro du lieu dans le tableau place, dont l'étiquette est goal, et -1 si on ne trouve pas.

public int findPlace(BranchGroup goal) : cette méthode retourne le numéro du lieu dans le tableau place, dont l'adresse est goal, et -1 si on ne trouve pas.

public void turn(int numLoc) : cette méthode tourne la vue vers l'objet place[numLoc].

public void move (int numLoc) : cette méthode place le point de vue devant l'objet place[numLoc].

public void applyMvt() : cette méthode applique les mouvements calculés par les méthodes move() et turn(), dont les paramètres sont vectView pour la translation et angleView pour la rotation selon l'axe Y.

public void updateSalience(int numLoc, int numCF) : cette méthode fait évoluer les CFs de chaque objet à chaque interaction significative de l'utilisateur, et crée un CF, dont les valeurs sont table[numCF][], pour l'objet place[numLoc].

public int findReference() : cette méthode renvoie le numéro de l'objet dans le tableau place le plus pertinent en cas de reference par anaphore.

public World() : constructeur vide. En effet, World est une applet, qui grâce à la classe JMainFrame peut être utilisé comme une application.

public void init() : cette méthode est appelée au lancement de l'applet (joue le rôle du constructeur).

public void destroy() : cette méthode est appelée lorsque l'on quitte l'application.

public static void main(String[args)]: méthode main permettant de faire tourner l'applet comme une application.

A.1.2 Classe MyUniverse

Attributs

Canvas3D canvas : élément indispensable d'un Univers Virtuel minimal(avec VirtualUniverse universe, Locale locale, View view).

TransformGroup vpTrans : TransformGroup auquel est attaché la point de vue. On permet d'en changer les paramètres afin de déplacer le point de vue, par la méthode applyMvt() de World.

Méthodes

public MyUniverse(Canvas3D c) : le constructeur instancie les attributs vus plus haut, ainsi qu'un arrière-plan de couleur bleue foncée.

public void addBranchGraph(BranchGroup bg) : cette méthode permet d'ajouter le graphe de scene du monde.

public View view() : cette méthode retourne view.

public TransformGroup getViewTransform() : cette méthode retourne la transformation du point point de vue, ce qui permet ensuite les déplacements.

A.1.3 Classe Landmark

Cette classe est celle de tous les objets accessibles du monde. Ils ont tous la même géométrie, et sont configurables par certains attributs.

Attributs

private Appearance LandAppear : cet attribut contient tous les éléments d'apparence communs à tous ces objets. De plus, on peut personnaliser la couleur.

int valueSalience[ ]: ce tableau contenant les CFs et leur 'âge'. Il s'agit d'une liste, avec deux entiers pour chaque CF actif: le numéro du CF, et son 'âge', permettant de connaître sa valeur, dans la classe World, table[CF][âge].

String label : cet attribut est l'étiquette de l'objet, utilisée dans les ordres.

int genre : cet attribut contient le genre de l'étiquette (0: masculin; 1: féminin), pour pouvoir afficher des phrases correctes.

Les attributs suivants contiennent les caractéristiques de l'objet affiché dans le monde 3D.

String text : texte affiché en 3D.

Color3f color : couleur de l'objet.

float posX : position de l'objet en X

float posZ : position de l'objet en Z

Méthodes

private Appearance createAppearance() : il s'agit uniquement de rajouter la couleur à l'objet.

public Vector3f getPosition() : cette méthode retourne la position de l'objet par rapport à l'origine du monde sous la forme d'un vecteur.

A.1.4 Classe PickMouse

Cette classe hérite de PickMouseBehavior et permet d'implémenter le pointage sur les objets, ce qui correspond à un CF particulier.

Attributs

World main : cet attribut permet de connaître la classe principale afin d'en appeler les méthodes, ou les méthodes de ses attributs.

Méthodes

public PickMouse(Canvas3D canvas,...) : ce constructeur de la classe fait essentiellement appel au constructeur de PickMouseBehavior.

public void updateScene(int xpos, int ypos) : cette méthode est appelée lorsque l'on clique avec la souris dans la fenêtre du monde, avec en paramètre la position du curseur dans cette fenêtre. On peut alors utiliser les méthodes d'un objet de la classe PickCanvas pour récupérer l'adresse de l'objet le plus proche dans la direction de pointage.

A.2 Interface graphique

L'interface graphique est divisée en trois classes, WinOrder pour les ordres et le bouton d'aide, WinAnswer pour afficher des informations sur l'état du système, et le bouton 'quitter', et Help, pour la fenêtre d'aide. Un problème commun à toutes ces classes est que l'on a supprimé toute accentuation, aussi bien dnas les ordres que dans les messages du système et les étiquette des objets, car le système ne les comprend pas et fait des erreurs.

A.2.1 Classe WinOrder

Attributs

JTextField order : fenêtre permettant de saisir du texte.

JButton help : bouton d'aide.

World main : cet attribut permet de connaître la classe principale.

Méthodes

public WinOrder(World w) : ce constructeur de la classe met les différents éléments de cette partie de la fenêtre en place.

public void parse1(String order) : cette méthode constitue la première mathode de parsage du pseudo-moteur linguistique. Elle oriente vers le bon traitement en fonction du permier terme de la chaîne de caractère order. Toutes ces méthodes fonctionne selon le même principe de reconnaissance des mots les uns après les autres dans un certain ordre.

public void parseVa(String text) : cette méthode permet le traitement des ordres du type 'va vers [objet]' et 'vas-y'.

public void parseTourne(String text) : cette méthode permet le traitement des ordres du type 'tourne vers [objet]' et 'tourne a gauche/droite'.

public void parseExiste(String text) : cette méthode permet le traitement des ordres du typs 'existe-t-il [objet] ?'.

public void actionPerformed(ActionEvent e) : cette méthode permet de détecter lorsque l'on a fini de taper dans la zone de texte, et que l'on appuie surla touche 'entrée', ou lorsque l'on a appuyé sur le bouton 'aide'.

A.2.2 Classe WinAnswer

Attributs

JTextArea message : on affiche dans cette zone de texte les message sur l'état du système, d'erreurs et de déplacement.

JButton quit : bouton 'quitter'.

World main : cet attribut permet de connaître la classe principale.

Méthodes

public WinAnswer(World w) : ce constructeur de la classe met les différents éléments de cette partie de la fenêtre en place.

public void giveAnswer(String text) : cette méthode permet d'afficher le message contenu dans la variable text.

public void giveAnswer(int text) : cette méthode affiche les messages d'erreurs, 0 si le système n'a pas compris l'ordre, et 1 si l'utilisateur n'a pas cliqué sur un objet.

public void giveAnswer(int text, int numLoc) : cette méthode affiche des messages formattés de réponse aux actions de l'uilisateur, contenant en général le nom de l'objet numLoc.

public void actionPerformed(ActionEvent e) : cette méthode détecte losrque l'on clique sur le bouton 'quitter', et ferme l'application.

A.2.3 Classe Help

Attributs

JTextArea zone : zone de texte dans laquelle on affiche l'aide.

JFrame fen : fenêtre d'aide.

JPanel barre : panneau contenant le bouton 'quitter'.

JButton butquit : bouton 'quitter'.

Méthodes

Help() : ce constructeur met en place les éléments de l'interface et affiche le texte contenu dans le fichier README.txt.

public void actionPerformed(ActionEvent e) : cette méthode détecte lorsque l'on clique sur le bouton 'quitter', et ferme la fenêtre.